University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Goran Molnar

METAHEURISTICS FOR PROBLEMS WITH LIMITED BUDGET OF EVALUATIONS

DOCTORAL THESIS

Zagreb, 2020

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Goran Molnar

METAHEURISTICS FOR PROBLEMS WITH LIMITED BUDGET OF EVALUATIONS

DOCTORAL THESIS

Supervisor:
Professor Domagoj Jakobović, PhD

Zagreb, 2020

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Goran Molnar

METAHEURISTIKE ZA PROBLEME S OGRANIČENIM BROJEM EVALUACIJA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Domagoj Jakobović

Zagreb, 2020.

Doktorski rad izrađen je na Sveučilištu u Zagrebu,
Fakultetu elektrotehnike i računarstva, na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Mentor: prof. dr. sc. Domagoj Jakobović

Doktorski rad ima: 249 stranica

Doktorski rad br.: ___________

About the Supervisor

Domagoj Jakobovic received B.Sc degree in December 1996 and MS degree in December 2001 in Electrical Engineering. He was awarded the "Josip Loncar" best student of the year award in 1996.

Since April 1997, he is a member of the research and teaching staff at the Department of Electronics, Microelectronics, Computer and Intelligent Systems of Faculty of Electrical Engineering and Computing, University of Zagreb. He received MS degree in December 2001 on the subject of the forward kinematics of Stewart parallel mechanisms. He received PhD degree in December 2005 on the subject of generating scheduling heuristics with genetic programming.

O mentoru

Domagoj Jakobović je diplomirao u prosincu 1996. godine, a magistrirao u prosincu 2001. u polju elektrotehnike. Dobitnik je nagrade "Josip Lončar" za najboljeg studenta na godini, koju je primio 1996.

Od travnja 1997. je član istraživačkog i nastavnog osoblja na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu. Diplomirao je u prosincu 2001. godine, na temu unaprijedne kinematike Stewartovih paralelnih mehanizama. Doktorirao je u prosincu 2005. na temu generiranja heuristika za izradu rasporeda sati korištenjem genetskog programiranja.

To my father Predrag, who showed me that science is the most beautiful way to get to know the world around me.

Acknowledgements

I would like to thank my supervisor, Prof. Domagoj Jakobović, for the wonderful collaboration while working on this PhD. What at times seemed like an obstacle that cannot be overcome, under his guidance gradually transformed into a series of small wins that persistently led me to the completion of this work. I feel gratitude for all the knowledge, constructive comments, support and perseverance while we were on this adventure.

Further, I would like to thank Prof. Gonçalo Correia, who supervised my work in the InnoVshare project while I was a visiting researcher at the University of Coimbra in Portugal. I am thankful for all the help in discovering the fascinating world of optimisations in transportation and carsharing, as well as the persistence and motivation that led to our published papers. I would also like to thank my colleague Diana Jorge for the successful work in the carsharing project and the paper we worked on.

Special thanks to the PhD committee members for this thesis: Prof. Marin Golub, Prof. Marko Čupić and Prof. Tomislav Rolich. I feel grateful for your comments and feedback on the work.

I started working on optimisation algorithms back in my Master studies with Prof. Marko Čupić, Prof. Bojana Dalbelo Bašić, and Prof. Domagoj Jakobović. Their gentle guidance and enthusiasm encouraged my first contact with the world of science and optimisation. In addition, I owe gratitude to my university colleagues Vatroslav Dino Matijaš, Vjera Omrčen, Tomislav Herman and Zlatko Bratković, with whom I participated in scheduling projects at the Faculty of Electrical Engineering and Computing in Zagreb, Croatia. Thanks for your kindness and support.

I would like to thank my colleagues at Asseco SEE Ltd in Zagreb, Matija Pavelić, Danko Kozar and Ivana Horvat Čuka, where for the first time I worked on a commercial workforce optimisation project. I would also like to thank Suzana Čanađija, Tea Pavešić, Bojan Brezlan and Dubravko Dobra on detailed feedback provided for the schedule prototypes. This work allowed me to realise the potential of practical applications of metaheuristics and further inspired my decision to pursue my PhD.

In addition, I need to express my deep gratitude to Mirjana Grubiša and Đurđica Tomić Peruško, the Administrators for Doctoral study programme at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. Thank you for your constructive feedback and patience during numerous challenges in the course of my PhD. Your positive attitude and guidance were of great help and encouragement.

From a more personal perspective, I must thank Karlo Kralj, Boris Vrdoljak, Ivica Radović, Marijo Marjanović, Luka Franov, Maja Grubić, Emil Drkušić, Maja Belušić, Tomislav Tomašić, Dražen Lučanin, Barbara Blečić, Tomislav Herman, Ana Legradi, Anderson Rodrigues, Diego Giménez, Ivan Baraldi, João Batista, Nuno Duarte Gregorio, Henrique Sengo Cordeiro, Pedro Leitão, João Santos, Gonçalo Cholant, Joana Pedro Ferreira, Tiago Bandeira, and numerous other friendly people for their generous support, encouragement and lots of patience with me while I was working on this thesis.

Lastly, I thank my family for all the love and support while working on this thesis, my mother Ančica, my father Predrag and my brother Darjan.

Zagreb, February 2020

Summary

Metaheuristic techniques are an essential set of optimisation techniques with broad applications in numerous problems of great practical importance. Despite their success, using metaheuristics also has notable drawbacks: they are highly complex algorithms whose implementation process still lacks a formal development methodology. Their development is expensive as it requires highly trained experts, considerable time and computational resources. This work proposes a bottom-up development methodology for metaheuristic development, based on a component-based view of metaheuristics and gradual addition of more complex elements. The development methodology has the potential to reduce development time and costs while providing high-quality results. This development methodology was experimentally validated on three difficult problems: (1) call centre workforce scheduling, (2) carsharing reservations optimisations, (3) carsharing variable trip pricing problem. The second and third problems were especially difficult given their resource-intensive objective function, that requires long evaluation times, this way restricting the evaluations budget. Solutions to these two problems are to the best of the author's knowledge, the first applications of the iterated local search metaheuristic on problems with a limited budget of evaluations. Further, these solutions do not use surrogate modelling, which is common practice with such problems. The work concludes with a set of guidelines for surrogate-free solving of optimisation problems with a limited budget of evaluations, based on the experiences solving these two problems.

Keywords: metaheuristics, development methodology, Iterated local search, workforce scheduling, carsharing, transportation optimisation

Prošireni sažetak

U svakodnevnom smislu, riječ *optimum* znači skup najpovoljnijih mogućih uvjeta ili okolnosti, najveći mogući stupanj nečega, najbolji rezultat koji se može ostvariti uz zadane ili pretpostavljene uvjete. Glagol *optimizirati* znači usavršiti do razine najboljeg, izabrati najbolji od svih mogućih izbora, učiniti nešto najsavršenijim mogućim. Riječ *optimizacija* je postupak traženja optimuma, čin, postupak ili metodologija usavršavanja nečega do najviše moguće razine.

Optimizacija je sveprisutna. Genijalni primjeri optimizacije mogu se naći posvuda u prirodi. Fizički sustavi prirodno teže stanju minimalne energije. Paukove mreže su iznimno optimizirane strukture, počevši od paukove svile koja se sastoji od vlakana visokih performansi, do njihove strukturne mehanike. Pahuljasta pera koja se nalaze u "padobranu" maslačka pokazuju savršeno podešena svojstva koja istovremeno omogućuju da ih vjetar nosi na daleke udaljenosti ali i minimizira količinu potrebnog materijala. Donošenje dobrih odluka važno je i za ljude. "Koju životinju loviti od svih životinja u krdu", "što je najbolje učiniti kada nekoga napadne tigar" ili "gdje je najbolje mjesto za izgradnju nastambe" tek su neki primjeri složenih odluka koje su ljudi uspješno rješavali, vođeni instinktom i znanjem koje se prenosilo iz generacije u generaciju.

Pojavom znanosti, optimizacija je postala u sve većoj mjeri formalni proces koji se istraživao korištenjem sve naprednijih, rigoroznih istraživačkih procesa. Problemi optimizacije proučavani su još od antike, te su se tijekom stoljeća njima bavili znanstvenici raspršeni u više područja, većinom matematike i fizike. Istovremeno, optimizacija je i područje u kojem su brojni izumitelji, inženjeri i programeri inovirali brojne tehnike. Danas probleme optimizacije proučava *matematička optimizacija*, područje matematike i računarske znanosti. Pri tom se problemi optimizacije najčešće postavljaju jezikom matematike ili nekim drugim načinom formalne definicije, a za njihovo rješavanje usavršen je velik broj raznovrsnih metoda i procedura.

Važan poticaj razvoju optimizacije pružio je razvoj računala. Dvadeseto stoljeće i početak dvadeset prvog je vrijeme iznimno brzog razvoja te su se početni velike nezgrapne naprave brzo razvile u umrežen ekosustav međusobno povezanih uređaja koji stanu u džep. Porast računalne snage i mogućnosti raznovrsnih primjena bili su dio tog iznimno brzog razvoja.

Prva računala nalazila su se pretežno u istraživačkim institucijama i koristila se većinom za matematičke proračune, a njihovo upravljanje i programiranje bilo je rezervirano tek za uzak krug inženjera i znanstvenika sa specijaliziranim obrazovanjem. Razvojem tehnologije, mogućnosti primjene računala postajale su sve šire te su postupno računala počela postajati sposobna za sve više aktivnosti koje su se smatrale tipično "ljudskim", poput igranja šaha, prepoznavanja lica, usmjeravanja vozila i brojnih drugih. Početak 2020 godine vrijeme je kad se računala nalaze gotovo posvuda, a zbog jednostavnosti za korištenje dostupni su gotovo svim građanima.

Još 1950. u svom članku "Strojevi koji računaju i inteligencija", Alan Turing je naslutio golemi potencijal računala. Taj rad je utabao trag za kasniji razvoj nekoliko disciplina računarske znanosti ali i postavio brojna filozofska pitanja poput "mogu li strojevi misliti". Uz brojne druge doprinose, u radu se postavlja ideja simulacije evolucije u računalu kao mogućnost za izradu univerzalnog pristupa rješavanju različitih problema. Tradicionalni pristup korištenju računala zahtijevao je ljude koji bi željene funkcionalnosti prevodili u strojni jezik, a koje bi računala tek ponavljajuće izvršavala. Nasuprot takvom pristupu, Turing je predložio razvoj računala koja bi put od problema do rješenja izvršila samostalno, pri tom stvarajući nove ideje i učeći. Navedena područja u suvremenom se računarstvu nazivaju strojnim učenjem i evolucijskim računarstvom. Navedeni rad bio je daleko ispred svoga vremena te je još dugo nakon Turingovog rada računala bila nedovoljno razvijena da bi omogućila razvoj i primjenu tih ideja.

Ipak, šezdesete godine 20. stoljeća i početak sedamdesetih su donijele su veliki optimizam, pobuđen ranim uspjehom na jednostavnim problemima. Pojavljuje se algoritam povratne propagacije pogreške (*engl.* backpropagation algorithm), koji omogućava razvoj umjetnih neuronskih mreža, inspiriranih načinom na koji je organiziran ljudski mozak i koje korištenjem navedenog algoritma dobivaju mogućnost učenja ili treniranja ponašanja koja se od njih očekuju. Mnogi istraživači se bave i začetkom evolucijskog računarstva koje je u suvremenom obliku populariziralo John Holland 1975. te se simulacija evolucije u računalu počinje intenzivno proučavati kao moguć univerzalan postupak rješavanja problema. Genetski algoritam će kasnije biti prepoznat kao jedna od prvih tehnika rješavanja optimizacijskih problema koji se danas nazivaju *metaheuristike*.

Veliki i ponekad neutemeljeni optimizam je splasnuo dolaskom razdoblja koje se naziva "zima umjetne inteligencije" (*engl.* AI winter) sedamdesetih i osamdesetih. Djelomično zbog nedovoljno razvijenog hardvera, a djelomično i potaknuto razvojem teoretske računarske znanosti, postalo je jasno da će brojni teški problemi još dulje vrijeme biti izvan dosega rješavanja na računalu. Primjer takvih iznimno teških problema su $\mathcal{NP}$-teški problemi. Taj razred je tijekom sedamdesetih godina dvadesetog stoljeća identificiran kao razred problema za koje ne znamo učinkovite načine rješavanja, a ne znamo niti je li uopće moguće postojanje takvih učinkovitih tehnika. Unatoč desetljećima truda najbriljantnijih znanstvenika i programera, na dana pitanja odgovori nisu poznati te je problem postojanja učinkovitog algoritma za $\mathcal{NP}$-teške probleme prepoznat kao jedno od najvećih neodgovorenih pitanja suvremenog računarstva, poznat i pod nazivom $\mathcal{P}? = \mathcal{NP}$ problem.

U navedeni razred teških problema ubrajaju se i brojni problemi optimizacije čije je rješavanje iznimno važno u praktičnim primjenama. Raznovrsne optimizacije procesa u prometu, izrada rasporeda smjena u poduzeću, pakiranje tereta tako da zauzme minimalan mogući skladišni prostor tek su neki primjeri. Budući da za takve probleme ne postoji poznat algoritam koji dovoljno brzo nalazi njihov optimum, za njihovo rješavanje se u praksi koriste *približne tehnike*, koje ne garantiraju pronalazak optimuma ali su brze u pronalasku rješenja koja su blizu optimalnog. U približne tehnike se ubrajaju i metaheurističke tehnike koje su predmet proučavanja ove doktorske disertacije. Metaheurističke tehnike su općeniti algoritamski razvojni okviri koji se mogu primijeniti na širok skup problema.

Tijekom posljednjih desetljeća, razvijen je velik broj metaheuristika. Neke se temelje na korištenju vrlo općenitih zdravorazumskih strategija rješavanja problema. Neke od takvih strategija su načelo “dok god se približavaš cilju, nastavi raditi male pomake” ili “ako mali pomak ne pomaže, napravi veliki pomak”. Navedene jednostavne ideje zajedno čine temelj metaheuristike ponavljajuće lokalne pretrage (*engl.* iterated local search). Drugi razred metaheuristika se temelji na računalnoj simulaciji prirodnih procesa. Primjerice, genetski algoritam koristi simulaciju prirodnog procesa evolucije kako bi postepeno razvio sve bolja rješenja zadanog problema. Slično, metaheuristika simuliranog kaljenja se temelji na oponašanju procesa kaljenja metala, koji je proučavan u području metalurgije te se koristi pri obradi metala, za poboljšanje njegovih svojstava.

Metaheurističke tehnike važan su skup tehnika optimizacije sa širokom primjenom u brojnim problemima velike praktične važnosti. Unatoč njihovim uspješnim primjenama, korištenje metaheuristika nosi i neka nepoželjna svojstva: radi se prije svega o složenim algoritmima za čiju implementaciju još uvijek ne postoje formalne razvojne metodologije. Njihov je razvoj skup, traje dugo, zahtijeva visoko obrazovane stručnjake i znatne računalne resurse. Uz to, trenutno ne postoje smjernice koje pružaju podršku pri izboru neke od velikog broja razvijenih metaheurističkih tehnika. Učinkovit razvoj npr. genetskog algoritma je složen postupak zbog velikog broja komponenti algoritama te složenosti njihova povezivanja u učinkovitu cjelinu.

Ovaj rad predlaže *bottom-up* metodologiju razvoja metaheuristika koja se temelji na rastavu metaheuristika na komponente. Implementacija započinje razvojem najjednostavnijih elemenata i nastavlja se postupnim dodavanjem složenijih. U svakom koraku implementacije, algoritam je funkcionalan te može dati potpuna rješenja. Prednost tog svojstva jest mogućnost davanja početnih rezultata u vrlo ranoj fazi razvoja, i veća agilnost razvoja. Proces se nastavlja dodavanjem složenijih komponenata i njihovim povezivanjem u sve složenije metaheuristike, sve dok se ne postignu dovoljno dobri rezultati za primjenu. Primjena razvijene metodologije može smanjiti trajanje i troškove razvoja, ali i dalje pruža kvalitetne rezultate.

Praktična primjenjivost navedene metodologije je eksperimentalno provjerena prilikom implementacije optimizacijskih algoritama za tri teška problema: (1) izrade rasporeda rada djelatnika pozivnog centra, (2) optimizacije rezervacija carsharing sustava, (3) rješavanja problema određivanja varijabilnih cijena carsharing usluge. Sva tri problema uspješno su riješena. Postupak dodavanja složenijih operatora zaustavljen je već pri izradi najjednostavnijih metaheuristika: GRASP (nasumična pohlepna prilagodljiva procedura pretraživanja, *engl.* greedy randomized adaptive search procedure) i ponavljajuće lokalne pretrage. Navedeni rezultati indikator su prilagodljivosti jednostavnih metaheuristika te potkrjepljuju tezu da se dobri rezultati mogu postići i bez dugotrajnog rada potrebnog za implementaciju složenijih algoritama kao što su genetski algoritam i algoritam mravlje kolonije.

Problem izrade rasporeda rada djelatnika pozivnog centra sastoji se od traženja optimalnih radnih vremena za djelatnike s ciljem što je moguće bržeg javljanja na dolazne pozive. Pozivni centri se sastoje od većeg broja educiranih djelatnika, koji se nazivaju *agenti*. Kad korisnik uputi poziv na telefonski broj poduzeća, njegov poziv bit će preusmjeren prvom slobodnom djelatniku, a do tad će korisnik čekati. Razvijen sustav podržao je vrlo složen skup nekoliko desetaka pravila, počevši od osnovnih kao što su radno vrijeme, do složenijih pravila koja služe povećanju zadovoljstva djelatnika, npr. izbjegavanje nepopularnih smjena. Velik broj pravila (*engl.* constraints) bio je izazovan za rješavanje. Za traženje najboljih rasporeda su implementirane dvije metaheuristike: ponavljajuća lokalna pretraga i GRASP, u skladu s metodologijom predloženom u ovoj doktorskoj disertaciji. Već te dvije jednostavne metaheuristike pružile su zadovoljavajuće rezultate.

Drugi i treći problem koji su riješeni u sklopu doktorskog istraživanja vezana su za uslugu *carsharinga*, koja se ubraja dijeljene prometne sustava (*engl.* shared mobility). Usluge carsharinga sastoje se od većeg broja automobila kojima upravlja jedna organizacija, a koji članovima omogućava kratkoročni najam. Najčešće se nudi u većim gradovima i naplaćuje po minuti korištenja te svojim korisnicima nudi prednosti automobila bez troškova i odgovornosti koji proizlaze iz kupovine vlastitog automobila. Tipične primjene carsharinga su primjerice vožnje radi obavljanja kupovine ili povremeni prijevoz radi zabave (kino, restoran).

Drugi problem, *problem rezervacija u jednosmjernom (*engl. *one-way) carsharingu* bavi se pružanjem usluge dugotrajnih rezervacija u takvim sustavima. Za razliku od carsharinga s povratnim vožnjama (*engl.* round trip carsharing), koji zahtijeva da korisnik automobil vrati na istu lokaciju s koje je preuzet, jednosmjerni carsharing pruža veću fleksibilnost jer dozvoljava vraćanje vozila na bilo koju lokaciju u servisnoj mreži. Zbog toga je taj oblik carsharinga i puno zahtjevniji za pružatelje usluga. U takvoj vrsti carsharinga pružanje rezervacija je posebno složeno za organizaciju te ih većina komercijalnih pružatelja ne nudi ili ih nudi u vrlo ograničenom obliku, npr. ne više od 30 minuta prije početka vožnje. U sklopu istraživanja u ovom doktorskom radu, razvijena je metaheuristika ponavljajuće lokalne pretrage za optimizaciju rezervacija u inovativnom načinu pružanja takve usluge. Rezultati simulacije pokazuju da predložene metode mogu znatno povećati razdoblje pružanja usluge (s trenutnih 30 minuta na više od 18 sati) bez značajnog gubitka profita.

Treći problem, *problem varijabilnih cijena u jednosmjernom carsharingu* sastoji se od podešavanja cijene carsharing usluge ovisno o mjestu početka putovanja te vremenu početka vožnje. Glavni cilj algoritma za optimizaciju bio je povećanje profitabilnosti sustava, ali implicitno je uz profit povećana i uravnoteženost broja vozila na parkiralištima diljem područja usluge. Problem je riješen metaheuristikom ponavljajuće lokalne pretrage. Rezultati simulacije ukazuju da se korištenjem optimizacijskog algoritma usluga koja je stvarala gubitke veće od 1000€ dnevno npr. uspješno preobrazila u profitabilan sustav s dobiti većom od 2500€ po danu.

Drugi i treći problem: optimizacija rezervacija te optimizacija varijabilnih cijena usluge carsharinga bili su posebno teški s obzirom na njihovu ciljnu funkciju koja zahtijeva intenzivne resurse, te se zbog toga na računalu izvršava dugo. Navedeni problemi ubrajaju se u razred problema s ograničenim budžetom evaluacija (*engl.* problems with limited budget of evaluations), skraćeno OBE. Tipične primjene metaheuristika implicitno pretpostavljaju mogućnost evaluacije velikog broja rješenja, što pri rješavanju OBE problema znatno sužava područje prostora rješenja koje metaheuristika može istražiti. Uobičajeno se u literaturi takvi problemi rješavaju izgradnjom nadomjesnog modela (*engl.* surrogate model) koji aproksimira zahtjevne funkcije cilja te se može evaluirati veliki broj puta. Rjeđe su primjene koje metaheuristike koriste direktno na sporoj funkciji cilja, bez korištenja nadomjesnih modela.

Rješenja drugog i trećeg problema su prema autoru dostupnim informacijama, prve primjene metaheuristike ponavljajuće lokalne pretrage na probleme s ograničenim brojem evaluacija. Nadalje, izgrađeni algoritmi uspješno su izgrađeni bez korištenja nadomjesnog modeliranja, što nije uobičajena praksa u literaturi te su korištene jednostavne metaheuristike. Rad završava nizom smjernica za rješavanje problema optimizacije s ograničenim brojem evaluacija bez nadomjesnog modela, temeljenih na iskustvima rješavanja ova dva problema. Prilikom istraživanja, kao najučinkovitije tehnike za unaprjeđenje učinkovitosti metaheuristika bili su usmjerenost na intenzifikaciju rješenja te nastojanje za ostvarivanjem što je moguće boljeg početnog rješenja za metaheuristiku.

Izvorni znanstveni doprinosi ovog rada su:

1. Metodologija razvoja za implementaciju metaheurističkih tehnika čiji su ciljevi brz razvoj, ali i visoka učinkovitost razvijenih softverskih rješenja,
2. Eksperimentalna evaluacija razvijene metodologije na tri optimizacijska problema:
    - Problem izrade rasporeda rada djelatnika u pozivnom centru,
    - Optimizacija rezervacija u carsharingu
    - Problem varijabilnih cijena u jednosmjernom carsharingu
3. Metaheuristika ponavljajuće lokalne pretrage prilagođena na rješavanje optimizacijskih problema s ograničenim budžetom evaluacija.

Ključne riječi: metaheuristike, razvojne metodologije, ponavljajuća lokalna pretraga, rasporedi rada djelatnika, carsharing, optimizacija prometa

Contents

Chapter 1

Introduction

Computers are continuously getting better at performing tasks of increasing complexity. They have transformed from isolated “big iron” devices into an ecosystem of interconnected devices that fit into a pocket, and their applications have changed in a similarly striking way. From being restricted to the mathematical calculation in research institutions, their applications expanded and nowadays they include tasks that were considered typically “human” such as driving and routing vehicles, image recognition, staff scheduling and numerous others, as well as tasks that were impossible to solve by people due to e.g. their size or complexity, and too much time that would be required.

An area in mathematics and computer science called *mathematical optimisation* is an important field of study that fuels the development of such applications. Solving problems using techniques of mathematical optimisation requires the existence of a mathematical description of the problem to be solved, and during decades of research, various techniques have been developed for a wide variety of problems. Initially, such techniques were used to produce solutions to logistical and economic problems. With more advanced computers and more research being available, it was demonstrated that numerous real-life tasks could be formulated and successfully solved as optimisation problems. Such algorithms can now perform highly sophisticated intellectual tasks—staff scheduling, packing goods, and transportation. Optimisation algorithms are increasingly used to help with complex problems that involve both technical and social decisions such as “how to perform railway infrastructure upgrade during the next few decades”.

This thesis focuses on *metaheuristics*. They are general optimisation techniques commonly applied to problems on which most other methods fail. Despite their great potential and improvements in theoretical understanding, implementing metaheuristics in real-world software projects is still complicated and expensive. It is CPU-intensive and requires highly trained developers. As a part of this work, the author investigated various strategies to help improve the typical workflow of implementing metaheuristics and reduce the complexity and development time.

Another area where applying of metaheuristics is hard are the problems with *limited budget of evaluations* (LBE). With such problems, estimating the quality of each potential solution requires significant time or other resources (number of CPUs, memory, real-world models and others). This hinders the ability of metaheuristics to find high-quality solutions since they rely on testing a large number of potential solutions. The current literature in this area typically recommends building quicker models called *surrogate models* or *surrogate functions* that gradually learn the features of the problem being solved. While the literature reports success with such methods, not much analysis is devoted to the fact that such extra layers add even more complexity to the implementation process. Further, little attention has been dedicated to the attempts to use metaheuristics without additional surrogate models and development of techniques to improve the performance of the algorithms on such specific problems.

These ideas were tested on three difficult problems:

- Call centre staff scheduling problem,
- Improving the carsharing reservation service by optimising the service quality parameters across the service area,
- Carsharing service profit optimisations with variable trip pricing in the zones of the service area and during the time of day.

The first problem (call centre scheduling) is a common problem, where very quick evaluation is possible, and checking tens of thousands of different schedules is possible in a reasonable time, before returning the best found as the solution. The second two are transportation problems, and they have much more demanding evaluation. In the second problem—carsharing reservations optimisation, a simulator is used to check the quality of each generated solution. In the third problem, the evaluation is even more complex and slow since a mathematical model is used to estimate the solution quality, and it can take up to around one minute to get an evaluation of a single solution.

Implementing complete algorithms for solving these three problems provided an environment where different approaches to the development of metaheuristics can be tested and further improved, to reduce the complexity mentioned above, that a typical metaheuristic based project inevitably bears. In addition to this, the second two problems are suitable test-cases for testing various techniques to solve problems with a slow evaluation function.

Although in this thesis, they are used to develop and test more general algorithm development concepts, all three problems are relevant on their own, both from scientific as well as practical perspective. Scheduling can be a difficult task that nearly all human organisations occasionally face. Scheduling problems commonly involve people, various assignments, meetings and other activities, and problem definition can include numerous preferences and requirements.

Carsharing is an increasingly popular transport mode, in which a fleet of cars is distributed around the service area (typically a city). There exist several different configurations of typical carsharing services that appeared during the last decades, some of which only recently. Most notably, one-way carsharing first appeared in 2008 and still poses a number of difficult challenges to any commercial provider.

As a venture into the interdisciplinary work, transportation problems described in this thesis were not only a very convenient playing field for algorithms research—they are also important and previously unsolved transportation problems. The carsharing applications provided several contributions in the area of carsharing. The reservation improvement algorithm is a part of an innovative method of providing long-term vehicle reservations, which are to this date not provided by any carsharing provider, nor were considered in transportation literature. Likewise, the variable pricing algorithm is the first known application of the variable pricing technique to incentivise user behaviour in carsharing systems.

All algorithms proposed in this thesis have the potential for practical applications. High-quality scheduling algorithms and software can bring substantial improvements in the functioning of complex organisations. It can save the time needed to produce and edit schedules by hand, and reduce the number of staff that must work on schedules. Such savings in the required time can allow cost savings and higher productivity.

Similarly, the carsharing reservation scheme investigated in this thesis has the potential to expand the carsharing market, attract new customers and increase user satisfaction and loyalty. Regarding the third problem—carsharing variable trip pricing, the algorithm is developed to maximise profit by adapting the prices in response to the demand across the service area. The results show that the algorithm was able to turn a simulated Lisbon carsharing provider struggling with losses of more than 1,800 €/day into a profitable company with the daily profit of more than 2000€.

1.1 Contributions

This thesis provides the following original scientific contributions:

1. Design methodology for applications of metaheuristics, targeting fast development and efficient software solutions.
   This contribution is elaborated in detail in Chapter 4.
2. Experimental evaluation of the developed methodology on three problems:
   - Call centre workforce scheduling
   - Carsharing reservation service optimisation
   - Variable trip pricing in carsharing

   This contribution is detailed in Chapter 5. The proposed solution of the call centre workforce scheduling problem can be found in 5.1. The novel carsharing reservation method and the proposed algorithm to further optimise such services is detailed in Section 5.3. The variable trip pricing problem in carsharing is discussed in Section 5.4.

3. Iterated local search metaheuristic adapted to solve optimisation problems with a limited budget of evaluations.

   The specific implementations of the algorithm is discussed in Sections 5.3 and 5.4. Further, in Section 6, a short general overview of the performed adaptations is presented.

1.2 Outline of the thesis

The thesis consists of 7 Chapters, including this Introduction.

Chapter 2 provides a general introduction to the area of mathematical optimisation, combining the mathematical and computer science perspective. It defines the most general terms that are thoroughly used in this thesis—the optimisation *problem*, *solution*, *local optima*, *global optima*, *variables*, and others. It further describes the most important classes of optimisation problems, lists the most important achievements of theoretical computer science that are relevant for the area and brings the general guidelines for solving optimisation problems. This way, the Chapter positions this work in the wide area that optimisation is today. The Chapter concludes with a few examples of noteworthy optimisation problems.

Chapter 3 introduces *metaheuristics*—it illustrates the contemporary view on them and how they can be defined. It further states the most important characteristics shared by all metaheuristics. Several techniques that satisfy nearly all of those characteristics are identified, most importantly *pure random search*, *greedy algorithm* and *local search.* While they cannot be considered metaheuristics, these algorithms are commonly used in algorithms that do fit the definition and have all the required characteristics. This analysis is used as a basis for the component-based view of metaheuristics explained later in the thesis. The chapter finishes with an overview of the established metaheuristic methods

Chapter 4 describes the *bottom-up* development methodology, proposed to increase the agility in implementing metaheuristic methods. The chapter first describes the current practices in developing metaheuristic algorithms and illustrates the high complexity in this process. Some theoretical and practical results relevant to the development are presented. The Chapter continues with the breakdown of all metaheuristics described in Chapter 3 into a set of standard components. The chapter then illustrates the bottom-up development methodology based on these algorithmic components, with the gradual addition of complexity, frequent performance testing and defining the implementation as done as soon as the satisfactory performance is achieved. The author argues that this methodology can bring considerable savings in the cost and required time to produce a well-performing algorithm.

Chapter 5 describes how the methodology proposed in Section 4 is applied to three difficult problems. First, the implementation of the call centre scheduling is described, along with the results. Given that the other two problems studied in this work are related to transportation, a brief introduction to carsharing is provided. The Chapter then provides definitions and the implemented iterated local search algorithm for solving two challenging problems in carsharing: carsharing reservations problem and variable trip pricing problem. Further, to the best of the author's knowledge, these are the first implementations of the iterated local search metaheuristics on problems with a limited budget of evaluations.

Chapter 6 summarises the guidelines arising from two successful implementations of the iterated local search metaheuristic into a set of principles for solving optimisation problems with a limited budget of evaluations. Similarly to the simplicity as a design goal for proper implementations of metaheuristics, this chapter is an extension of Chapter 4 which describes the application of the bottom-up development methodology on problems with a limited budget of evaluations. The Chapter concludes by stating that surrogate models for solution evaluation, which are a common practice in this area, might not be necessary.

Chapter 7 concludes the thesis, with a summary of key findings, guidelines and contributions as well as some ideas for the future work.

Chapter 2

Optimisation problems

In the contemporary world, the area of optimisation is an intersection of several scientific disciplines. Such studies are highly relevant since numerous problems of great theoretical and practical importance can be formulated as optimisation problems. During centuries, several classes of such problems were studied separately, initially in the areas of mathematics and physics. Those studies identified systematics of optimisation problems, several ways to hierarchically organise problem types, and specialised solving techniques. With the computer revolution in recent decades, hardware of high computational capability, as well as increasingly clever algorithms, became ubiquitous. This allowed people to use computers to solve problems unprecedented in their complexity and size. Further, theoretical computer science gained valuable insights into problem complexity as a research area and provided a rigorous analytical framework for ranking problem difficulty.

This Section positions this research of metaheuristic techniques in the broad area of optimisation. It provides a short historical overview of the most important results, followed by definitions used in modern research. It further presents the systematics of optimisation problems and some of the results from the theoretical computer science that shape the directions of research and development of optimisation algorithms. Finally, short descriptions of several important difficult problems are provided.

2.1 Historical overview

In the most common everyday sense, the word *optimum* means the greatest degree of something, attained or attainable under specified or implied conditions, or the amount of something that is the most favourable to some end. The meaning of verb *to optimise* is to make as perfect, functional or effective as possible. Related to these definitions, *optimisation* is the act, process, or methodology of making something as fully perfect, functional, or effective as possible. In a general sense, optimisation is a process of selecting among a set of possibilities to find the best one [1].

Optimisation is ubiquitous. Ingenious examples of optimisation can be found across nature. Light follows the path that minimises the travel time. Physical systems and chemical reactions have a tendency towards states of minimal energy. Spider webs seem to have highly optimised properties, both in terms of spider silk as a high-performance fibre, as well as their structural mechanics [2, 3, 4]. Dandelion seed pappus[1] appears to have perfectly tuned porosity that enhances the flight capacity and minimises material requirements [5]. Performing good decisions was always very important for humans as well. Which animal in a herd to hunt, what to do when one sees a dangerous animal such as a tiger, where to build a settlement—people were able to successfully solve complex problems guided by instinct and accumulated knowledge passed on through generations. With the emergence of science, optimisation became an increasingly formalised process, explored in rigorous ways, as well as an area where numerous innovative algorithms were crafted by engineers.

2.1.1 Antiquity

The first recorded evidence of optimisation problems defined mathematically that is known to the author is the description of the *isoperimetric problem* in ancient Greece [6, 7, 8]. The problem consists of finding the figure in the plane with a given perimeter that has the maximum surface, and analogously in three dimensions, finding the solid with a given surface that has the largest volume. According to the commentator Simplicius from the 6th century, it was known even before Aristotle (4th century BC), that the solutions to the problem in the plane are a circle, and that in space, the solution is a sphere [6, 9, 10].

In the year 19 BC, Latin poet Virgil, in the epic poem *Aeneid* tells the legend about the foundation of the ancient Carthage in 814 BC [11]. According to the legend, the Phoenician princess Dido (also called Queen Elisa) fled her tyrant brother with a group of faithful companions. They decided to settle on the north region of Africa (modern-day Gulf of Tunisia). She was able to persuade the native inhabitants to allow them to take only "as much land as they can enclose in a bull's hide". The resourceful Dido cut the bull's hide in thin strips, this way producing a long rope, and used it to enclose a considerable area of land, clearly attempting to maximise the surface that can be enclosed using the limited material given to her [7].

Another excellent example of optimisation in antiquity comes from China and dates from the 4th century BC. It is preserved in the form of the story about the horse race of Tián Jì [12, 13]. Tián Jì was a general in Chinese county Qí who loved horse racing. One day, the King Wei of Qí, ordered Tián Jì to have a horse racing match with him. The match consisted of three rounds, and in each round, each side could choose a horse to compete with the other

[1] In botany, a *pappus* is a body of feathery bristles around the seed that help disperse the plant seed using wind.

side, with the winner being the side that wins more rounds. Tián Jì knew that his best horse was not a match for the King's best one, that his second horse was slower than King's second, and likewise, his third horse was slower than king's third. Advised by the military strategist Sun Bin, he used the following strategy: he selected his worst horse to race with king's best. After the King won this race, Tián Jì selected his best horse to race against King's second-best and his second-best to race with King's third. In this ancient story, Sun Bin solved what is today known as a *weighted bipartite graph matching problem*, or *assignment problem*, and found the optimal strategy to participate in the tournament. The results confirmed that. Despite losing his first race, Tián Jì won the last two, and by the result of 2:1, he won the entire tournament [12, 13].

The previously mentioned isoperimetric problem was relevant for questions related to land distribution and was mentioned and studied by several scholars in Ancient Greece. In the 4th century BC Aristotle states "*Now, of lines which return upon themselves, the line which bounds the circle is the shortest*" [14]. Approximately in 2nd century BC, Zenodorus proves that the circle has a larger area than any polygon with an equal perimeter. In a rigorous modern sense, after more than 2 millennia of effort, the problem was solved in the 19th century AD, by Swiss mathematician Jakob Steiner who proved that circle indeed is the solution to the problem, using a geometric method [10].

In his celebrated *Elements*, Euclid provides the means to construct a parallelogram with the largest area in a specific setup of parallelograms, along with a geometric proof that the constructed parallelogram indeed has the maximum area (Book 6, Proposition 27) [15, 16]. By applying this proposition, it is possible to provide a solution to a related problem—finding the parallelogram with the largest area that can be inscribed in a triangle and shares an angle with the given triangle. The solution to this problem is a parallelogram formed by the point of the shared angle and the midpoints of the triangle's sides.

Ancient Greek science provided very early observations of optimisation in nature. In the 1st century AD, Hero of Alexandria (1st century AD) in his *Catoptrics* describes that reflected light follows the *shortest path* [17, 18, 19]. In the 4th century AD, Pappus observes that bees build their honey-combs using hexagonal shapes without any space in between cells. He proves that this property allows them to maximise the honey volume in each cell while minimising waste, and minimising the quantity of material needed to build cell walls [19, 20, 21].

2.1.2 17th – 19th century

The 17th century is noted as the time when revolutionary ideas related to calculus of variations were born. This is the first time universal techniques for optimisation were developed—unlike the specialised mathematical treatment of each individual problem that was the case before. The great scientists such as Fermat, Newton and Euler solve complex problems and create new areas of mathematics and physics [10, 22]. Further, optimisation methods from mathematics start being used in other areas of science, such as economics.

In the year 1636, Pierre de Fermat publishes the pioneering work in the area of mathematical analysis, *Methodus ad disquirendam maximam et minimam et de tangentibus linearum curvarum*, in which he shows that derivatives at the extreme points of a function are equal to zero [22]. During the mid 17th century, James Gregory, Isaac Barrow, Isaac Newton and Gottfried Wilhelm von Leibnitz develop the *mathematical analysis* [10, 23].

Several important problems were studied at the time that inspired the development of specialised techniques for solving that were gradually refined and generalised. In the book "*Mathematical Principles of Natural Philosophy*, published 1687, Newton studied a problem of finding the best possible shape of an object moving through "a rare medium" to minimise the resistance from the medium [24, 25]. The *brachistochrone problem*, proposed by Bernoulli in 1696, is to find the curve that minimises the time of descent for a body moving under gravity between two points of a different altitude [26]. The solution is a cycloid curve [27, 28, 29]. These types of problems were an entirely new type of problems that have *curves* or *functions* as solutions [30], and together with the techniques of mathematical analysis, they led to the development of *calculus of variations*. These methods were further developed in the 18th and 19th century by Euler, Lagrange, Weierstrass and others [10]. The calculus is widely used for solving numerous tasks in modern science and engineering, and it is included in almost all higher education textbooks in mathematics. Notwithstanding the importance of this discovery with mathematicians of that time, the area of optimisation was relatively scattered across mathematics. The optimisation was not a formal discipline, and some important contributions were left unpublished, such as the first definition of the modern Steiner tree problem [2] [13, 31, 32].

The emergence of *graph theory* in mathematics, inspired by the work of Leonhard Euler in 1736, provided a very natural framework to specify various combinatorial optimisation problems [33, 34]. Graph theory allows a very intuitive insight into computational difficulty, since numerous graph problems are trivial to define, but incredibly difficult to solve. For example, the classical *travelling salesman problem* was first proposed as a graph problem called *minimum Hamiltonian cycle* problem [33, 35]. Determining the chromatic number of a graph [33, 36], Steiner tree problem [32], minimum spanning tree [37], shortest path [35] and maximum network flow [38] are all classic problems of varying difficulty [33].

In 1784, Gaspard Monge publishes the first formalization of what is today known as *Monge-Kantorovich transportation problem* [39, 40]. The problem was stated as "Given a pile of sand and a set of holes with an equal volume, find the way to realise the transportation of sand into holes with minimum cost. Leonid Kantorovich later improved the formulation [41].

[2] The *Steiner tree problem* is to find a network that fully connects a number of points and minimises the total distance to all points.

In 1823, Jean-Baptiste Joseph Fourier proposed the first definition of a *linear programming problem*, in the form of a system of linear inequalities [42, 43, 44, 45]. He also develops an algorithm for solving it, known today as *Fourier-Motzkin elimination*, after Fourier and Theodore Motzkin, who discovered it independently more than a century later (along with other independent authors) [44, 45, 46]. Cauchy presents the *steepest descent* method [47, 48]. Nowadays, it is still an essential element of numerous nonlinear optimisation algorithms. The development of this method was motivated by solving complex systems of equations.

The 18th and 19th century brings ideas from mathematics and optimisation into mainstream economics. Authors such as Gabriel Cramer, Daniel Bernoulli, and Anne Robert Jacques Turgot introduce the ideas of *marginal utility* and utility maximisation as a goal [49, 50]. David Ricardo, Thomas Robert Malthus, and other authors simultaneously introduce the *law of diminishing returns* [50]. In his nearly forgotten, then gradually rediscovered work from 1854, Hermann Heinrich Gossen states: “*Man should organize his life so that his total life pleasure becomes a maximum”* [51]. It appears that he considered optimisation as one of the most basic principles of human existence. In his scientific work, he assumes the existence of *utility functions*, hypothesises that man’s needs are hierarchically (lexicographically) ordered and discusses individual utility maximisation in a remarkably modern way, providing techniques for utility maximisation under limited time and income constraints [50, 51, 52]. The ideas of utility are further expanded and analysed by Antoine Augustin Cournot, who applies calculus to develop strategies to maximise profit in competition [50, 53, 54]. Léon Walras describes the theory of *general economic equilibrium* [55]. In his theory, under the assumption of absolutely free competition, the maximum utility of each market participant is compatible with the maximum utility of others, and the cumulative social benefit is maximised as well. He further argues that the state of equilibrium is achieved through a process called *tâtonnement* in French, which is usually translated as “trial and error”, and is a type of what in modern mathematics would be described as a hill-climbing heuristic [55].

Expanding on the development of optimisation techniques, such as calculus, several philosophers and scientists became convinced that nature itself, in a certain sense, optimises. Pierre de Fermat shows that the light traverses space in the way that takes the least time, in what is today called the least-time principle, or Fermat’s principle [56, 57]. Gottfried Wilhelm Leibniz argues that the Universe is the *best possible*, otherwise, it would not be distinguishable from God [58]. Leibniz, Leonhard Euler and Pierre Louis Maupertuis propose a generalisation of Fermat’s principle, called *the principle of least action* [59, 60, 61]. The principle, postulated by Maupertuis as “*Nature is thrifty in all its actions*” [59] can be used to deduce equations that govern motion in various physical systems, such as classical physics, relativity and quantum mechanics.

2.1.3 20th century

In the 20th century, the area of optimisation was advanced through several overlapping research directions and became the modern interdisciplinary area it is today. New research directions begin to appear, especially with developments of computer science. The research interest keeps increasing, as evidenced by the literature volume and number of conferences. To the best of author's knowledge, the first textbook that focuses on optimisation, called *Theory of Maxima and Minima* was published by H. Hancock in 1917 [13, 62]. The first conference in the area of mathematical programming, *International Symposium on Mathematical Programming* was started in Chicago in 1949 [13, 45].

Transportation problems

Difficult transportation problems become defined in their modern form. In 1930, Karl Menger defined the messenger problem, now called the travelling salesman problem (abbreviated as TSP), and noted an obvious but slow brute force method [63, 64, 65]. Additionally, he notices that the greedy approach of always selecting the nearest neighbour does not produce optimal solutions [3]. A generalisation of TSP, the *vehicle routing problem* (abbreviated as VRP) was proposed by George Dantzig and John Ramser in 1959 [66, 67]. For some transportation and coverage problems, very efficient algorithms have been found. Examples include the shortest path problem (Dijkstra's algorithm 1956 [65, 68, 69]), and the minimum spanning tree problem (Borůvka's algorithm 1926, Kruskal's algorithm 1956) [69, 70, 71, 72]. For others, including TSP and VRP, efficient optimal solving seemed elusive, despite great research effort [63, 66, 69].

Simplex method, optimisation becomes ubiquitous

Development of the *simplex method* in the late 1940s starts a new era of optimisation, when it becomes widely used and studied [13, 45]. The simplex method is the result of several years of work of George Bernard Dantzig, who at the time worked for the US Air Force. He was assigned to apply his mathematical skills to find a way to "mechanise" the planning of training and logistical supply, and speed up such processes. He modelled what the military called "ground rules" using an objective function, and by 1947, he was able to include all required technological relations into his models, and experimentally validate that the method was fast enough for practical applications [13, 73, 74]. The method was first published in 1951 [75].

[3] "*We denote by* messenger problem *(since in practice this question should be solved by each postman, anyway also by many travellers) the task to find, for finitely many points whose pairwise distances are known, the shortest route connecting the points. Of course, this problem is solvable by finitely many trials. Rules which would push the number of trials below the number of permutations of the given points, are not known. The rule that one first should go from the starting point to the closest point, then to the point closest to this, etc., in general does not yield the shortest route.*"

Examples of optimisation techniques clearly exist pre-1940, however, aside from the far-reaching invention of calculus, they were scattered across several disciplines, and important results were frequently ignored [13, 45]. This is even more emphasized in the history of linear programming. Dantzig mentions "*What seems to characterize the pre-1947 era was lack of any interests in trying to optimize*" [13, 76]. There were some exceptions—special cases of the linear programming problem and early methods for solving were published independently by Fourier in 1823 [42, 43, 44], Charles Jean de la Vallée Poussin in 1911 [77], Theodore Motzkin in 1936 [46], Leonid Vial'evich Kantorovich 1939 [78], and Frank Lauren Hitchcock in 1941 [79]. However, most of this work did not consider practical applications, was forgotten soon after publishing, and his predecessors' achievements were unknown to Dantzig while he was working on the simplex method [45, 75, 76, 80, 81].

The pioneering work in the area published by Kantorowich in 1939 was exceptionally extensive [45, 78, 81]. Despite the important innovations, the research was met with hostility from the USSR authorities. They viewed his ideas about mathematical optimisation in economics non-Marxist. In 1943, the Soviet political climate was so bad for Kantorovich that he regretfully decided to postpone his research. In his own words, "*It was dangerous to continue*", and his work was left unknown [82, 83, 84, 85]. Kantorovich's research was widely circulated only in 1959, after considerable progress in the linear programming was made [84, 85, 86].

The simplex method allowed economists to analyse and optimise models of unprecedented complexity in an efficient, systematic way. In its earliest days, it was used without electronic computers, on analogue devices or hardware based on punch-cards. The rapid development of electronic computer technology in the post-war years further advanced development of optimisation as a formal discipline. Since the 1950s, generations of economists, mathematicians and engineers were trained to use the simplex method. Even today, it is considered one of the most widely used optimisation techniques, due to abundant literature, simplicity of modelling, and advanced software for solving [45, 80, 82, 87].

The simplex method is very fast with the great majority of inputs. However, there exist classes of problems on which it is not efficient. In 1979, Khachiyan presented the ellipsoid method, and in 1984 Karmarkar presents his projective algorithm. While these algorithms work faster when applied to problematic classes for the simplex method, overall, the simplex method still outperforms them both, especially when solving large problems [87].

Nonlinear and combinatorial optimisation

In the area of nonlinear programming, the definition of *Karush–Kuhn–Tucker conditions*, abbreviated as *KKT conditions* represent an important breakthrough, first published by Karush in 1939 in his Master's thesis [88] and later, independently, by Kuhn and Tucker in 1951 [89]. These conditions are necessary conditions for optimality in nonlinear programming, provided that the problem being solved meets the regularity conditions defined by the KKT theorem. It allowed the development of a new type of optimisation algorithms, that are based on numerical solving of the KKT system [87].

The fifties are the time when important advances in combinatorial optimisation were achieved. They had numerous applications in the aforementioned area of transportation optimisation. In 1956, Fulkerson and Ford study network flow [90], and Kruskal developed his minimum spanning tree algorithm [72]. In 1958 Gomory initiated the study of *integer programming* [91], and in 1959, Dijkstra proposes his shortest path algorithm [68].

Wider usage of heuristics, the appearance of metaheuristics

The fifties, sixties and seventies further diversify the area of optimisation, as new subtypes keep appearing: stochastic programming, global optimisation etc [92, 93, 94]. The increases of computational power expand the areas where algorithms for optimisation can be applied. More memory and faster processors allow larger and more complex models [13]. Artificial intelligence appeared as an exciting research direction [95].

During this time, a new class of optimisation techniques appears. Heuristics and their more general version called *metaheuristics*, are introduced for difficult problems [96, 97]. They are usually applied to problems where more analytical and exact approaches are too slow or too difficult to implement, and an especially notable application area is large scale combinatorial optimisation problems [96]. Metaheuristic techniques are the focus of this thesis.

Early ideas in this direction appeared even before the sixties in a rudimentary way. Ideas such as *regret selection*, *greedy selection*, and *local search* are general guidelines that can be applied for development of efficient heuristic algorithms [69, 98]. The algorithmic principle of maximum regret avoidance was presented in 1951 by Leonard Jimmie Savage [99]. Greedy selection is the idea that drives Kruskal's spanning tree algorithm, and Dijkstra's shortest path algorithm, both published in 1956 [68, 72]. Nevertheless, one could argue that they are such common-sense general ideas that they were present even before.

The idea of local search, also called *hill climbing* is also a very general and widely used notion, and the moment in time when it was first published is difficult to pinpoint [69, 95, 98, 100]. The already mentioned steepest descent method developed in the 19th Century by Cauchy (for continuous optimisation) [48] has such remarkable similarity with the local search (usually mentioned for combinatorial optimisation) that it can be viewed as a continuous version of local search. To the best of author's knowledge, the first published local search algorithm in combinatorial optimisation was the *2-opt* heuristic for solving the travelling salesman problem by Flood (1956) [101] and Croes (1958) [102]. A similar edge exchange nowadays referred to as 3-opt was proposed by Bock in 1958 [103]. While local search, greedy selection and regret selection can be regarded as *proto-metaheuristics*, they are still simple guidelines with a notable drawback—they do not provide the ability to avoid local optima [69, 96, 104, 105].

The first metaheuristic in the modern sense was inspired by nature and ideas in artificial intelligence: instead of using custom algorithms built by programmers for each task, we could try adding some general problem-solving strategies to computers. Equipped with such strategies, the computers would then solve problems on their own. Attempts to do that were either simulating the ways people solve problems or were simulating *evolution* [95, 96]. Ideas of simulating evolution for solving complex problems can be dated back to Alan Turing, who suggested in 1950 that evolution could be used to build a "learning machine" [106]. During the fifties, Nils Aall Barricelli and others developed *evolutionary algorithms* as a research technique in biology to study natural evolution [107, 108]. Soon, it was evident that simulated evolution could be used to perform optimisation as well, and authors like Box, Friedmann, and others were the first to apply simulated evolution for solving optimisation and machine learning problems [96, 109, 110]. A notable early optimisation technique was the initial development of *evolution strategy*, by Ingo Rechenberg in the 1960s and 1970s [111, 112, 113]. This new optimisation technique was based on successive *mutation* of a single solution and choosing the changed version if it improved the solution quality and was applied to the difficult problem of aerodynamic wing design [114]. An approach in which artificial intelligence can be evolved in computers, nowadays called *genetic programming* was first proposed by Fogel in 1966 [96, 115].

The *genetic algorithm*, proposed by John Holland, was the first metaheuristic in the modern sense [96]. It was published in 1975, and along with the mutation from evolution strategies it featured new ideas of using population and crossover operators in simulated evolution [96, 116]. These three parts formed the generic problem-solving method that had the ability to focus the search around good solutions while avoiding being stuck in local optima. Another important achievement by Holland was the foundation of theoretical investigation in metaheuristics—his *schemata theory* states that genetic algorithm iteratively increases the frequency of good components that improve solution quality [116, 117]. Early results indicated that evolutionary computation might be a possible solution to the quest of artificial intelligence—a problem-solving method that does not require detailed programming based on features of the problem to be solved. Nevertheless, several of these claims were later found to be too optimistic and based on insufficient understanding of crucial issues such as scalability [95, 96, 118]. Nevertheless, the genetic algorithm quickly became popular, and completely new area of optimisation appeared. In the following decades, great enthusiasm was present with numerous published papers, and new conferences and journals dedicated to the genetic algorithm and evolutionary methods for optimisation [96, 117].

During the 1980s, several other metaheuristics were developed [96]. By also drawing inspiration from nature, the *simulated annealing* was proposed in 1983 [119]. It is an algorithm that mimics the annealing process used in metallurgy to improve the characteristics of the material.

The process consists of first heating, then slowly cooling the metal, and doing so reduces the free energy in the material. By analogy, the objective function to be minimised can be viewed as the free energy, and simulating the process of annealing can be used as a minimisation technique [96, 119, 120, 121, 122].

Two more metaheuristics inspired by nature were developed in the nineties. The *ant colony optimisation*, published by Marco Dorigo in 1991 is inspired by the behaviour of ants and the way they find good paths while searching for food. It uses an array of agents that coordinate building of a solution from individual elements, and during this process gather and communicate information about decisions that increased the quality of solutions. This technique also brings an innovative way to use known information about good solution parts, if it is available [96, 123, 124]. The *particle swarm optimisation* metaheuristic is based on social behaviour of animals. It was inspired by the way fish swim in large flocks. In this method, an array of agents called particles is initialised in random locations of the vector hyperspace that represents all possible solutions, and different points have different quality. These particles fly through the solution hyperspace in order to approach the optimum. The algorithm uses known locations of good solutions to strategically adjust the speed vector of each particle [125, 126].

While nature has been a great inspiration for several metaheuristics, there are also those who are not inspired by natural processes, but instead use ideas that are based in ways people might use to solve problems [96]. A simple, but effective algorithm using this principle is *iterated local search*. In order to avoid local optima, the idea is to iterate runs of local search, however, each time from a slightly different point, close to a previous local optimum [96, 127]. First developments of this method can be traced back to 1981, when Baxter uses it for solving the depot location problem [128]. It was later rediscovered independently by several authors under different names, until its current name became common [96, 127]. *Tabu search* method, proposed by Glover in 1986 uses a simple idea of using memory to keep track of the previously visited solutions. The algorithm prohibits returning to recently visited solutions, this way reducing the probability that the algorithm will stop at a local optimum without exploring other regions [129, 130, 131, 132]. Another way of doing this was discovered in 1989—the technique called *GRASP* [133], which is an abbreviation for *greedy randomized adaptive search procedure*. The key idea is iteratively restarting the local search, each time from a different point generated by randomized greedy algorithm [96, 134, 135]. In 1997 Mladenović and Hansen suggested that using more than one neighbourhood definition is beneficial in the methods and developed their *variable neighbourhood search* [136, 137].

Theoretical computer science

The theoretical computer science started developing in the 20th century, with Gödel, Church and Turing providing first theoretical insights on the limits of computation [138, 139, 140, 141].

Computation complexity theory developed the techniques to evaluate the *time complexity* of algorithms, and communicate it using the asymptotic time complexity notation [142]. The theory of $\mathcal{NP}-completeness$ is a fundamental achievement with great impact on optimisation. It identified large classes of problems for which no efficient algorithm is known despite decades of effort [143]. Finally, the *no free lunch theorem*, proved by Wolpert and MacReady in 1996 gives a strong argument for *specialisation* to improve algorithm performance [144, 145].

Nobel prizes

Two Nobel prizes were awarded for achievements in the area of optimisation. In 1975, Leonid Kantorovich and Tjalling Charles Koopmans received the Nobel Prize in Economics, for their contributions to the theory of the optimum allocation of resources [13, 76, 146]. In 1990, Harry Markowitz received the Nobel Prize in Economics for his pioneering work in the theory of financial economics, developing a theory for households' and firms' allocation of financial assets under uncertainty, the so-called theory of portfolio choice. It was based on quadratic programming, an important area of study of nonlinear optimisation [13, 147].

2.2 Optimisation in theory and practice

Generally, the process of applying optimisation techniques to a problem starts with four key steps:

1. Identifying the optimisation objectives,
2. Identifying the variables or characteristics that can be decided on,
3. Identifying any restrictions in the values of the variables,
4. Identifying the dependencies between variables and the objectives.

This process is called *modelling*, and the result of this process is called *model* or *mathematical model*. It usually involves a multidisciplinary approach where domain experts for a specific problem communicate with the optimisation consultants that guide them through the process. Each of these steps must result in a formal definition, and the produced model is a description of the problem properties in the language of mathematics [87].

The first step involves choosing the most important goals of the optimisation project. Those are typically minimisation of undesired properties – such as risks, costs, time, energy, travelled distances. Conversely, goals can include maximisation of desired outcomes, and some examples include robustness, profit, efficiency, user and employee satisfaction.

The second step includes specifying the variables that can be adjusted during optimisation, and that have an impact on optimisation goals. Such components are called *variables* or *decision variables*. Each variable needs to be set to a value, and setting each variable represents a choice that needs to be performed in order to solve the problem. In an application for vehicle routing and delivery optimisation, a decision variable might be a sequence of "where to go next" decisions, as they have a great impact on transport efficiency, total distance driven and total cost. During this step, it is also important to specify which components of the system cannot be decision variables—because changing them is not possible or is prohibitively difficult.

The third step consists of identifying restrictions in the values of the variables, called *constraints*. Those are typically physical, legal, quality or safety limitations of what is allowed in the system we are optimising. In a physical model for example, no speed can exceed the speed of light. In a vehicle routing application, no driver should drive without a break longer than specified in the labour laws and traffic security regulation [87, 148].

In the final step, to be able to evaluate the quality of each decision, a description of dependencies between the variables and the desired objectives needs to be determined. Some decisions related to values of the variables help achieve the optimisation objectives, others not so much, and a way to evaluate them must be defined as a formula, called *objective function*, or *cost function*. Using this function, one can numerically check how "good" the selection of variable values is. Formulating a good model requires adding enough detail for the model to be realistic. Simple models might be easy to handle, however they might not be able to capture sufficient detail of the real system and might produce results that do not have much value. Conversely, insisting on too much detail might produce a model that is too complex to solve [87].

After the model of the optimisation problem is complete, various optimisation techniques can be used to find solutions. A "universal" optimisation algorithm that performs well on any given problem does not exist. Instead, a variety of optimisation algorithms emerged, with specialised techniques for various different problem types. Choosing an appropriate algorithm for the problem is an important step to ensure efficient solving [87].

2.3 Definitions

In the broadest sense, an *optimisation problem* is defined as finding $\mathbf{x}$, to

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimise}} \quad f(\mathbf{x}), \tag{2.1}$$

$$\text{subject to} \quad g_i(\mathbf{x}) \geq 0, \quad i = 1, \dots, m, \tag{2.2}$$

$$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, p, \tag{2.3}$$

where $f(\mathbf{x})$ is an objective function, while g_i and h_i are functions that define the constraints. The objective function f, and all the constraints g_j, h_j are scalar functions calculated based on the values of decision variable vector $\mathbf{x} \in \mathbb{R}^n$. Each decision variable in $\mathbf{x}$ can also be called a *solution component*. The constraints defined using greater than inequalities (g_i) are called inequality constraints, while those defined using the equality operator are called equality constraints (h_j). Each complete assignment of all the decision variables corresponds to a potential *solution* of the optimisation problem [87].

In the standard form, as specified above, the right-hand side of all constraints is equal to zero—some transformation might be necessary to represent a problem in this form, since for people it is usually more natural to express constraints with constants at the right-hand side. For example, the speed limitation, that restricts all speeds in a physical model not to exceed light speed, written as $v \leq c$, where $c = 299\,792\,458\,\frac{m}{s}$ can be easily transformed to the standard form $c - v \geq 0$. Likewise, a similar transformation might be needed to convert maximisation into minimisation, which is a standard form to define general optimisation problems. This can easily be done by negating the objective function, and minimising $-f$. For example, the problem of profit maximisation with objective function f_{max} is equal to the minimisation problem $\underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimise}} \quad -(f_{max}(\mathbf{x}))$.

If all the constraints g_i and h_j are satisfied for a point $\mathbf{x}$, then it is called*feasible*. Points that do not satisfy all the constraints are called *infeasible*. If there are no constraints, all points are feasible. The definition of feasibility mostly relates to suitability for use in the real world. Since the constraints usually represent strict limits on what is allowed in a solution, violations in the constraints correspond to issues that prevent applications in practice, either due to safety or legal limits or because they lead to completely senseless system states. Some examples of constraint violations might include air traffic optimiser suggesting the plane to go beyond the designated altitude limit, classical thermodynamic system model in which temperatures drop below 0 K, or a vehicle routing application suggesting that a driver should drive 18 hours without a break [87]. Note that in a broader sense, an objective function might not be defined in algebraic terms—it might be an evaluation by groups of people, a result of a simulation or physical testing.

Formally, an *instance of an optimisation problem* is defined as a pair (S, f), where S is the set of feasible points, and an objective function $f : S \rightarrow \mathbb{R}$ that assigns an *objective function value* to each point. The objective function value is frequently abbreviated to *value*, and is sometimes called *cost*. The problem is to find $\bar{s} \in S$, that minimises the objective function f [8, 69].

An *optimisation problem* is a set of instances of an optimisation problem. While formally, any set of problem instances can be defined as a problem, in the optimisation community, well-defined problems always consist of a set of instances sharing the definition of the decision variables, the constraints, and the objective function. It is a collection of similar problem instances, that share the same structure, as opposed to a single instance, that can be viewed as "input data", with all required details to perform optimisation in that specific case [8, 69].

2.3.1 Solutions and optimality

In the optimisation community, the word “solution” can have multiple meanings. In the narrow sense, solution of an optimisation problem is the *optimum*, a point $\bar{\mathbf{x}}$, for which the objective function f reaches its minimum, and that satisfies all the constraints g_i and h_j. In a broader sense, the word solution can mean the output of an optimisation algorithm, that might or might not be optimal, depending on the used algorithm. In an even broader sense, the terms *candidate solution*, *potential solution* and *possible solution*, sometimes abbreviated to “solution”, can mean any assignment of the decision variables, regardless of its quality, or the constraints. In this sense, they are synonymous with the word *point*, as any possible solution of an optimisation problem as defined in Equations 2.1–2.3) forms a point in an n-dimensional hyperspace. Therefore, the set of all potential solutions, or the domain of the objective function f is sometimes called *solution space*, or *search space* [149, 150].

The best possible outcome of any algorithm would be finding the best solution, also called the global optimum. Formally,

a feasible point $\bar{\mathbf{x}}$ is a *global optimum* of the function $f(\mathbf{x})$
if $f(\bar{\mathbf{x}}) \leq f(\mathbf{x})$, for all $\mathbf{x}$,

where $\mathbf{x} \in \mathbb{R}^n$ (or some other domain, according to the problem definition). Global optimum achieves the best value of the objective function $\bar{\mathbf{x}}$, also called *optimal value* [87]. An optimisation problem can have:

- *no* optimal solutions, in cases of constraints that cannot be satisfied at once, or unbounded objective functions,
- *one* (unique) optimal solution,
- *multiple* optimal solutions, in cases when there exist multiple feasible points with equal objective function value.

If an optimal solution exists, there can be only one optimal value [87].

Global optimum is difficult to find and many optimisation algorithms can find only a *local optimum*. Local optimum is tightly related to the definition of the neighbourhood. A *neighbourhood* around the point $\mathbf{x}$, is a subset of the objective domain, that contains $\mathbf{x}$. Which points are elements of the neighbourhood and which not is decided based on some appropriate definition of “closeness”, which is domain and problem-specific. The neighbourhood around $\mathbf{x}$ is denoted $\mathcal{N}(\mathbf{x})$, and contains all elements of the function domain that are close enough to $\mathbf{x}$ [69, 87]. Local optimum is the best solution in this subset of the function domain. Formally,

a point $\mathbf{x}^*$ is a *local optimum with respect to* $\mathcal{N}$
if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{N}$

The idea behind finding local optima is the following: since it is very difficult to find a global optimum, and such algorithms might not be able to produce results in reasonable time, restricting the domain to some smaller region allows the algorithm to find a result quickly, however, this result is guaranteed only to be locally optimal. With difficult problems, unless the process has been incredibly lucky, a local optimum in a neighbourhood does not correspond to the global optimum [69, 87].

2.4 Types of optimisation problems

Studying problems separately allows development of efficient algorithms and gaining theoretical insights into the properties of each problem, instead of simply creating a "quick-and-dirty" algorithm for each instance. Further, identifying key properties of various problems allows defining related problems and transferring successful ideas to similar problems. Contemporary computer science and mathematics gathered rich insights into various problem types and knowing the problem type for the problem one is solving is critical to allow choosing appropriate direction in finding successful solution techniques [87]. Several ways to classify optimisation problems are considered valuable.

2.4.1 Constrained and unconstrained optimization

Depending on the existence of the constraints, optimisation problems can be classified into constrained and unconstrained. Unconstrained problems are defined by only the objective function, with an empty set of constraints g_i and h_i in the standard form (equations 2.2 and 2.3). They appear in numerous applications, especially in natural sciences and mathematics. Physical systems tend to the states of minimum energy, and finding function extremes is frequently needed in mathematics. Unconstrained problems can arise from simplifications of problems with constraints that are safe to disregard under certain conditions. Further, some constrained problems can be approximated as unconstrained problems, using penalty and barrier methods. Such transformations add appropriate cost to the objective function value when the constraint is not satisfied, this way discouraging the optimisation algorithm from breaking the constraints [69, 87].

Conversely, *constrained optimisation* is the area of optimisation that studies problems with explicitly stated constraints (equations 2.2 and 2.3). Constrained optimisation is a broad area where the problems and their complexity greatly vary depending on the number of constraints, restrictions they introduce and the overall constraint complexity. The initial general definitions of g_i and h_j allow any function to be used to specify the constraints. In simplest cases, such constraint can be variable value limits, e.g. $x > 0$ and $x < 50$. In more complex cases, a constraint could be any non-linear relation of the variables. Recognising the type of constraints is very important to choose the appropriate solving technique, and further classification of such problems exists, depending on constraint specifics, as elaborated below [69, 87]. Problems with large number of constraints or with very complex constraints are called *highly constrained problems*. For such problems, even finding a feasible point might be a challenging task [151].

2.4.2 Stochastic and deterministic optimization

Determinism is a possible feature of both the optimisation *problem*, as well as the optimisation *algorithm* used to solve the problem. The noun *determinism* and the adjective *deterministic* means that some value or outcome can be assessed conclusively, and with certainty. *Nondeterministic* means that something is not deterministic, and the related word *stochastic* indicates something that has a random variable or is based on a random process, and cannot be determined with certainty [87, 152].

Optimisation poblems

Deterministic problems are fully specified and in such models, no uncertainty is present. For a given point, the objective function value is always the same. The assumption of deterministic model is that the model captured sufficient level of detail to provide predictions whose error can be neglected. Determinism of the model simplifies applications of the optimisation algorithms, which can assume the stability of the problem and optimise only one scenario of the given problem.

Reality can be difficult to predict and model accurately. Approximations and simplifications are necessary for any model of real systems to allow models to be computationally tractable, and allow analysis and optimisation in reasonable time. Further, for some problems, fully precise information about the system state is unknown. Any physical measurement implies noise in the data, and uncertainty is always present in models that deal with prediction of the future. Forecasting the temperature in a given city tomorrow, demand for taxi vehicles in a street of that city, or the sales results from a store in that street is not possible with full accuracy and some randomness in such predictions cannot be avoided. Nevertheless, modellers are frequently able to estimate the uncertainty of their forecasts, and assign different probabilities to various scenarios. In such cases, techniques for solving *stochastic optimisation problems* can be used to handle the uncertainty and include randomness in the optimisation problem. The area of optimisation studying this class of problems is called *stochastic optimisation*. Using these techniques, users can optimise the desired criteria across multiple scenarios [87].

This thesis focuses on deterministic problems. Adding randomness to the model is in contradiction with assumptions about the stability of the objective function value that is implied with algorithms discussed in this work. Note, however, that numerous stochastic algorithms represent randomness as a set of deterministic subproblems, that indeed can be solved by algorithms for deterministic optimisation [87].

Optimisation algorithms

Algorithms that produce solutions to the input problems can be classified into

- deterministic algorithms,
- stochastic or nondeterministic algorithms.

Deterministic algorithms behave in an easily predictable way and guarantee equal output for each equal input. Conversely, *stochastic algorithms* (also called *nondeterministic algorithms* and *randomized algorithms*) do not guarantee this, perform random choices, and for a single input, might produce different output each time they are used [153, 154].

2.4.3 Continuous and discrete optimisation

The general problem definition in Equations 2.1–2.3 assumes that the domain is a set of vectors of real numbers $\mathbf{x} \in \mathbb{R}^n$. While large array of problems can be represented in this way, for some problems, variables can hold only integer values. This frequently happens when dealing with atomic units that cannot be split into parts. For example, there is no sense in asking a delivery driver to ship 3.7 of four books a customer ordered today, then 0.3 books the next day, as there is no sense in ordering 0.7 ships from a ship factory. Likewise, a “yes or no” decision might be a part of the problem [69, 87, 148]. Formally, the restriction to integrality is modelled by adding the constraint

$$x_k \in \mathbb{Z} \quad k = 1, \ldots, q, \tag{2.4}$$

where x_k are all components of the vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ that must have integer values, and $\mathbb{Z}$ is the set of all integers. If the vector $\mathbf{x}$ consists only from integers, and no component is real-valued, the problem is called an *integer programming problem.* Problems of this type are solved using techniques of *discrete optimisation* [69, 87]. Problems with both integer and real-valued variables are called *mixed integer programming problems.* Conversely, problems with only real variables, that deal with finding the best value out of an uncountably infinite [4] set and smooth [5] objective function f are called *continuous optimisation problems* [87]. Optimization of nons-

[4] A set S_u is *uncountably infinite* if it is infinite and there exists no one-to-one correspondence (*bijective function*) between S_u and the set of natural numbers $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$ [155, 156].

[5] A function is *smooth* if its derivatives up to a certain order are continuous [157, 158]. A function is continuous if small changes in the argument change the function value for a small value. More formally, a function $f(x)$ is *continuous* at point x_0 if it is defined at this point, the limit of f as x approaches x_0 is defined, and $\lim_{x \to x_0} f(x) = x_0$

mooth functions, that might not be differentiable or might have discontinuities is a separate area of research [87].

A *discrete optimisation problem*, or *combinatorial optimisation problem* deals with finding the best item in a finite or countably infinite set [6]. Contrary to the idea that a reduction in the type and the number of elements to search also renders the problem simpler, discrete problems are generally more difficult to solve. Solutions to combinatorial optimisation problems are typically an integer or a vector of integers, graphs, sets, or subsets [69, 87, 162].

Continuous optimisation techniques use information about the objective function to infer function behaviour in the proximity of observed points, which speeds up the solving process. Such general conclusions about the function value close to a point cannot be deduced with discrete problems, where large differences in objective function value are possible for very close points. Due to great differences in the important properties of discrete and continuous optimisation, the two fields diverged during history and developed separate solving techniques [87]. This thesis is focused on combinatorial optimisation problems.

2.4.4 Exact and approximate optimisation

Methods for optimization can be classified into *exact* and *approximate* [8, 69, 162]. With exact methods, the user is always guaranteed to get the optimal solution to the problem. In continuous optimisation, there exist classes of problems where finding the exact solution is fast, however, in general non-linear problems, finding the global optimum can be difficult. Writing fast exact algorithms for combinatorial optimisation is even more difficult. Typical issue with exact algorithms is the fact that it can be difficult to prove that the point $\bar{x}$ is an optimum, and finding the optimum is even more challenging. As a consequence, exact algorithms can be slow and scale poorly with increases in problem size, especially on certain classes of difficult problems. For them, these scalability issues mean that the algorithm is too slow to be considered, except on the smallest problem instances, and even for the moderately sized problems, exact algorithms could take centuries or even millennia to complete. This considerably limits practical applicability of such methods. For numerous problems of great practical importance, there exists no known fast exact algorithm [8, 69, 87, 162].

Approximate methods do not guarantee finding the optimum. Instead, they use different strategies to try to approach as close to the optimum as possible in the provided time. These methods typically provide suboptimal solutions, however they do that in a relatively short time. Approximation algorithms, heuristics and metaheuristics are important classes of approximate methods, mostly used in the area of combinatorial optimisation [8, 69, 162, 163].

[159, 160, 161]

[6] A set S_c is countably infinite if it is infinite and there exists a one-to-one correspondence (*bijective function*) between the set S_c and the set of natural numbers $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$ [155].

2.4.5 Single and multiobjective optimisation

The standard form of the optimisation problem in Equations 2.1–2.3 has only one objective function. Such problems are called *single-objective* optimisation problems. In practice, it is common for a problem to have more than one objective that must be optimised at once. For example, an investor might want to invest money into stocks that will have the highest return and minimum risk, a car manufacturer might want to use motor with maximum power that also has minimum weight. These problems are called *multiobjective optimisation* problems. To solve such problems, complex tradeoffs are frequently necessary to achieve good balance of multiple objectives, especially when there is a constellation of mutually conflicting goals. Further, since in multiobjective optimisation the result of evaluation by the objective functions is not a scalar, but a vector, refined ways to evaluate each point in the feasible region are needed [164].

For such problems, generally, it is rare to find a solution that outperforms all others across all the objective functions. Frequently, the criterion of *Pareto optimality* is used—a point is considered *Pareto optimal* if it is not possible to improve the value of a single objective without degrading some of the others. Using the ideas of Pareto optimality, a solution of the multiobjective optimisation problem is not a single point, but instead a set of points. Without additional preference information, all such points are equivalent in terms of finding the best [164, 165].

Various solving techniques are used for multiobjective problems. A very common technique for solving is *scalarisation*—converting multiobjective problem to a single-objective problem. After a suitable representation as a single-criteria problem has been found, the common optimisation techniques for the single-objective problems can be applied. A widely used simple scalarisation technique is the *weighted-sum method*, where a *weight factor* is assigned to each objective, and the scalarised objective is the minimisation of the sum of weighted individual objective function values [164]. More complex techniques can assume a hierarchy of objectives or intrinsically handle multiple solutions in the algorithm, with the goal of approaching the Pareto optimal set. Regardless of the used solving methods, detailed insights from the users are crucial to identify and adequately model multiple objective priorities [164].

2.4.6 Important classes of objective functions

Some objective functions are easier to optimise than others. If it is possible to prove that the objective function satisfies certain desirable properties, it is possible to apply specialised, efficient algorithms that use specifics of the objective function to work faster [87].

In continuous optimisation problems, a notable class are *convex optimisation problems* and *linear programming*. For such problems, it can be proved that every local optimum is a global optimum as well. This significantly simplifies the search process and alleviates the need for sophisticated local optima avoidance techniques, that are necessary in general optimisation algorithms [87]. Formally, the problem is convex if

- the objective function is convex [7],
- inequality constraints $g_i, i = 1, \dots, m$ are concave [8].
- equality constraints $h_j, j = i, \dots, p$ are linear [87].

When the objective function f, as well as all the constraints g_i, h_j are linear, the optimisation problem is called *linear programming problem.* Very efficient algorithms are developed for this class of problems, most notably the *simplex algorithm* and the *ellipsoid method.* The simplex algorithm of G. B. Dantzig is a classical optimisation algorithm that has undergone decades of development and is considered the fastest algorithm for linear programming. Nevertheless, for certain problems, it is inefficient, as the number of steps to complete becomes exponential to the problem size. The ellipsoid algorithm has better worst-case complexity—it requires the number of steps that is polynomial to the problem size. Unfortunately, this method approaches its most pessimistic number of steps on all problems, and therefore on almost all problem instances, the simplex algorithm is faster than the ellipsoid algorithm [87, 166].

Unfortunately, for general nonlinear and especially for nonsmooth and discrete problems that are the focus of this thesis, such speedups are difficult or near impossible to achieve. These functions have fewer properties that the algorithm can utilize to reduce optimisation complexity. As algorithms get more general and applicable to a broader range of problems, they also tend to be less efficient. Therefore it is highly important to recognise the class of the problem to be solved. If specialised fast algorithms can be applied to the problem being solved, it is always recommended to use them instead of more general methods [69, 87, 145].

2.4.7 Problem size

Problem size in optimisation is usually evaluated using the number of variables. For discrete problems, problem size is sometimes also evaluated using the number of elements in the domain. The problem size itself does not tell us much about the problem difficulty—it is the type of the objective function and the constraints that are the principal cause of complexity. As an illustration, linear programming papers by Barnhart et al. [167] and Bixby et al. [168] published during the 1990s routinely report solving problems with millions of variables in less than an hour. For a different problem, even a hundred is a lot to deal with, e.g. Pecin et al. in [169] report that it takes up to 17 hours to solve some VRP instances with 100 locations, using state-of-the-art algorithms and hardware available in 2017.

[7] A set $S \in \mathbb{R}^n$is called a *convex set* if any straight line segment connecting two points in the set is entirely in S. A function $f$is *convex* if the domain is a convex set and any line segment connecting two points in the graph of f lies above or on the function graph [87].

[8] A function g is concave if $-g$ is convex [87]

Still, for the instances of the same problem, the size can be used as one of the indicators of relative complexity as compared to other instances of the same problem. Generally, the larger the problem is, it is more challenging to solve using computers. With a large number of variables, the number of different variable combinations to evaluate becomes increasingly larger, which adds difficulty in finding the optimal or near-optimal solutions. Further, even for problems for which efficient algorithms are known, and especially for those that we do not know to solve efficiently, very large problems might go beyond what is possible on the available hardware in terms of memory and computation resources. Such big problems are called *large scale problems* and are studied in a separate subfield of *large scale optimisation* [87, 170].

Very large problems are usually solved using strategies such as *decomposition* or *partitioning* into a series of smaller problems that can be effectively solved. Other commonly used techniques are various approximations and heuristics. In combinatorial optimisation, which is the focus of this work, large-scale problems are very common [87, 170].

2.4.8 Objective function evaluation

The process of finding the optimum assumes that the objective function can be evaluated—that for each $\mathbf{x}$, it is possible to find the value of the objective function $f(\mathbf{x})$ at that point. Additionally, it is typically assumed that the objective function evaluation can be done using reasonable resources such as time and computational power. Quicker the evaluation, the algorithm will also in general be quicker to find a good solution. For a great deal of optimisation problems, e.g. when the objective function is a simple algebraic formula with a reasonably small number of variables, an optimiser can evaluate large numbers of potential solutions before providing the result [87, 171]. The travelling salesman problem satisfies this assumption since the objective function is a simple sum of all the distances in a route that can be calculated very quickly. Likewise, the constraint of having all cities visited only once can be verified using several quick and simple set algebra calculations.

Still, there exist important optimisation problems that do not allow evaluating large number of solutions [171, 172, 173, 174]. Examples include transportation problems that are evaluated using simulation [172, 175, 176, 177, 178, 179, 180, 181, 182] and engineering problems that require computationally demanding steps such as solving differential equations [183, 184, 185, 186, 187, 188, 189]. These processes are computationally intense and to perform an evaluation they require either a lot of CPU power, a lot of time or both. In some cases, the real values for the optimisation problem evaluation can be found only by building physical models such as wind tunnel experiments or synthesizing chemical molecules and performing biological tests with them [184, 190]. While the cost of building and testing phisical models is usually too high to be used as a part of the optimisation algorithm, such high accuracy methods can be used to validate the final solution, while the algorithm uses a simplified evaluation function. The third category are studies where solutions need to be evaluated by people [191, 192, 193]. A similar difficulty can arise when the algorithm is a part of a realtime system and has a very short time to produce solutions, even when the objective function is simple and can be calculated quickly, e.g. for problems in robotics [194]. All these categories are especially difficult to solve since algorithms need to access the objective function sparingly. These problems are called *problems with limited budget of evaluations* (LBE problems) or *expensive optimisation problems*.

Problems with limited budget of evaluations are an important part of this thesis. Two out of three problems for which a solving algorithm is presented in Chapter 5 are transportation problems with a limited budget of evaluations. The difficulties in evaluating large number of solutions is especially challenging when using metaheuristics and is an active research area [173, 175].

2.5 Theoretical computer science

Theoretical computer science provided numerous results of great practical, theoretical and philosophical importance. The most important achievements for the area of optimisation are *analysis of algorithms*, *computational complexity theory*, and the *no free lunch theorem*. Algorithm analysis helps practitioners and algorithm developers get an estimate of how long the algorithm will run and how much memory it will take [195, 196]. Computational complexity theory generalises algorithm behaviour and provides far-reaching insights that shape the techniques to solve problems. It helps us decide if it is possible to develop an exact method for a given problem, and compare the difficulty of various problems [197, 198, 199]. Finally, the no free lunch theorem is a result that describes algorithm performance in very general terms. It proves that providing a general algorithm that is equally efficient on all possible problems is not possible. It is a strong theoretical indicator of the merit of high specialisation of optimisation techniques developed during centuries [144, 145, 200].

2.5.1 Analysis of algorithms

Analysis of algorithms is an essential discipline in computer engineering and computer science. The term was invented by Donald Ervin Knuth and much of his monograph *The art of computer programming* published in 1968 [196] is dedicated to this, then new, branch of computer science. Analysis of algorithms provides useful insights about algorithm properties depending on inputs of various sizes, and allows comparison with other algorithms. Most important properties of an algorithm are the time and memory requirements—how much time will an algorithm spend and how much memory will it need, although use of other resources can be analysed as well (e.g. battery drain on mobile devices, number of comparisons in search and sort algorithms). Analysis of algorithms includes finding the worst-case, average and best-case performance of the algorithm, and how the resource requirements change with increases of the input size. This is performed by building mathematical models that describe algorithm computational complexity in sufficient detail level [196, 201].

The required time is one of the most important characteristics of an optimisation algorithm, since some of them can scale poorly. Algorithm runtime estimates are usually communicated using the *big-O* notation, also called *Bachmann-Landau* notation, that defines an upper asymptotic bound (worst case algorithm performance). It was first mentioned by mathematicians Paul Gustav Heinrich Bachmann in 1894 [202] and Edmund Landau in 1909 [203], who used it for asymptotic analysis of functions. This notation provides an upper bound for a function when argument value becomes large.

Formally, $O(f(n))$, where $n \in \mathcal{N}$ is the set of all functions that are asymptotically bounded from above as $n \to \infty$. For an individual function $f(n)$, it is said that it is asymptotically bounded from above as n becomes very big, denoted

$$f(n) = O(g(n)) \quad (n \to \infty) \tag{2.5}$$

if there exist positive constants M and n_0, for which

$$|f(n)| \leq M \cdot |g(n)| \quad \text{whenever } n \geq n_0. \tag{2.6}$$

In other words, this indicates that $f(n)$ grows equally or slower than $g(n)$. Using this way of communicating algorithm time requirements abstracts unnecessary detail such as differences in hardware and compilers, while it does not suppress useful information about general algorithm behaviour with different inputs [196, 197, 199, 201].

2.5.2 Computational complexity theory

Computational complexity theory is the area of theoretical computer science that studies the problem difficulty and investigates why some problems are hard to solve by computers, and the complex interactions between problems (tasks) and problem-solving methods (algorithms). Developing suitable models of computational devices, investigating their characteristics and limits, formal languages, language recognition, algorithms and problems are key study areas of computational complexity theory. It extensively uses analysis of algorithms and provides general insight into the properties of any algorithm that can be applied to a problem. It is very important in optimisation because it provides rigorously proved far-reaching discoveries about general problem properties and their intrinsic difficulty [196, 197, 199].

What is *fast enough*?

Techniques developed for analysis of algorithms and communicating the time complexity allow us to compare performance of different algorithms for different problem types. There exist algorithms that scale great, for example, the constant runtime algorithms, with complexity of $O(1)$, whose duration is not dependent on the input size n. Conversely, some algorithms require much more steps as input size increases, for example the algorithms whose complexity is exponential, such as $O(2^n)$. When the analysis of an algorithm is done and its time complexity is known, engineers and scientists need to be able to decide if the algorithm is fast enough to be suitable for practical applications.

A commonly accepted rule for deciding if the algorithm scales sufficiently well is to check if the time complexity of the algorithm can be bounded by polynomial time or not [197, 199, 204]. This rule creates two fundamental classes of algorithms:

- *polynomial* algorithms and
- *superpolynomial* algorithms.

Polynomial algorithms can be bounded from above by a polynomial function. For superpolynomial algorithms, this is not possible and they grow faster than any polynomial function [197, 199, 204].

The above classification is an essential criterion to decide suitability of an algorithm for use in practice, and is described in the *Cobham-Edmonds Thesis* [205]. Algorithms requiring polynomial time are usually suitable for practical use, while those that require superpolynomial time almost never are. Polynomials have reasonably slow growth, which is extensively confirmed in practice. Further, they have a very convenient characteristic of being “closed” with regard to the usual ways of algorithm composition—calling a polynomial algorithm as a subroutine from a polynomial algorithm produces a polynomial algorithm [204].

Key characteristic of superpolynomial algorithms that prevents practical applications is the very rapid growth of such functions. This growth is so fast that it implies prohibitively long solving times on any except the smallest problem instances. Most important class of superpolynomial algorithms are the *exponential* complexity algorithms such as $O(2^n)$. Note that all superpolynomial algorithms are sometimes also called *exponential* algorithms, even though this might not be strictly correct. Functions that have slower growth than exponential, but cannot be bounded by a polynomial such as $O(n^{\log n})$ are a good example [204, 206, 207]. In this work, the expression superpolynomial will be used, unless the algorithm complexity is bounded by a strictly exponential function $O(k^n)$, $k > 1$. Functions with even faster growth than exponential do exist, such as factorial $O(n!)$, $O(n^n)$ and double exponential $O(2^{2^n})$[153, 197, 206].

The above classification into polynomial and superpolynomial algorithms provides satisfactory results in a great majority of cases. However, it is still a very general rule of thumb, that has its limitations. The big-O notation that is standard in computer science captures general detail about the complexity growth, however, it suppresses details such as constant factors and low-order terms [153, 196, 208, 209]. Depending on these neglected details, exceptions in this general classification are possible. Consider for example the polynomial $10^{100}n^{73}$ and the exponential function $2^{0.01n}$. In this example, the polynomial function values are greater than the exponential even for large values of the argument[9], and this difference is especially notable for small n. While in practice such great degrees of polynomial complexity occur only exceedingly rare, the example is a good illustration that there might exist cases when exponential algorithms are suitable for use, or even outperform polynomial algorithms.

It should also be highlighted that the bounds given in this analysis are the worst-case bounds for algorithm behaviour. The bounds for the average input, might be lower, which might mean that algorithm is generally well suited for practical applications [153, 196, 204, 206]. There might exist broad subclasses of problem instances where very efficient solving is possible, as is the case with the simplex algorithm [87]. Therefore, one should be careful not to interpret the above guidelines as "*any* exponential algorithm is slower than *all* polynomial algorithms for *all* problem instances".

Turing machine

In the early 21st century saying *computation* is almost synonymous with electronic computers. According to the Merriam Webster English dictionary, computation is "the act or action of computing", or "the use or operation of a computer" [210]. Modern computers are complex systems. This makes it difficult to create formal mathematical models of behaviour of computers that include the specifics of all their components such as memory and multicore central processing units. To allow mathematical analysis, simpler models of computing machines were developed. They hide technical detail that is not necessary for general questions in the focus of theoretical computer science and computational complexity theory.

In 1936, Alan Turing proposed his abstract model of computation that represents a "purely mechanic process". He called it "a-machine", and used it in theoretical research on what can be computed by such processes [141]. It is now called Turing machine, and is an important tool in thought experiments of computer science. Turing machine is generally regarded as a highly simple abstraction of an algorithm, and it can perform anything that any modern computer can. In the strict sense, much like other models of computation, a Turing machine output is either acceptance or rejection of an input string [69, 197, 204, 206].

It can be proven that anything that a computer does in n steps can be performed on a Turing machine in at most $P(n)$ steps, where P is a polynomial function. Put in broad terms of the efficiency criteria discussed above, anything that can quickly be performed on a computer can

[9] Numerically solving the equation of the polynomial and the exponential, $10^{100}n^{73} = 2^{0.01n}$ shows that the polynomial is bigger for all $n \leq 159\,378$

also be quickly performed on a Turing machine, and vice-versa. This is an important result that indicates that computers and algorithms and Turing machines are equivalent—both in terms what can be computed using them, and in terms of similar speed of computation [211].

A *Turing machine* consists of a tape used as a storage and the "machine head" that can move on the tape in both directions. The tape of the Turing machine is infinite in length in both directions, and is divided into discrete blocks or cells. Each tape cell can be either *blank* or it can store a symbol from the *tape alphabet*. The machine head can read and write on the tape. Turing machine keeps an internal *state* at all times and is provided with a set of *instructions* that define how the head will move, and what symbols will be written under which conditions. Instructions are also called the *transition function*. Along with movements, reading and writing, the instructions also specify under which conditions will the input string be accepted and when the machine halts. The set of allowed tape symbols and the set of instructions are provided as inputs. Each input must also specify the initial configuration: the initial head state, the initial string of symbols written on the tape, and the initial head location. The Turing machine head then reads the initial symbol, and performs the following actions according to the instructions:

1. Updates the tape cell value to a symbol from the tape alphabet,
2. Updates the head state,
3. Moves the head position, either to the left tape cell, or to the right.

The above procedure is repeated until the *halting condition* is met—either the machine reached an accepting state, or there is no move defined for the current combination of the head state and symbol on the tape [69, 198, 206, 211].

Formally, a Turing machine M is defined as an ordered 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F), \tag{2.7}$$

where the tuple components are:

- Q is the finite set of *states* of the machine.
- $\Sigma \subseteq \Gamma \setminus \{B\}$ is the finite set of *input alphabet* symbols.
  It is the set of symbols that are used to specify the input on the tape before starting the machine.
- Γ is the finite set of symbols in the *tape alphabet*.
  Input alphabet Σ is always a subset of the tape alphabet Γ, since the tape alphabet contains a blank symbol B that cannot be used when defining an input.
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the *transition function*.
  The domain of the transition function $\delta(q, X)$ are ordered pairs of
  1. the machine state $q \in Q$, and
  2. the input symbol $X \in \Gamma$.

The value of the transition function, is a triplet (p,Y,d), where

1. $p \in Q$ is the next state,
2. $Y \in \Gamma$ is the symbol to write to the tape,
3. $d \in \{L,R\}$ is a set of possible movement directions, and L and R denote left and right.

The transition function δ can be undefined for some pairs of (q,X). If this happens while the machine is working, the Turing machine halts.

- q_0 is the initial state of the machine.
- B is the *blank* symbol.
  It is a part of the tape alphabet Γ, however it does not appear in the input ($B \notin \Sigma$).
- $F \subseteq Q$ is the set of acceptable states.

The abstract computational device defined above is also called *deterministic Turing machine*, abbreviated DTM. The deterministic Turing machine has a transition function that for a single argument (q,X) provides a single transition (p,Y,d) [69, 198, 206, 211]. Unlike the above device that can also be quickly simulated in reality on conventional computers, there exist purely fictious computing models.

A *nondeterministic Turing machine*, abbreviated NTM, is a fictitious computing device conceived for theoretical investigations. It is similar to the deterministic Turing machine, however, it has a different transition function, that might produce more than one triplet (q,Y,d) as the result:

$$\delta : Q \times \Gamma \quad \longrightarrow \quad \mathcal{P}(Q \times \Gamma \times \{L,R\}), \tag{2.8}$$

where $\mathcal{P}$ is the power set of triplets (q,Y,d), that contains all subsets of the set that contains all combinations of such triplets. Given this transition function, after each tape read, the nondeterministic Turing machine can proceed to a set of triplets $\{(q_1,Y_1,d_1),(q_2,Y_2,d_2),\ldots,(q_n,Y_n,d_n\})$. For each choice, the following transition will again be a set, and this way, computation using a nondeterministic Turing machine becomes a tree whose branches correspond to different paths of selected possibilities. The nondeterministic Turing machine halts and accepts the input string if there exists an acceptable state in any possible sequence of transitions from the initial configuration [69, 198, 206, 211].

Nondeterministic Turing machine is an abstract computing device, that was never intended as a model of something that could be realistically built. While a NTM can be simulated on a DTM, straightforward ways to achieve that are exponentially slower—if a task can be decided using an NTM in time $f(n)$, then it can be decided by a DTM in $O(c^{f(n)})$, where $c > 1$ is a constant. There is no known way to do this with only a polynomial slowdown. The previously shown equivalence between computers and DTM in terms of speed therefore also indicates that there is no known general way to simulate an NTM on a computer without a superpolynomial increase in time requirements [69, 198, 204, 211].

Complexity classes $\mathcal{P}$ and $\mathcal{NP}$

The most important achievement of computational complexity theory from the optimisation point of view is the theory of *$\mathcal{NP}$-completeness*. It relies on previously defined notation and abstract computing machinery to provide valuable insight into the intrinsic difficulty of problems, regardless of the specific choices of known algorithms. Using this theory, the complexity of problems and problem classes can be estimated and compared [197, 199, 204, 211].

Two key complexity classes provided by this theory are:

- class $\mathcal{P}$, that contains all problems for which there exists a polynomial-time algorithm,
- class $\mathcal{NP}-complete$, for which there is no known polynomial-time algorithm.

Put more formally, the class $\mathcal{P}$ contains all problems that can be solved in polynomial time on a *deterministic* Turing machine. Given the fact that any problem that is solvable on a deterministic Turing machine also corresponds to a problem that can be solved on computers in similar time, this implies that the class P is the class of all problems solvable on computers in polynomial time. Since all algorithms that require polynomial time when running on computers are considered to be quick enough for everyday use, the class $\mathcal{P}$ is regarded as class of problems that can be efficiently solved [197, 199, 204, 211].

The class $\mathcal{NP}$-complete is more complex to precisely define. Definition of the $\mathcal{NP}$-complete complexity class is based on the notion of

- the definition of the class $\mathcal{NP}$ and
- polynomial-time reduction.

The class $\mathcal{NP}$ is the class of all problems that can be solved in polynomial time on a *nondeterministic* Turing machine. Clearly, $\mathcal{P} \subseteq \mathcal{NP}$ [197, 199, 211, 212].

Definition of polynomial-time reduction is based on the idea of converting an instance of a problem to an instance of another problem. Let us consider a situation when we are given a decision problem (such that has an answer yes or no), denoted P_1, for which we only have a slow solution procedure S_1. It might be possible that this decision problem can be reduced to a different problem, P_2 for which a faster solution procedure S_2 is known. This would assume that there is a procedure that can produce an equivalent instance of the second problem p_2 for each instance p_1 of the initial problem. Equivalent here means that the answer $s_2 = S_2(p_2)$, given by procedure S_2 with p_2 as the input must be equal to the solution provided by the initial algorithm on the initial problem $s_1 = S_1(p_1)$ for each converted problem instance. A reduction is called a *polynomial time reduction* if it is possible to convert any instance of P_1 to an instance of P_2 in polynomial time [69, 197, 199, 213].

Given the definitions of $\mathcal{NP}$ and polynomial-time reduction, it is possible to formally define the class of $\mathcal{NP}$-complete problems. A problem c is in the class $\mathcal{NP}$-complete if it satisfies the following conditions:

1. c is in $\mathcal{NP}$, and

2. There exist a polynomial-time reduction of any other problem in $\mathcal{NP}$ to the problem c.

The second condition ensures that $\mathcal{NP}$-complete contains the hardest problems in $\mathcal{NP}$ and that it is “closed” with regard to polynomial reducibility. Problems that satisfy only the second condition are called $\mathcal{NP}$-hard, and they are considered at least as hard as any problem that can be solved by NTS in polynomial time. In (1) it is defined that any problem in $\mathcal{NP}$-complete can be solved in polynomial time on an NTS. Since there is no known way to perform calculations of an NTS without at least a superpolynomial slowdown, it implies that none of these problems can be solved on a computer in polynomial time [197, 199, 206, 211].

It should be noted that the definitions of this problem class commonly include the phrasing “no known”, and similar. This refers to the $\mathcal{P}$ vs. $\mathcal{NP}$ problem. It is one of the most famous open problems in modern computer science and mathematics, and it is equivalent to the question of existence of a polynomial reduction of any problem from $\mathcal{NP}$-complete to any $\mathcal{P}$ problem. It can also be stated as a problem of existence of a polynomial algorithm for any $\mathcal{NP}$-complete problem, or as a problem of simulating the NTS on a DTS with at most polynomial slowdown. Note that due to the fact of polynomial reducibility of any problem in $\mathcal{NP}$-complete to any other problem in that class, it would be sufficient to show that polynomial solving is possible for any problem in that class to allow polynomial solving of all other problems in it. Despite decades of effort of brilliant computer scientists, such an algorithm has not been discovered. Nevertheless, it is also not proven that such an algorithm cannot exist. Therefore the definitions of complexity classes are typically cautious to allow for the possibility that such algorithm exists and has not been discovered yet [197, 199, 206, 211].

The theory of $\mathcal{NP}$-completeness was started when *Cook-Levin* theorem was proved in early 1970s [143, 214, 215], however the term $\mathcal{NP}$-complete was introduced later [198, 216, 217]. The theorem was proved independently by Stephen Cook and Leonid Levin, and it shows that the Boolean satisfiability problem, i.e. setting the values in a Boolean logical formula, such that the formula evaluates as true. The proof states that any problem in $\mathcal{NP}$ can be reduced in polynomial time to the problem of Boolean satisfiability [143, 215]. The paper received great interest after Richard Karp showed that numerous practical problems are $\mathcal{NP}$-complete and he provides the famous list of *Karp’s 21 NP-complete problems* [36].

2.5.3 No free lunch theorem

In 1997, David Wolpert and William Macready published the *no free lunch theorem*, abbreviated as NFL-theorem [145]. It is another crucial achievement of theoretical computer science that has direct consequences in the entire area of optimisation. In the recent decades, great effort was devoted to both *specialisation* as well as *generalisation* of algorithms. Specialisation is usually used when developing high-performance techniques that work great on a narrowly defined problem, with the goal of using as much problem-specific features possible to improve the efficiency. The attempts to achieve generalisation are especially prominent in the area of metaheuristics, where numerous general optimisation techniques were proposed. It is natural to ask—of all the possible problem-solving methods for a problem at hand, which one is the best.

The no free lunch theorem provides a formal mathematical insight into this question. Let us define the sets $\mathcal{S}$ and $\mathcal{Y}$ as two finite sets, where $\mathcal{S}$ is the set of possible solutions, and $\mathcal{Y}$ is the set of their values (e.g. given by an objective function to each element of $\mathcal{S}$). Further, let us define $f : \mathcal{S} \rightarrow \mathcal{Y}$ to be a mapping between solutions and their values. We denote a set of all possible such mappings $\mathcal{F} = \mathcal{X}^{\mathcal{Y}}$. We now define $T_m^s = \{s_1, \ldots, s_n\}$ to be a sequence of solutions with the length m, considered by an optimisation algorithm while finding the optimum of an optimisation problem corresponding to f. Analogously, let us define $T_m^y = \{y_1, \ldots, y_m\}$ to be a sequence of the corresponding values for each possible solution $f(s_1), \ldots, f(s_m)$. Finally, let us use $P(T_m^y \,|\, f, m, a)$ to denote a probability tht the search algorithm a will in m steps find precisely the sequence of values T_m^y.

Then, the no free lunch theorem states that for each pair of algorithms a and b,

$$\sum_{f \in \mathcal{F}} P(T_m^y \,|\, f, m, a) = \sum_{f \in \mathcal{F}} P(T_m^y \,|\, f, m, b) \tag{2.9}$$

We use $\Phi(T_m^y(a, f)$ to denote the function that gives performance of an algorithm a that performed m steps while optimising the function f. A typical algorithm performance function could be the minimum value y_j that the algorithm has found. Then, from the no free lunch theorem it follows that any performance function averaged on the set of all functions $\mathcal{F}$ is independent of the algorithm a. Conversely, for any two algorithms a and b, there exists two functions f and g, such that the sequence of values obtained using algorithm a to optimise function f is equal to the sequence of values using the algorithm b to optimise g.

The *sharpened* version of the no free lunch theorem brings the set of theoretical conditions that a subset of the set of all functions $\mathcal{F}$ must satisfy in order for the theorem to hold on the subset too [218]. These are defined in purely mathematical terms that can be difficult to check. Still, it is shown that the sharpened NFL holds only on trivial subproblems of e.g. symmetric travelling salesman problem, and not for the symmetric TSP in general. Similar results are reported for several other classical combinatorial optimisation problems [219].

2.5.4 Implications for optimisation

The elaborated achievements of theoretical computer science have direct implications when developing optimisation algorithms. After Karp's 21 $\mathcal{NP}$ problems [36] were published, the awareness grew about the fact that numerous optimisation problems that have high importance in practice belong to the class of intrinsically difficult problems. Further, numerous attempts to provide fast and exact solutions were not successful. This leads to very strong guidelines when developing optimisation problems, that can be summarised as follows:

1. Try developing a simple and fast exact algorithm for the given problem,
2. If you fail in doing (1), try proving that the problem is $\mathcal{NP} - hard$,
3. If you successfully prove (2), use approximate methods.

The above general rule is based on the fact that not only algorithms need to provide solutions quickly—the algorithm developers are also encouraged to produce efficient algorithms as soon as possible. When facing an $\mathcal{NP}$-hard problem, the algorithm developer must know that the problem to solve is so difficult that despite decades of effort of most brilliant minds in computer science and mathematics, such an algorithm was not found. This means that it is not viable to spend time on an unsolved problem, and therefore the recommendation (3) is given for such cases. Continuing with the attempts to produce an efficient solution is considered as a valid research direction only when conducting research of theoretical computer science [199].

Similarly to the theory of $\mathcal{NP}$-completeness, the no free lunch theorem has also shown to have far-reaching conclusions and shaped the way modern optimisation algorithms are developed. It is a strong indicator that efficiency can be achieved only by means of specialisation. It also provides an important point of view when performing common development tasks. For example by tuning an algorithm, its efficiency increases on a particular subset of problems. In the no free lunch framework, better performance on a subset of all functions also means that the performance can drop on other problems, however this might be hidden to a developer unaware of the NFL-theorem. Given the wide availability of various different types of metaheuristic algorithms, and the general intent to provide as much generality as possible in such methods, the NFL-theorem suggests that the best metaheuristic for a given problem is the one that can be best adapted to the problem [8, 220]. It can also be argued that it shaped the metaheuristic research to move from the method-centred research towards a framework-centred, where metaheuristics are viewed more as a framework to develop heuristics than different methods.

2.6 Solving difficult problems

The superpolynomial dependency between the input problem size and the number of steps to complete the exact methods applied to $\mathcal{NP}$-hard problems is an indicator that simply waiting for more efficient hardware to appear is not an option when facing problems of such complexity [8]. Let us consider an exact algorithm that needs to complete 10^{100} steps to complete. In the most optimistic case, assuming that a single processor cycle is sufficient for one algorithm step, with a conventional 3 GHz processor available in 2019, this algorithm would require $2.6 \cdot 10^{82}$ years to complete. In practice, problems of such and even greater complexity are common in combinatorial optimisation. Given the current state of microprocessor industry, microprocessor units with 100 times more computational capacity than currently available are not very likely to appear in the next few years. Since the paradigm in increasing computational capacity is not increasing the clock rate, but instead adding more cores to the processor chips [221], this would also mean less than 100 times faster solving, since not all exact algorithms can be easily parallelised [222, 223, 224]. Even in the most optimistic case, when this would lead to 100 times faster solving, this speed increase would not mean much for the practical applications in the given example—using 100 times faster hardware, the algorithm would now require $2.6 \cdot 10^{80}$ years. This also leads to the conclusion that such algorithms are not suitable even when utilising large cloud or grid resources—a speedup factor of 100 or even 5000 in available CPUs does not mean much for exact algorithms and problems of this size. Quantum computing is an exciting research area, and for general search problems, it can achieve quadratic speedups [225, 226, 227]. While quadratic speedups would significantly improve the capability to solve smaller problems, there is still no practical benefit from these speedups in the larger problems. It is unclear if it can achieve more significant breakthroughs in the $\mathcal{NP}$ class in general. Despite efforts like IBM Quantum cloud [228], quantum computing hardware is still expensive and impractical for use [229].

The above discussion shows that the hardware sufficient for applying currently known exact methods to solve $\mathcal{NP}$-hard problems will not be available anytime soon. This means that the effort to solve such problems with more success must lie in development of better algorithms, which motivated a great research effort in the area of combinatorial optimisation, including approximate methods. In the years to come, unless there is a breakthrough that would show that $\mathcal{P} = \mathcal{NP}$, efficient exact solving of anything but toy-sized instances of these problems will not be viable in practice, and providing good solutions will inevitably involve approximate methods [8].

2.6.1 Developing good optimisation algorithms

Optimisation algorithms start with an initial guess of the solution (or a population of solutions), and then iteratively attempt to improve it and approach the optimum [69, 87, 105, 230]. The way they change the variables during this process can be viewed as a series of movements through the domain, and each algorithm has its unique way in doing so. For any optimisation algorithm, it is desirable that it is efficient, as general as possible, and accurate. Efficient algorithms produce solution in short time, without too much computational resources. General algorithms are robust enough to be applied to a wide variety of problem instances, and tolerate different choices of initial points. Finally, an algorithm must be accurate in terms of precisely setting variables to values that make sense, and it must be able to tolerate minor errors in the input and due to rounding when representing numbers in computer memory [87].

These goals can be contradictory to each other, especially the generality and efficiency. A very fast algorithm might require a lot of memory. Therefore a balance between performance and available resources also needs to be kept in mind when developing optimisation algorithms [87].

2.6.2 Superpolynomial exact algorithms

For $\mathcal{NP}$-hard problems, the best-known algorithms have superpolynomial complexity. This is in general considered a very unfavourable option, that frequently leads developers to use approximate methods without even considering exact methods. Still, it should not be forgotten that they might indeed be practical for solving small problem instances. They can be an attractive option when the problem can be solved using algorithms that are trivial to implement, and we are sure that we will only have sufficiently small inputs [69]. Before attempting to do that, nevertheless, it is necessary to perform thorough algorithm complexity analysis and ensure that problem instances that will be too large to solve will indeed never be encountered. It is crucial to communicate these restrictions with users, who should have enough information to understand the limitations of this approach.

Exhaustive search

A very simple exact method in combinatorial optimisation is called *exhaustive search*, or *brute force* algorithm. It successively enumerates all the elements in the problem domain, and keeps the one with the lowest objective function value as the result. Such algorithms are usually very simple to implement, and guarantee finding an optimum. However, as any other exact method, they scale poorly, and for complex combinatorial structures, the implementation might not be trivial [69, 231].

Identifying subproblems and probabilistic analysis

An important reason for performing the complexity analysis before starting work on the algorithm is the fact that some problems that are in the $\mathcal{NP}$-hard class might have special cases that are easy to solve. If the users focus solely on such easier instances, specialised fast exact algorithms for this subset of the problem could be applied. Even when the problem does not have special cases that are easy solvable, probabilistic algorithm analysis might reveal that an exact algorithm performs good on *most* problem instances, with the worst-case scenario happening sufficiently rarely to consider using the algorithm in practice [69]. Unlike with applying exhaustive search, it might be more difficult to understand on which problems the algorithm can work, and on which it cannot. This requires very careful communication with users to ensure they understand the limits on the algorithm applicability.

2.6.3 Approximation

Approximate methods include approximation algorithms, heuristics and metaheuristics. They are not guaranteed to produce an optimum, however, they use various strategies to get as close to the optimum as possible. The key differences in these methods are in

- guarantees of closeness to the optimum they provide, and
- their generality and applicability to different problems.

Approximation algorithms

Approximation algorithms are fast algorithms that can produce a feasible solution and provide strict guarantees regarding the closeness to the optimum, expressed as a factor or percentage of the objective function value at the optimum [69, 163]. For example, an algorithm might guarantee that the solution will be less than or equal to the double of the optimal value. While this is an appealing property, approximation algorithms can be complex and difficult to implement. They are usually highly specialised, and rely on problem-specific features. Therefore they cannot easily be modified for use on other problems. Further, for several classes of problems, there are proofs that fast approximation algorithms with tight closeness to optimum cannot exist, unless $\mathcal{P} = \mathcal{NP}$ [69, 232].

Heuristics and metaheuristics

Heuristics are approximate methods for which there is no formally provable ratio between the worst-case result and the true optimum. They are usually specifically designed for a certain problem and during the optimisation process, they use problem-specific features to speed-up the search for the optimum [8, 69, 233]. *Metaheuristics* are algorithmic frameworks that allow quick development of heuristics. They do not focus on efficiently solving a single problem, instead they are generic search strategies that allow searching for good solutions in very large domains. This thesis focuses on expanding the applicability of metaheuristics to problems where they are difficult to apply, as well as improving the implementation development methodologies [105, 129, 234].

When implemented properly, heuristics and metaheuristics can provide excellent results. The fact that they provide no guarantees on the solution quality, is not an attractive property from the analytical and purely mathematical point of view. However, in many practical applications, such as engineering, construction, and especially business, to consider an algorithm successful, it is sufficient that the algorithm outperforms computation by hand, or some other simple technique. The benefits of using heuristics are even more obvious when the only alternative are exact algorithms that are too slow to consider. Given the prospect of waiting years or even centuries for the algorithm to finish, a slight decrease in solution quality is a great tradeoff that allows getting solutions in short time. While their speed is a good argument to use heuristics, the lack of guarantee on the solution quality does not necessarily mean that the solutions provided by them are not good. For many problems of great practical importance, heuristics and metaheuristics provide state-of-the-art results [8, 235].

2.7 Noteworthy problems

Following the discovery that the widely studied *boolean satisfiability problem* is $\mathcal{NP}$-complete, during the 1970s [143, 214, 215], an entire array of other problems of equal difficulty was discovered. All these problems are characterised by the fact that they are the most difficult problems in $\mathcal{NP}$, and that each of them can be polynomial-time reduced into any other problem from this class. An important milestone was the publishing of Karp's 21 $\mathcal{NP}$-complete problems in 1972 [36]. This paper demonstrated that being $\mathcal{NP}$-complete is not an isolated curiosity of the boolean satisfiability problem, and that such problems of great practical importance are fairly common. Several other important problems were later proven to be at least as difficult, such as *vehicle routing problem* [236] *timetabling problem* [199, 237, 238] and others.

Boolean satisfiability problem

The *Boolean satisfiability problem* is the problem of checking the existence of an assigment of variables that evaluates a given Boolean formula to *true* [69, 197, 239]. Boolean variables are values that can assume values *true* or *false*, and Boolean logic formulas are built from Boolean variables using the logical *and* ($\wedge$), *or* ($\vee$) and *not* ($\neg$) operators. Each Boolean formula can be transformed into an equivalent standardised form called *conjunctive normal form*, which consists of several *clauses*, connected using the *and* operator. Each clause is a logical formula that can only be a disjunction of literals, where a *literal* is either a variable (x) or a negation of a variable ($\neg x$). In the example below, five boolean variables $x_1, \ldots x_5$ are used in a Boolean formula that already is in a conjunctive normal form:

$$(x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_3) \wedge (\neg x_4) \wedge (x_1 \vee x_5) \tag{2.10}$$

For each given assignment of logical variables, the entire formula evaluates to either *true* or *false*. Formulas that are *satisfiable* evaluate to *true* for some assignment of variables. Conversely, formulas that are not satisfiable can never evaluate to true, and contain an intrinsic *contradiction*. Using the conjunctive normal form representation, for the formula to be satisfiable, there must exist an assignment of variables that evaluates all clauses as *true*. The above example is a satisfiable formula since it evaluates to true for $x_1 = x_2 = x_5 = true$, $x_3 = x_4 = false$.

Formally, given a set of Boolean clauses $C_1, C_2 \ldots C_n$, where C_i is a clause, the boolean satisfiability problem is defined as the problem of checking if the formula $C_1 \wedge C_2 \wedge \ldots \wedge C_n$ is satisfiable [69, 197, 239]. It was the first problem proven to be $\mathcal{NP}$-complete [143, 214, 215, 240]. Efficient solving of satisfiability problems is important in practice for theorem proving algorithms, circuit design, formal verification of hardware and software. While all best-known algorithms have exponential worst-case complexity, it was demonstrated that most randomly generated problem instances are fairly easy to solve. In practice, great results are achieved, and problems with more than million variables have been successfully solved [239, 241].

Integer linear programming

The *integer linear programming problem*, abbreviated ILP, is an extension of the linear programming model, in which all the variables are restricted to integers [87, 242, 243, 244, 245]. More formally, the problem is stated as follows:

$$\text{minimise } \mathbf{c}^{\mathsf{T}}\mathbf{x}, \tag{2.11}$$

$$\text{subject to } A\mathbf{x} = \mathbf{b}, \tag{2.12}$$

$$x_j \geq 0, \qquad \text{for each } j \leq n, \tag{2.13}$$

$$x_j \text{ integer}, \quad \text{for each } j \leq n, \tag{2.14}$$

where $\mathbf{x}$ is a vector of n variables, $\mathbf{b}$ and $\mathbf{c}$ are integer vectors, and A is an integer matrix [242, 243, 244, 245]. The problem is $\mathcal{NP}$-complete [197, 246]. The restricted problem, when the variables in $\mathbf{x}$ can be only zero or one is called *0-1 integer programming*, and in this version, it is one of Karp's 21 $\mathcal{NP}$-complete problems [36, 243, 244]. Solving such problems is important in production planning, when constraints are linear, and the variables must be integral or boolean. Scheduling and transportation problems also can be defined as integer linear programming problems.

Important class of subproblems that can be efficiently solved was identified—if the problem is only to solve the system of equations that define the constraints and the number of variables or constraints is fixed, then there exists a polynomial algorithm to solve it [247]. While the problem formulation is analogous to the problem of linear programming, the integrality constraint causes the problem to be more complex to solve than linear programming with real variables. Since general linear programming can be solved using fast algorithms, and given the apparent similarity of the problem, it might be tempting to ignore the integrality constraint, and solve these problems as if they are linear programming with real variables. Solutions of this could then be rounded to nearest integer values. This approach is called *LP relaxation*. Such solutions are not guaranteed to be optimal or to satisfy all the constraints, however they can be used as good starting points for exact algorithms [244]. For the general ILP, strategies such as *branch and bound* can be used to find exact solutions, however, branch and bound has exponential worst-case complexity [248]. Heuristics and metaheuristics such as tabu search are also used as a solving method [131].

Knapsack problem

The *knapsack problem* is formally defined using integer linear programming, as follows:

$$\text{maximise} \sum_{j=1}^{n} c_j x_j, \tag{2.15}$$

$$\text{subject to} \sum j = 1^n w_j x_j \leq K, \tag{2.16}$$

$$x_j \, \text{integer}, \tag{2.17}$$

where c_j is the value of each item j, w_j is the *weight* of each item, akd K is the capacity. Given a knapsack whose capacity is K kilograms, and a choice of items with known weights w_j, $j = 1, \ldots, n$, the problem is to decide which items to fit into a knapsack, so that the total value of the selected items is highest, while not exceeding the knapsack capacity. In the above version, also called *unbounded knapsack problem*, there is no limit on the number of items that can be selected, assuming infinite number of each item can be selected [69, 243, 249].

There exist several variants of the knapsack problem. If there is a limited availability of each item, then the constraint 2.17 is replaced with $0 \leq x_j \leq b_j$, where b_j is the available stock for each item j, and such problem is called *bounded knapsack problem.* A problem might be restricted to selecting only a subset of n, available items, called *0-1 knapsack problem*, that has the constraint 2.17 defined as $x_j \in \{0, 1\}$. Additional constraints can be added to the problem, e.g. specifying the dimensions of the items in addition to their weight, and such problems are called *multidimensional* or *multiconstraint knapsack problem* [69, 243, 249]. All versions of the knapsack problem are $\mathcal{NP}$-hard, and the 0-1 knapsack problem is $\mathcal{NP}$ complete [199].

There exist techniques that can produce exact solutions to the problem in less than exponential, however not polynomial time. Exact solving techniques include dynamic programming and branch and bound approach [199, 249, 250, 251, 252]. Given the fact that it is simple to model, the problem is also an attractive benchmark for other combinatorial optimisation methods, including approximation algorithms and metaheuristics [253, 254, 255, 256, 257].

Bin packing problem

The *bin packing problem* is a generalisation of the knapsack problem. Given a number of *items* with known weight, and a number of *bins*, where all bins have equal capacity, the problem is to fit all items into bins in a way that does not exceed the capacity of any used bin, using as few bins as possible [69, 242]. The problem is $\mathcal{NP}$-hard and exact algorithms are too slow for practical use [197, 199, 258]. However, approximation algorithms are successful on the problem, and even for the simple heuristics such as first-fit decreasing[10] it can be proven they provide results close to the optimum [259, 260, 261, 262].

The problem can be generalised by specifying that e.g. each bin can have a different capacity, a variant called *variable sized bin packing problem* [258]. Another generalisation is the *multidimensional bin packing problem* that has several flavours as well. In the *2-dim* version, the problem is to pack a set of rectangles in a number of rectangular bins, where rotation might or might not be allowed. In *d-dim vector packing*, each bin can have multiple constraints, e.g. weight and volume. In this case, bin capacity and item weights are multidimensional vectors. The sum of all the items in each bin is also a vector, and the components of the total sum vector must not exceed the corresponding components in the capacity vector [263, 264]. It is proven that this problem has no efficient approximation algorithms [265].

Travelling salesman problem

Given a set of cities to visit, and the distances between each pair of cities, the *travelling salesman problem* (abbreviated TSP) is to find the shortest closed path that visits each city exactly once. In terms of graph theory, it corresponds to the problem of finding the shortest Hamiltonian cycle in a complete weighted graph, where edge weights represent distances between cities represented by nodes. Any valid solution starts in the initial city, visits each city exactly once and then returns to the initial city [69].

It is one of the most famous combinatorial optimisation problems, that has been widely studied by generations of mathematicians and computer scientists. It is trivial to define and notoriously difficult to solve. The simplest solving approach, by enumerating all the possible solutions scales terribly—it has time complexity of $O(n!)$. Several exact methods have been developed, however all of them have exponential worst-case complexity. Despite decades of effort, no exact algorithm faster than $O(2^n)$ is known [197, 266, 267, 268, 269]. In its general form, the problem is not well suited for approximation algorithms, however there are subproblems that can be well approximated [232, 270, 271, 272, 273]. Various heuristics and metaheuristics have been used, both in practice and in research. The TSP is an especially attractive problem to test new algorithms, due to the simplicity as well as high theoretical and practical importance [124].

[10] The first fit heuristic first sorts the items by their weight, from largest to smallest, then packs them using the following two rules:

1. Try fitting the current item in the first bin that has sufficient space,
2. If no bin has sufficient space, allocate a new bin for the current item.

Vehicle routing problem

The *vehicle routing problem*, abbreviated as VRP is a generalisation of the travelling salesman problem—given a set of vehicles and a set of locations (*customers*) to visit, the problem is to find the shortest path that visits each location exactly once by any of the available vehicles. This transportation problem is very important in numerous practical applications. Any company that deals with physical goods can benefit from efficient transportation of the product, and successful solving of practical variants of the vehicle routing problem can produce higher customer satisfaction, as well as great savings in transportation costs [274]. The problem was first proposed by Dantzig and Ramser in 1959 [66].

Due to the wide adoption in practice, depending on the specific requirements of each transportation process, there are numerous extensions of the above basic problem definition. In the *vehicle routing problem with time windows*, the time window $(t_{earliest}, t_{latest})$ is specified for each customer, along with the *service time*, an estimated time that the vehicle must spend at each location to finish the delivery. In good solutions, each delivery must start during the requested interval. The *capacitated vehicle routing problem* defines a *capacity* of each vehicle, and the *demand* of each customer, both usually in kg. This problem variant adds the capacity constraint—no vehicle can carry more cargo than the capacity of the vehicle allows [274].

The vehicle routing problem is $\mathcal{NP}$-hard [274, 275, 276, 277, 278]. There is considerable progress in improving the speed of exact techniques, including impressive exact solutions of two large problem instances with 400 and 1000 customers [279]. However, the problem is very difficult to solve using exact techniques, and in most cases they are limited to small problem instances with up to 200 customers [280, 281, 282]. Solvers used in practice usually use some type of a multiphase heuristic solving. Metaheuristics have been widely applied as well [283, 284, 285, 286, 287, 288, 289, 290, 291].

Scheduling problems

In the most general sense, *scheduling* is deciding which resources will be doing which tasks at what time. They commonly appear in manufacturing (which machine should be processing which input), service industries (which staff member will work at which position at what time), transportation (which flight attendant will work on which flight), education (school timetabling), etc. Depending on the application, the problem definitions can be very diverse [292]. Numerous scheduling problems are $\mathcal{NP}$-hard, such as job shop scheduling problem, cloud task scheduling, university course timetabling and call centre scheduling [199, 237, 238, 293].

In some scheduling and timetabling problems, the constraints are divided into hard and soft constraints. Hard constraints are rules that *must* be satisfied in any schedule. Typically, they are the most basic rules, such as "do not put more people in a room than the room can fit". Not satisfying such rule, by e.g. scheduling 100 people in a classroom that can accommodate 20 would clearly prevent such a schedule from being used in practice. Conversely, *soft constraints* are preferences that increase schedule quality, however, they are not a strict requirement. They need to be minimised, however, completely avoiding might not always be possible. Specifying "teachers prefer to arrive to work at the same time each day" could be an example soft constraint. When modelling scheduling problems, in terms of the optimisation problem as stated in Equations 2.1 - 2.3, hard constraints can be translated into constraints (Equation 2.2 and 2.3), while soft constraints can be encoded into the objective function [294]. It is common for scheduling problems to be highly constrained [295, 296, 297, 298].

The *job shop scheduling problem* is to schedule jobs on m machines, so that each job consists of a sequence of *operations*. Sequence of operations is strictly ordered for each job, and for each operation, a machine on which the operation takes place and processing time are defined. Processing of each operation must be uninterrupted, and a single machine can work on at most one job at a time. There are no precedence rules for jobs and each job can be processed only on a single machine at a time. The objective of the job shop scheduling problem is to find a sequence of operations for each machine that respects the constraints and minimises the *makespan* of the entire project: the time between the first job start and the completion of last job [292, 299]. This problem was a subject of great research interest, and it was proven to be $\mathcal{NP}$-complete for any problem with two or more machines [199]. Popular solving techniques include branch and bound (exact), heuristics and metaheuristics [300, 301, 302, 303, 304, 305, 306].

The *cloud task scheduling problem* is to distribute the virtual machines in a cloud computing service to available physical hardware. With the popularisation of cloud computing, efficient usage of hardware resources is becoming increasingly important to achieve good performance with reasonable use of computer resources [307, 308, 309, 310, 311]. The problem can be formulated as the aforementioned d-dim bin packing, where requirements such as memory and CPU are the constraints related to physical machines [312]. As a separate problem, a novel variant of bin packing is proposed, in cases when virtual machines can share parts of their memory (e.g. the operating system) [313, 314, 315]. Large part of cloud scheduling techniques are *online* algorithms, that can dynamically schedule tasks incoming one-by-one, and work without complete information about tasks that will arrive in the future [316]. Along with minimisation of running machines, it is frequently an attractive option to also model energy consumption and minimise energy consumption as one of the objectives [314, 317, 318, 319]. If a bin packing model is used, the problem is at least as hard as the bin-packing problem, therefore it is $\mathcal{NP}$-hard [317].

The *university course timetabling problem* is a problem of finding good schedules for university course that keeps students and as satisfied as possible. The problem is to schedule students into rooms where lectures are held. The problem is defined as a set of events, classrooms, students and timeslots. Each course is a set of events or lectures. For each event there is a list of students that attend the event and a set of required features related to the room where the event can take place, e.g. some events might require a computer to be installed in the room. Further, to be suitable for the event, any room must have sufficient capacity for all the attending students. For each room, a set of features it offers and the capacity is known. For each student, a subset of enrolled courses is defined. Finally, a set of time slots in which all events can take place are defined. The problem includes the following hard constraints:

- A student can be in only one place at a time,
- Classrooms have sufficient capacity for events held in them,
- Classrooms satisfy all required features for events held in them,
- At most one event can be held in a classroom at any time.

Possible soft constraints include:

- A student should not be scheduled in the last time slot of the day,
- A student should not have more than two classes without a break during a single day,
- A student should not have only one class in a day.

A complete solution is an assignment that places each event in a room and a time slot. Good schedules satisfy all hard constraints and minimises soft constraint violations [297, 320, 320, 321, 322]. The problem is $\mathcal{NP}$-hard [323]. It was first defined in the *Metaheuristic network* research project [324]. The problem was successfully solved using metaheuristics [297, 320, 320, 321].

The *call centre scheduling problem* is a problem of producing work schedules for call centre staff with the primary goal of reducing customer waiting times. Call centres are offices organised to answer large volume of customer calls and are frequently used by companies to interact with their customers. Prominent feature of call centre scheduling is *variable demand*, that might be known only approximately. A centre might have only a few incoming calls in the morning, and a peak with hundreds of calls around 17:00, however the exact number and duration of calls that will take place cannot be known in advance. Call centre scheduling usually starts with a forecasting step in which various techniques are used to estimate the number of needed people for each time unit in the scheduling period. With this information, the scheduling problem is to decide which staff member comes to work at which time, in order to minimise the difference to forecasted ideal staff number [325, 326, 327]. While there exists a polynomially solvable restriction of the staff scheduling problem [328], general staff scheduling is recognised as *NP*-hard, even in the simplest versions [199, 329, 330]. There are numerous constraints such schedules must satisfy. Total working hours a person can do in a day and in a month without causing too much fatigue is limited, and usually even strictly enforced in labour laws. Duration of the working day might differ, depending on the employee type, and part-time staff might be available only a day or two per week. Since each organisation might have various specific constraints, problem might be highly constrained [331]. Various techniques such as dynamic programming, mixed integer programming, linear programming relaxations, heuristics and metaheuristics have been used for solving the call centre scheduling problem [325, 327, 332, 333, 334, 335, 336, 337, 338, 339].

Chapter 3

Metaheuristic methods

This chapter presents an overview of early heuristic design principles, followed by short descriptions of established metaheuristics. This theory establishes the requirements for the analysis of components of metaheuristics and allows their faster development. This Chapter further brings an overview of current development methodologies for metaheuristic algorithms and addresses an important assumption on the quick evaluation of solutions. “Slow evaluation” can be an obstacle that prevents a successful implementation. This thesis later brings some adaptations to the ILS algorithm to show how it can still achieve good results even in cases when evaluation function takes time, or conversely, when number of evaluations is limited.

3.1 Definitions

The word *metaheuristic* comes from the Greek prefix *meta* that means “after” or “beyond” [340, 341] and the word heuristic, which also comes from Greek and means “I find” or “I discover” [340, 342]. In the modern sense, the word “metaheuristic” implies a higher-level abstraction beyond heuristics. The term was first proposed by Glover in 1986 [129]. The contemporary definition of metaheuristics that will be used in this work is from Sörensen and Glover [234]:

> *“A* metaheuristic *is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework.”*

Another commonly accepted definition, from Handbook of metaheuristic [150] is is that they are

> *“Solution methods that orchestrate an interaction between local improvement procedures and higher-level strategies to create a process capable of escaping from local*

> *optima and performing a robust search of a solution space.”*

The second definition highlights that aside from being general frameworks to develop optimisation algorithms, they all share similar working principles. According to this definition, any metaheuristic must have three basic components: (*i*) a local improvement procedure that finds local optima, (*ii*) strategies that prevent the algorithm being stuck in a local optimum too long without exploring other areas of the solution space and (*iii*) a strategy to decide when should each of these two principles be used, depending on the current state of algorithm progress.

Development of these techniques is also an attempt to add more generality to *heuristics*, which are highly problem-specific. Developing an efficient heuristic requires an in-depth understanding of the intricate features of the problem being solved, and it sometimes takes a detailed interdisciplinary effort to allow building such algorithms. It is not a very efficient strategy to try solve a problem using a heuristic developed for other problems, since other problems usually have a completely different structure with their own set of subtle features that need to be used to find good solutions [8].

Like heuristics, there is no guarantee that using a metaheuristic will provide an optimal solution, nor any guarantees on how close to the optimum the results will be [8, 69, 150, 162, 234]. As elaborated in Section 2.6.3, there exist difficult problems, for which exact methods are prohibitively slow. The goal of metaheuristics is to willingly give up on the guarantee of optimality on such problems, and be fast [8, 69, 234].

As summarised in Section 2.1.3, metaheuristics are inspired either by natural processes such as evolution or by general ideas on search strategies such as “do not go back to already explored areas”. They are non-deterministic algorithms that largely rely on randomness . Metaheuristics are high-level frameworks. Further, metaheuristics are not finished algorithms that simply need to be coded in the desired programming language. Instead, they provide general descriptions on how to work with solution components in various circumstances, and especially with regard to decisions that improved the solution. The work of connecting the abstract guideline such as “mutate the solution” in the genetic algorithm framework with precise changes in the variables of a candidate solution is left to the developer to decide on[150, 162, 343, 343]. Developing an efficient metaheuristic algorithm can be a difficult research problem [8, 283, 344, 345, 346].

Unlike heuristics that require an extensive understanding of subtle details of the problem being solved, metaheuristics in general assume very little requirements in order to apply them to an optimisation problem. The ability to evaluate different points in the solution space, a deterministic objective function, and the freedom to decide on the variables are all that is needed to implement a metaheuristic. No detailed understanding on how the objective function is needed, in fact there exists an area of *black-box optimisation* that deals with developing good optimisation techniques when the objective function is not known in an analytical form and acts as a "black-box" to the developers [96, 162, 343, 347, 348, 349, 350].

They have shown to be very effective in problem-solving. Given their low requirements, they are usually used to solve problems where all other methods fail. Such problems include hard and large scale combinatorial optimisation problems. These problems do not have favourable characteristics that help find the optimum. Their objective function is not smooth and can have great variations for small changes in the variables which makes it difficult to decide in which direction to move in the solution space. We are usually provided with very little information on how to set the variables to find good results, and it is not possible to use some reasonable rules to deduce the values of the optimum (or near-optimal) point. Finally, given the large scale of the problem, the simple exhaustive search is also not a good way to search for good results. Given the very few requirements they have, metaheuristics are an excellent choice for problems that fit the above description. They provide reasonable guidelines on how to move in the vast search spaces, without knowing anything else than the objective function value. The class of $\mathcal{NP}$-hard problems satisfies all the above conditions, and it is not surprising that these problems are the most typical problems solved by metaheuristics [96, 150, 162, 234, 343].

The great level of generality also implies that metaheuristics can be applied to almost any optimisation problem. This is a highly favourable option when solving practical problems. For numerous important problems, they are providing state-of-the-art results [127, 150, 351]. A number of software packages that use metaheuristics has been developed, and their role is especially notable in vehicle routing and scheduling [283, 352, 353]. They are common in simulation software to allow simulation optimisation, such as OPTQuest, developed by OptTek [354, 355, 356]. Finally, they are widely used in more general commercial packages for modelling and solving optimisation problems, such as IBM ILOG CPLEX Studio, which uses an evolutionary algorithm to improve the result of their ILP solver [357, 358, 359]. Libraries such as COIN OR, OptaPlanner and LocalSolver also include several metaheuristic frameworks [360, 361, 362, 363].

Criticism of metaheuristics includes the already mentioned fact that they are not completely finished algorithms—instead, there is considerable work needed to bring the general and problem-specific together in an implementation of a metaheuristic algorithm, especially if high performance is required [8, 234, 235, 364]. From the software industry point of view, metaheuristics still require expert knowledge and their development is expensive and requires a lot time. The methods have been criticised as difficult to understand as well, with a large number of complex operators [234, 235, 364, 365]. The area of metaheuristics has several established methods. Further, tens of methods published during the "metaphor controversy", have questionable novelty and add even more complexity [96, 234]. Therefore, newcomers and outsiders of the metaheuristic community that simply want to apply an efficient method to solve their problem are frequently discouraged from using metaheuristics. Finally, there is a general lack of development methodologies that would work well in the software industry and help save development time and costs. Almost all research in academia is focused on achieving better and better results on simplifications of practical problems [96, 234, 365]. Very little research is devoted to inventing methods that are simple to understand and quick to implement under a well-defined development methodology [96, 234], and the fact that ease and simplicity of implementation are equally important, is generally ignored in academic work.

This thesis attempts to help address some of these criticisms. By viewing the array of popular metaheuristics as frameworks that can be assembled from independent reusable blocks, it is possible not only to easily create hybrid metaheuristics with high performance [235, 357, 366, 367], but also save development cost. Numerous companies are not interested in finding the formally proven optimum. What they need are algorithms that produce *good enough* solutions for their client's demands that might change during time. By selecting the simplest metaheuristic components, it is possible to save development time and stop adding more complex operators, if the algorithm is performing efficiently enough. Details about this development methodology are given in the next Chapter.

As mentioned in the above definition, the word "metaheuristic" typically refers to both the algorithmic framework and the specific realisation of an algorithm. This way, "ant colony optimisation" can be both the metaheuristic framework, as well as specific algorithm implementation to a specific problem, such as "ant colony optimisation for solving the travelling salesman problem". To resolve this ambiguity, it was proposed that the term "metaheuristic framework" is used for the general metaheuristic, while the term *metaheuristic algorithm* denotes an implementation of some metaheuristic framework to solve a specific optimisation problem [96].

3.2 Defining characteristics of metaheuristics

Based on the definitions of metaheuristics discussed at the beginning of this Chapter, there are several identifying characteristics that all metaheuristics have [150, 162, 365]:

1. Metaheuristics are general optimisation frameworks, applicable to a wide variety of problems,
2. Metaheuristics place very little requirements on problems to be able to solve them,
3. Metaheuristics have a local improvement operator that is able to find a local optimum around a point,
4. Metaheuristics have procedures to avoid being stuck in local optima for too long,
5. Metaheuristics have a strategy to balance local improvement and local optima avoidance [150, 162, 365].

Along with the characteristics that are provided in their definition, it is agreed that:

6. They are stochastic algorithms that rely on random value generators [150, 343, 368],

7. They scale well [150, 162, 365, 368],
8. They provide no guarantee on the optimality of results [162, 365].

3.3 Types of metaheuristics

During the last decades, numerous different metaheuristics have been developed. While they all share the same goal of finding "as high quality as possible" solutions, and doing this task "as quickly as possible", the internal mechanisms they use in an attempt to realise this goal vary significantly. Some metaheuristics, such as the genetic algorithm and ant colony optimisation have clear inspiration in natural processes. Others do not have any metaphor in their basis, and instead, use reasonable strategies to overcome issues with the underlying simpler methods. Despite the great variety, metaheuristics can be classified into several categories [96, 150, 162, 365].

Depending on the number of solutions they consider in each iteration, metaheuristics can be a single state or the population-based. Based on the prerequisites, the components of metaheuristics can be constructive or modification components. They can use the search history (memory), or be memoryless. Finally, there is a great variety of techniques for handling constraints [96, 162, 343, 365]. These categories mostly overlap and more complex implementations might be difficult to categorise.

3.3.1 Single-state and population metaheuristics

Single-state techniques, also called *trajectory methods* always keep a single solution in each iteration of the metaheuristic. They find good solutions by doing systematic modifications to the current solution and checking if each change achieved an improvement. The sequence of modifications to the currently active solution can be visualised as a multidimensional point travelling through the solution space. For this reason, they are sometimes called trajectory methods. The best solution so far is always kept, therefore any improvement is preserved while the quality of the current solution can oscillate [162, 343, 365]. Popular single state methods are GRASP, iterated local search, variable neighbourhood search, tabu search and simulated annealing [119, 127, 129, 133, 135, 136, 137, 369, 370].

Conversely, *population methods* are capable of keeping more than one solution at each iteration. They can be visualised as multiple points whose positions are jumping around the solution space. Having more than one point means that it is possible to cover wider areas of the solution space in one iteration, and by such sampling get more information about the problem and high-quality solutions. Drawbacks of such approach are higher memory requirements and more processing resources needed for each iteration [162, 343, 365]. Widely used population methods include ant colony optimisation and genetic algorithm [116, 124, 150, 343, 351, 371, 372, 373].

3.3.2 Search history

Search history, also called *search memory* is the capability of a metaheuristic to remember solutions from previous iterations and use the information gathered during the search to achieve better decisions. Search history is also sometimes called *search memory*, and it can be further classified into *short-term memory*, and *long-term memory*. Short term memory keeps a record of the most recently visited solutions or moves that the metaheuristic has performed. Long term memory can contain information like statistics about components that were parts of good solutions etc. Using such long term information, the metaheuristic can behave more reactively to the current state of the search process [150, 162, 343, 365].

Depending on their use of the search history, metaheuristics can be classified into *memoryless* techniques, that do not use search history, and those that incorporate *memory*. The techniques that do not use memory keep track only of the current solution (or population), like in a pure *Markov process*, in which the next state depends exclusively on the current state. At the very least, in addition to the current state, all reasonable metaheuristic implementations keep a copy of the best-so-far solution [150, 162, 365, 368, 374]. Using search history is an attractive research area with high potential to improve metaheuristic performance [127, 369, 375, 376, 377]. However, identifying the best strategy to use memory can be demanding and require a lot of experimentation.

A notable area in which the search history is intensely used are *adaptive metaheuristics*, sometimes also called *reactive search*. With conventional metaheuristics, the implementation decisions are done by the human developer before the metaheuristic is run, and do not change, regardless of the search progress on the current problem. Unlike the conventional metaheuristics, adaptive metaheuristics do not have such restriction and use various adjustments during the optimisation algorithm run. They react to the current state of the optimisation process, and can change the behaviour of the algorithm in an attempt to improve efficiency. This can help them self-adapt to various different inputs problems [378, 379, 380].

3.3.3 Constructive and perturbation-based metaheuristics

Each metaheuristic is based on several types of *operators* and *procedures*. They start with an initial solution or a population of initial solutions that need to be initialised in some way, therefore all types of metaheuristics must at the very least have a solution initialisation procedure. The simplest way to achieve this is using random construction, by initialising each variable to some random value from its domain. Advanced versions might use a greedy randomised algorithm, include feasibility restrictions, or include a variety of problem-specific speedups

[127, 135, 150, 162].

The operators of a metaheuristic are typically classified into *constructive* and *modification* operators. *Constructive operators* start with an uninitialised solution where no values are assigned to any of the variables. Constructive operators assign suitable values to these uninitialised variables. Typically, a constructive procedure can also take an incompletely initialised solution and complete it into a fully specified solution. Solution initialisation procedures are constructive in regular implementations of metaheuristics . Conversely, *modification operators* perform modifications of existing solutions and they assume that a completely specified solution will be provided as an input [127, 150, 162, 351]. Using an incompletely initialised solution with such operators results in an exception during runtime.

Constructive operators include random initial solution initialisation procedures, ant agents in the ant colony optimisation metaheuristics and the greedy algorithm. A common modification operator is a *local search* procedure, that searches for a local optimum in the proximity of an initial point. Note that unlike constructive procedures, as explained in the previous paragraph, local search has a solution as an input. Likewise, various *perturbation operators* are used to introduce random changes in a solution [127, 150, 351].

Based on this classification of the operators they use, metaheuristics can be classified into *constructive* and *perturbation-based* metaheuristics. In *constructive* metaheuristics, a current solution in each iteration is initialised by a constructive operator. Ant colony optimisation and GRASP metaheuristics use this approach [124, 135, 351]. *Perturbation-based* metaheuristics create an initial solution or a population of solutions that is then modified using the modification operators. Examples include iterated local search, variable neighbourhood search, tabu search and simulated annealing [127, 137, 369, 370]. In population-based metaheuristics, there exist more complex operators that do not take only one solution to modify as the argument. The crossover operator, used in e.g. genetic algorithm is a notable exception. It takes two (or more) solutions as an input and as a result, it produces one or more solutions by mutually exchanging values of the solution variables that were provided to the operator [116, 373].

3.3.4 Constrained problems and feasibility

Solving constrained problems, as defined in Sections 2.3 and 2.4.1 adds complexity to solving techniques. With constrained problems, in addition to investigating the areas of the objective function, any algorithm for optimisation must also take care of satisfying the constraints. This can be a highly complex task [295, 381, 382, 383]. Regardless of the used technique, there exist three general ways to add constraint handling using optimisation algorithms:

1. Prohibiting infeasible solutions,
2. Using repair procedures for infeasible solutions,
3. Allowing infeasible solutions. [162, 295, 382, 384, 385, 386, 387, 388, 389, 390, 391]

The first and the last option are two independent categories, while the second option is a middle-ground approach that can also be used as an addition to the other two. In an algorithm that prohibits infeasible solution, a repair might be attempted in cases when the current operator produced an infeasible solution. Likewise, a problem specific repair procedure can be used when infeasibility is not prohibited, to help the metaheuristic with known problem-specific speedups [387, 388, 389, 390].

Note that while these techniques are widely used, the list above is not exhaustive as there exist ways to handle constraints that do not fit into any of these three general ways [162, 387, 390, 391]. For example, multi-stage solving is sometimes used in timetabling, and consists of dividing the problem into two different problems, solved separately. In the first phase, the goal of the metaheuristic is to only find a feasible region, and then in the second phase, the full problem is solved using the complete objective function, and starts from the feasible point from the first phase [392]. Another example can be found in [393], where authors developed a completely new variant of the ant colony optimisation metaheuristic, especially suited for highly constrained problems. While authors report success of these techniques, they bring more complexity to the process of development.

Prohibiting infeasible solutions

The first technique—prohibiting infeasible solutions restricts the algorithm to search in the feasible region only. It might be a good option for simple constraints as it helps the metaheuristic with problem specific operators. However, in highly constrained problems, e.g. in scheduling, it might be difficult to find even a single feasible point [391, 394].

Drawbacks of this method include the fact that restricting the search to feasible options also requires that the operators must produce feasible results. This can be very difficult to achieve in highly constrained problems, as it requires both a careful formal analysis of the problem specifics, as well as a high level of support in the codebase to achieve feasibility in each step [343, 388, 390, 391, 395]. High complexity increases the required development and maintenance time and effort. Even after the algorithm is finished, a simple request to add a single new constraint might not fit in the constraint model that was specified in the earlier stages of the project and require costly revisions and rewrites in the codebase [388, 391, 395, 396]. Finally, even with highly detailed rules to ensure feasibility, since e.g. scheduling is $\mathcal{NP}-$ *hard*, there might be cases where the constraint handling system is unable to produce any new feasible points. When this happens, the algorithm ends up in a “dead-end” and all attempts to modify a solution without returning to a previous one results in an infeasible step. It is not clear what to do in such situations—possibilities include backtracking, repeating the operator with a higher degree of allowed changes, or restarting the search from a completely different point [162, 352, 397, 398].

Despite higher complexity, this technique has its advantages. Most important benefits include the guarantee to the users that any solution produced by the algorithm is always feasible, whenever the input problem allows it. Additionally, restricting the search only to a feasible parts of the solution space can speed up the metaheuristics. The potential speedup is due to the smaller space to cover as well as the fact that the metaheuristic does not have to invest additional effort to bring the points to feasible regions [388, 391, 395, 396].

Using repair procedures

Repair procedures, sometimes also called filtering procedures attempt to repair an infeasible solution and bring it to the feasible region. Typically they perform modifications based on problem specifics. For example, in [399], a repair operator is defined for the knapsack problem. The algorithm tries to repair infeasible problem instances by removing elements from the overloaded knapsack until the capacity constraint is met. While in this simple example a repair is always possible, doing similar repairs with solutions of highly constrained problems is more difficult and might not always be possible in a reasonable computing time. Discussion and some results related to the use of repair methods has been reported in [383, 387, 388, 390, 391, 400, 401, 402, 403].

Allowing infeasibility

When infeasible solutions are allowed, finding good solutions is achieved by introducing modifications into the objective function. Modified objective functions like that include a *penalty* component, so that each constraint violation is discouraged by a low-quality value. Not respecting constraints typically has higher penalty than having a bad objective function value, and multilevel lexicographic ordering objective functions are a good fit for modelling such problems [87, 385, 386, 404, 405, 406, 407, 408].

When infeasible solutions are allowed, the metaheuristic will need to discover how to find feasible regions by itself. This can require considerable computing resources and the algorithm might spend a lot of time being stuck in infeasible regions when problem being solved is highly constrained. Further, in cases of difficult problems, the users might occasionally be presented with an infeasible solution, which is not a good option for the users since such results are essentially useless in practical applications [386, 387, 409].

Advantages of this technique is the simplicity and low cost of development. Everything that is needed is to appropriately encode the constraints into the objective function. Simply specifying and coding those conditions in a programming language is typically much simpler than developing operators that guarantee that feasibility is always conserved [386, 396].

Explicit constraint handling in metaheuristics

Most metaheuristics are deferring the constraint handling to the developer, since the above-defined techniques are standard ways to implement them. Nevertheless, some metaheuristic frameworks handle the constraints in explicit ways. Most notably, the literature about the GRASP metaheuristic implies that the infeasible solutions are filtered during construction and further specifies a repair step after the solution construction in case the construction algorithm did not produce a feasible solution [135]. In a similar fashion, the simplest variants of the ant colony metaheuristic assumes that the construction procedure will avoid infeasible solution components [124, 351]. These are, of course, guidelines, not critical components of neither of those algorithms. In practical applications, constraints can be implemented differently, since a lot of problem-specific details always must be added to each metaheuristic. Both GRASP and ant colony optimisation are compatible with all three ways to add constraints [124, 135, 351]. For example, in [410], a hybrid GRASP metaheuristic is used where constraint violations are allowed in the solution construction algorithm. In fact, there is a large volume of research devoted to hybrid metaheuristics that do not follow any metaheuristic in its pure form and achieve good results [397, 411]. Instead, hybrid algorithms combine operators and general principles from several techniques.

3.4 Intensification and diversification framework

In the previous paragraphs, it was described that metaheuristics use a variety of different types of operators. In the detailed descriptions of each metaheuristic in the rest of this Chapter, it will be further described how the basic ideas that inspired some of the popular metaheuristics come from various areas. Some metaheuristics are devised following logical ideas such as “do not return to already visited places in order to discover new better points”. Some metaheuristics simulate natural phenomena, including ants, metallurgical processes, evolution.

Despite differences, from the above discussion, it is also clear that they have important similarities. They are all algorithm development frameworks, with the goal to find the best possible solutions in as short time as possible. They can all share similar constraint handling techniques and search history. In their essence, they all perform strategic probing of the solution space in order to approach the optimum. This Section will present probably the biggest generalisation in the area of metaheuristics, the *intensification and diversification framework* [162].

Basic ideas and definitions

Intensification and diversification are two principles included in every metaheuristic, and each metaheuristic is unique in the way it achieves both. Each metaheuristic should be able to focus on the good areas of the search space and in them, systematically and intensely scan the proximity of top quality solutions. However, when no improvement is achieved in these regions for a while, it should continue and explore other areas with solutions that are significantly different from those explored before [162, 412, 413]. The first idea of detailed search in regions with good solutions is called intensification, while the second one, moving to previously unexplored areas is called diversification. These concepts were first mentioned in the investigation of the tabu search metaheuristic [412]. Common names for the related principles in evolutionary computation are *exploitation* and *exploration* [162, 414].

Intensification and diversification are effects of metaheuristic components. In the spectrum between pure intensification and pure diversification, there are numerous intermediate possibilities. In their well-known overview, Blum and Roli [162] introduce the following defining criteria for *diversification*: moves that are not guided by the objective function and moves guided by random decisions. Conversely, they define *intensification* as moves guided exclusively by the objective function.

Intensification and diversification spectrum

Most operators can be placed close to the pure categories, however even the operators traditionally viewed as diversification operators, usually have some elements of intensification and vice-versa. For example, a perturbation operator in the iterated local search metaheuristic is viewed as a diversifying operator. It is indeed an operator whose outcome is almost entirely diversification. Still, it is not pure diversification. Firstly, the design guidelines specify that changes need to be limited to sufficiently close points. Further, it can also be argued that even the very structure of the neighbourhoods with well-tuned operators always has a certain bias towards fostering good moves with the given objective function. Therefore even the random perturbation moves cannot be considered pure diversification, and have a slight intensification component, unless the neighbourhood is also randomly selected and has an unlimited distance [162].

Balance between intensification and diversification

The described ideas are tightly related to the definitions of “locality” and “distance”, which are always problem specific and difficult to formally define. Likewise, it is always difficult to say how long should a search focus on the good region before moving on to more distant ones. Finding the answer to this question, and more broadly, finding the right and efficient balance between intensification and diversification is a crucial issue in metaheuristics [415], and the design goal for any successful implementation of a metaheuristic technique [413, 416, 417].

If the intensification is insufficient, then the metaheuristic will not spend enough time in careful examinations of the neighbourhoods of known good solutions. This way it will not be able to catch the best solutions that lie in the neighbourhood and move on to the next region too soon. Conversely, staying in the neighbourhood of a good solution can waste a lot of time if a highly detailed search of the neighbourhood is not producing improvements or the improvement is small. By focusing too much on the known regions with mediocre solutions, the algorithm will not have sufficient time to reach the areas with best solutions.

Finding this delicate balance between intensification and diversification is the ultimate goal in any project that uses metaheuristic techniques. Further, the process of implementing metaheuristics to achieve this desired goal in large deal still resembles more an art, guided by the developers' intuition rather than a rigorous engineering process based in science and formal processes. While a quick initial implementation of metaheuristics generally works, to achieve state-of-the-art performance, detailed experimentation with different algorithm components and their parameters is necessary. Despite progress in the tuning algorithms, this process typically requires a large number of computationally demanding experiments [8, 413, 416].

3.5 Proto-metaheuristics

There exist solving techniques that satisfy some of these conditions that have been in use even before the term "metaheuristic" was proposed, however they are not metaheuristics in the modern sense. These techniques, such as greedy search, hill-climbing, and similar [69, 98] are developed out of some general rules of thumb, and are too simple to be called metaheuristics. Still, all of them satisfy almost all the requirements of a metaheuristic, most importantly, their functional requirements (1) and (2) from Section 3.2, and complete metaheuristics can be developed by only adding a single simple operator to them. These techniques, especially the local search are used as components of almost all metaheuristics. Therefore, four such techniques, for which the term "proto-metaheuristic" is proposed in this thesis are described in this Section.

3.5.1 Pure random search

From the general definition of an optimisation problem, stated in Section 2.3, it is clear that a solution of any optimisation problems in the most broad sense is a problem of deciding on the values of *variables* for which the value of the objective function will be as low as possible. These variables are from some problem-specific domain and are also subject to *constraints*. Given this definition, and a random value generator, it is trivial to implement the simplest problem-solving algorithm: *pure random search*, sometimes also called *Monte Carlo method*, *crude random search* [418] etc. Random search consists of assigning each variable a random value from that variable's domain, usually following a uniform distribution. This process is equivalent to selecting a random point in the solution space. Pure random search repeats this procedure of random sampling while keeping the best solution so far. The technique's earliest mentions can be traced back to Anderson, Brooks and Karnopp [93, 94, 231, 419, 420, 421].

This technique surprisingly, satisfies all of the above requirements except (3) and (5) from Section 3.2. It can even be proven that the technique converges to optimality in probability [368]. This does not claim anything related to the time requirements to find an optimum which might be immense. Still, this result is remarkable, and shows that simply maintaining the best solution so far can be the differentiator between converging in probability or not. In [374], it is shown that not even the highly sophisticated way of creating solutions used by a canonical genetic algorithm is enough to ensure that the final population will contain a globally optimal solution, in fact, it was proven that it never will converge in probability. However, when a standard practice of maintaining a copy of the best solution so far is used, the GA converges in probability.

If the search space consists of large regions of good solutions, then indeed the probability that a random selection of variables will also provide good result is not negligible. For such problem instances, this technique might indeed work in practice. Naturally, this is not the case for any realistic problem studied in this thesis. In fact, the opposite is true since difficult combinatorial optimisation problems can have multiple isolated local optima. When solving constrained problems, in large parts of the solution space not even the constraints are satisfied. Pure random search is not a good choice to solve such problems [231, 368].

Despite the fact that is not an effective search algorithm, developing a pure random search can be a simple and useful first step in any optimisation project. Creating and analysing random solutions gives useful insights on the problem features. Moreover, random search is commonly used in more complex metaheuristics, to select initial points or populations [127, 150, 230, 422, 423].

3.5.2 Greedy algorithm

A *greedy algorithm* is an algorithm that chooses a locally optimal decision in each step. Similar to the random search, it is a solution construction procedure that produces a completely new solution in each iteration. It is a refinement of the random search process, that does not set the variables to completely random values—instead it assumes that choosing what currently seems the best might be close to the optimal solution to the problem while gradually assembling a complete solution, step by step. They do not find the global optimum for all problems, however, for some, it can be proven that they indeed always lead to a global optimum [69, 153, 424]. Notable example is the famous Dijkstra's algorithm for finding the shortest path between two nodes in a weighted graph [68, 69]. Greedy principle satisfies all the properties of metaheuristics stated above, except (4), (5) and (6).

Greedy algorithms are solution construction procedures that iteratively set values of the variables in the optimisation problem instance. They assume that it is possible to approximate the contribution that setting a variable in an incomplete solution will have to the overall objective function, and that it is possible to filter out values that will lead to infeasible solutions. Let us define the function φ, as an objective function that evaluates such incompletely constructed solutions, where part of the variables is still not set. More precisely, let $\varphi(x,x_i,v)$ to be the *incremental cost function*, in this work also called *partial objective function*. This function calculates the approximate change in the objective function of the incomplete solution $\mathbf{x}$, when the previously unset variable $x_i \in \mathbf{x}$ is set to the value v. Further, let us use $feasible(\mathbf{x},x_i)$ to denote a procedure that will provide all possible values of the variable x_i in the incompletely initialised solution $\mathbf{x}$ for which the constraints are not violated. In the most general sense, the algorithm would set each variable sequentially, and estimate the overall contribution to the objective function of each possible value, as outlined in the pseudocode in Algorithm 1. Using this approach, each variable would be set to the value that brings the least estimated increase in the objective function value.

Note that to be compatible with the greedy algorithm, the $feasible(\mathbf{x},x_i)$ function must return a finite number of elements. This requirement implies that greedy algorithms are not applicable directly for continuous problems. To allow applications to continuous problems, some type of discretisation or a suitable sampling strategy would be required. An example of such technique is developed as a part of the continuous GRASP metaheuristic [424, 425], where a line search across each variable is performed in the construction process.

Algorithm 1 Greedy algorithm pseudocode

```
procedure GREEDY()
    $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, for all $i \in [1,n]$, $x_i$ = null        ▷ Uninitialised solution
    for all $i \in [1,n]$ do
        $v_i^*$ = null        ▷ Uninitialised localy optimal value of the variable $x_i$
        $p_i^* = \infty$        ▷ Best partial evaluation
        for all $v_{ij} \in feasible(\mathbf{x},x_i)$ do
            $p_{v_{ij}} = \varphi(\mathbf{x},x_i,v_{ij})$
            if $p_{v_{ij}} < p_i^*$ then
                $p_i^* = p_{v_{ij}}$
                $v_i^* = v_{ij}$
            end if
        end for
        $x_i = v_i^*$
    end for
    return x
end procedure
```

As a simple example, when solving the knapsack problem, defined in Section 2.7, we could always select the item with the highest profit-to-weight ratio that does not exceed the current capacity [249]. In a slightly more complex example, we could develop a greedy algorithm for the travelling salesman problem (TSP) (Section 2.7). A true random solution would mean that starting from the initial city, the salesman proceeds to a random city. Results constructed with this procedure would almost always break the constraints: there is no guarantee that a random solution would be a cycle and that it would not return to the initial city too early or have repeated visits to the same city. A slight refinement by adding the constraints to the algorithm would ensure that random solutions are at least feasible, and that starting from the initial city, a random city is selected without repeated visits and closing the cycle too soon. This algorithm would not be a random search in the strict sense, and the solution quality would still be low. A further refinement by choosing the closest instead of random city at each step [424] would transform the algorithm into a greedy algorithm for the travelling salesman problem:

- *In the initial city, choose the closest city.*
- *In each of next cities, choose closest city that has not yet been visited.*
- *When all cities have been visited, return to the initial city.*

Similarly to the random search, greedy algorithm can be used as a component of other metaheuristics. It is a popular choice for solution initialisation method [127, 351]. However it should be noted that for e.g. the asymmetrical travelling salesman problem, it is possible to create problem instances for which the greedy strategy is a poor choice, even worse than random search [426]. Therefore, some initial testing on a representative set of problems is always advisable, when formal or empirical performance results are not available (e.g. for new and highly specific problems).

3.5.3 Regret avoidance

The principle of *regret avoidance* is the idea of constructing the solution in a way that will minimise the regret in each step of the solution construction. Regret is viewed as the possible loss from not choosing a solution element. Similar to greedy algorithms, this method can be used to initialise solutions that will be further improved using other techniques, and it assumes the existence of the Φ function that can evaluate incomplete solutions. Given such a function, *regret* can be formally defined as the difference in the contribution of the best and the second-best outcome.

The principle was first considered by Leonard Savage in 1951 [96, 99, 427], and examples in the literature include the Vogel's approximation method for the transportation problem, and vehicle routing [428]. It has all the characteristics that are shared by metaheuristics from our list, except (4), (5) and (6). While it is a rather general framework to work with optimisation problems, there is still more complexity when applying regret avoidance than greedy algorithm or random search. Therefore, with regard to (1), it can be argued that this approach is slightly less general.

Let us consider a vehicle routing problem (VRP) instance. A regret avoidance heuristic is building a solution by going through a list of customers and assigning a vehicle for the current customer. For each unassigned customer c_i, let us assume that the algorithm first calculates the best insertion place in the current routes of all vehicles v_i. For some customers, the cost of assigning the customer to vehicle v_i might be approximately equal, regardless of the selected vehicle. For other customers, there might be considerable variation. A regret for each customer $r(c_i)$is defined as the difference between the second-best (v^{*2}) and the best (v^{*1}) vehicle choice:

$$r(c_i) = estimateContribution(v^{*2}) - estimateContribution(v^{*1}). \tag{3.1}$$

A large regret means that there is a significant difference between the best vehicle and others and that there is not much good alternatives for the insertion of the current customer. Low regret means that it is possible to assign the customer into multiple different vehicles without great decrease in quality. The regret avoidance heuristic assigns the customer with the highest regret to the best vehicle. Indeed, it can be argued that it is reasonable to assign this customer first since there are very limited options for assigning it, while for other customers there is still more opportunities to find equally good vehicles [427, 428].

3.5.4 Local search

Local search is based on the idea that a sequence of small changes to the solution can produce a considerable improvement. In the basic form, it can be summarised as the principle of "keep doing small improvements, as long as you're improving the solution". Unlike previously described proto-metaheuristics, local search is not a constructive approach. Instead, it assumes that it starts with a fully specified initial point, where all variables are assigned to values from the appropriate domain. From there, local search modifies the components of the solution in an attempt to improve it. It cannot construct new solutions starting from nothing (an uninitialised or unspecified solution) [69, 98, 100].

In the list of the characteristics of metaheuristics, local search satisfies all conditions except (4) and (5). Local search is based on definitions of the problem (objective function and constraints), as defined in Section 2.3, and a definition of a *neighbourhood* on the problem solution space (Section 2.3.1). Each local search procedure can only guarantee that the result will be at best a *local optimum* [69, 87, 98, 100]. As described in Section 2.3.1, there is no guarantee that a local optimum is also globally optimal. In difficult combinatorial problems, unless the choice of the initial point was incredibly lucky, results of local search will not be globally optimal [69, 87, 100].

In the general optimisation problem, given an initial point $\mathbf{x}$, the local search performs the search of the neighbourhood $\mathcal{N}(\mathbf{x})$ around the initial point, using some search strategy. The search strategy could be "*try incrementing each variable $x_i \in \boldsymbol{x}$ to the next feasible value*".

The pseudocode, provided in Algorithm 2 consists of a loop, where the search procedure that looks for improvements in the neighbourhood is denoted *searchNeighbourhood*, and $f(\mathbf{x})$ is used for the objective function. If an improvement has been found, the procedure repeats in the new neighbourhood, around the previously found improved point. It stops when the attempt to improve a solution by searching the neighbourhood no longer improves the result [69, 98, 100, 429].

Algorithm 2 Local search pseudocode

```
procedure LOCAL SEARCH($\mathbf{x}$)                    ▷ $\mathbf{x}$ is the initial solution
    $\mathbf{x}_{best} = \mathbf{x}$                    ▷ $\mathbf{x}_{best}$ is the best solution found so far
    repeat
        $\mathbf{x}^* = searchNeighbourhood(\mathcal{N}(\mathbf{x}))$
        if $f(\mathbf{x}^*) < f(\mathbf{x}_{best})$ then
            $\mathbf{x}_{best} = x^*$
        end if
    until $\mathbf{x}_{best}$ did not improve
return $\mathbf{x}_{best}$
end procedure
```

The definition of the neighbourhood and the choice of the search strategy inside the neighbourhood specify the local search operator. The *searchNeighbourhood* procedure can use the *first improvement* strategy, where the search of the neighbourhood $\mathcal{N}(\mathbf{x})$ stops as soon as the first improvement is found. Another version of the search strategy could be the *best improvement* strategy, in which the entire neighbourhood is enumerated and evaluated by the objective function, and the best point in the neighbourhood is returned. Local search should be tuned to achieve a good balance of speed and thoroughness. Narrow neighbourhoods and the first improvement strategy will work quickly, however, they might ignore large parts of the neighbourhoods, including potentially great solutions. Conversely, a decision to increase the size of the local search will find better results, however it might take much more time [69, 98, 100].

For the travelling salesman problem, there are numerous local search techniques that have been developed, and the earliest is the *2-opt*, published in 1950s by Croes [102]. This algorithm defines the neighbourhood around a current solution to the TSP as a set of all solutions that can be reached by swapping two connections in a route, e.g. modifying the sequence $A-C-B-D$ to $A-B-C-D$. Generalisations of this technique exist: the *3-opt* algorithm that first deletes and then reassigns three components of the current route, and the *Lin-Kerningham* heuristic, which does this for an arbitrary number of exchanges [430, 431].

Local search is widely used as the operator in other metaheuristics. It allows achieving the characteristic (3) in an easy way. Even if the local search is not specified in the basic version of some metaheuristic, experimenting with adding it is always useful as it can improve the performance [127, 150, 351]. Such is the example of the genetic algorithm, that in the canonical version does not have a local search operator, and it's extension, memetic algorithm [1] [373, 432].

3.6 Established metaheuristic methods

Unlike the simpler techniques that are described in the previous Section, metaheuristics are complete frameworks that are characterised by having both the ability to identify local optima (3), as well as to escape local optima if the search is stuck (4) [150, 234]. Some reasonable combination of characteristics (3) and (4) produces (5). Indeed, it can be observed that all of the discussed proto-metaheuristics lack (5) and either (3) or (4).

This conclusion can be utilised to quickly assemble simple metaheuristics from individual algorithmic components. For example, a developer might try using random search to initialise initial points, combine it with local search to improve the initial point, and repeat. While very simple, this procedure is in fact a complete metaheuristic satisfying all the required properties, and it is called *random restart local search*. More sophisticated ways exist to achieve condition (5) and balance local optima avoidance with keeping the local search deep enough to capture the good solutions.

Several different ways to achieve this were developed. A great number of different methods were proposed, especially during the "metaphor controversy" period. In this thesis, only the established methods will be analysed further. While deciding what is "established" always must be somewhat subjective, two criteria were used: (*i*) their historical significance and introduction of new ideas, (*ii*) the diversity criterion, to cover wide range of different frameworks [96].

3.6.1 Random restart local search

The *random restart local search* is one of the simplest metaheuristics. It is based on the idea of combining random search and local search, as mentioned above. Random point selection is used to provide an initial point in the solution space and the local search procedure then tries to improve that initial point. The described steps are repeated until the termination criteria (e.g. number of iterations or a timeout) is met [8, 127, 343]. The pseudocode of the metaheuristic is provided in Algorithm 3. Unlike any of the aforementioned proto-metaheuristics, it satisfies the criterion (3) as well as criterion (4). The criterion (3) is satisfied by using the local search which finds local optima, and the criterion (4) is achieved by starting from a new initial point at each iteration. In realistic combinatorial problems, the solution space is typically so large that the probability of ending up stuck in a local optimum is negligible [127].

[1] In this text, the original names for genetic algorithm and memetic algorithm are kept. It should be noted, however that in modern language, both are better described as metaheuristic frameworks than algorithms.

Algorithm 3 Random restart local search

```
procedure RANDOM RESTART LOCAL SEARCH( )
    $\mathbf{x} = selectRandomPoint()$                ▷ $\mathbf{x}$ is the initial solution
    $\mathbf{x}_{best} = \mathbf{x}$                  ▷ $\mathbf{x}_{best}$ is the best solution found so far
    repeat
        $\mathbf{x}^* = localSearch(\mathbf{x})$
        if $f(\mathbf{x}^*) < f(\mathbf{x}_{best})$ then
            $\mathbf{x}_{best} = \mathbf{x}^*$
        end if
        $\mathbf{x} = selectRandomPoint()$
    until end condition is met return $\mathbf{x}_\mathbf{b}$est
end procedure
```

Assuming that the quality of an average local optimum is sufficient for practical use, even this simple algorithm can be good enough. Unfortunately, case studies in properties of local optima well as the author's practitioner experience indicate that in large-scale combinatorial optimisation problems, local search around random points produces globally suboptimal results, unless the selection of the random points was incredibly lucky [433]. In the travelling salesman problem case study [434]and for graph partitioning problem [435], it is shown that the results of local search algorithms have a mean quality $\overline{y^*}$ that is a fixed percentage worse than the optimal point, and that the distribution of the local optima quality becomes peaked around that mean as the problem size increases. This indicates that the probability that random restart local search will find solutions better than the average local optimum $\overline{y^*}$ decreases as the problem becomes bigger. In practice, the search space of local optima is large, and there is typically too much mediocre local optima to rely on random restart to find the best ones [127].

While it can be argued that the algorithm has a way to control exploration versus local improvement, called criterion (5) in this work, it is rudimentary. The intensity of the local search can be configured by selecting the breadth of the search procedures. The local search, however, starts in a new, completely random point at each iteration without any option to control how far from the known points will the algorithm be going. The aforementioned discussion indicates that simply restarting the local search in random points is not flexible enough to ensure high-quality results in typical large-scale combinatorial optimisation problems [127]. During decades, several more complex metaheuristics have been devised that attempt to provide a better balance between staying around local optima and venturing into unknown regions of the search space. As will be shown in the following paragraphs, it is precisely the mechanism that provides this balance that is the key identifying characteristic of all metaheuristics [150, 343].

3.6.2 GRASP

The *greedy randomised adaptive search procedure*, usually referred to in short as *GRASP* [135] can be seen as a refinement of the random restart local search metaheuristic. It was proposed by Feo and Resende in 1989 [133] who applied it for solving the set covering problem. Similarly to the random restart local search, the algorithm consists of a construction and improvement phase. Using GRASP, however, each initial point is not completely random. Instead, it is constructed using an appropriate randomized greedy algorithm. Provided that the greedy algorithm works well on the problem and produces better solutions than random search space sampling, the local search is applied on solutions that are already showing promise. Based on this idea, it is conjectured that the algorithm can perform better than simply starting from a random solution each time [133, 135, 343].

Algorithm 4 GRASP pseudocode

```
procedure GRASP( )
    $\mathbf{x} = greedyRandomisedConstruction()$                ▷ $\mathbf{x}$ is the initial solution
    $\mathbf{x_{best}} = \mathbf{x}$                                ▷ $\mathbf{x}_{best}$ is the best solution found so far
    repeat
        $\mathbf{x}^* = localSearch(\mathbf{x})$
        if $\mathbf{x}^*$ is not feasible then
            $repair(\mathbf{x}^*)$
        end if
        if $f(\mathbf{x}^*) < f(\mathbf{x}_{best})$ then
            $\mathbf{x}_{best} = \mathbf{x}^*$
        end if
        $\mathbf{x} = greedyRandomisedConstruction()$
    until end condition is met
    return $\mathbf{x}_{best}$
end procedure
```

From the metaheuristic pseudocode in 4, it is apparent that the algorithm is highly similar to the random restart local search, with two differences:

- inputs to the local search operator are constructed using the greedy randomised solution construction procedure, and
- for constrained problems, initial points are checked for feasibility, and in cases when the initial solution is not feasible, problem-specific repairs that attempt to bring them back into the feasible region are performed. Examples of such repair procedures can be found in [403, 436, 437] .

The *greedy randomised solution construction* is a randomised variant of the greedy algorithm described in Section 3.5.2. This algorithm first constructs a restricted candidate list *RCL* of best options. The *RCL* can contain the best p options (cardinality based limit) or the options with the quality above some threshold (value-based limit). Then, a random component from the

RCL is selected. By repeating this process component by component, the algorithm constructs a complete candidate solution [135].

Using the value-based limit with a threshold that is relative to the currently available best and worst option, it is possible to control the algorithm in a very intuitive way. This value threshold denoted α is always in the interval $[0,1]$ and specifies that the *RCL* will contain all elements whose quality is in the interval $[c_{min}, c_{min} + \alpha(c_{max} - c_{min})]$, where c_{min} and c_{max} are the partial evaluations of the components with the best and worst quality. Specifying $\alpha = 0.2$ means that in each step, the algorithm chooses from a subset of components whose quality is in the best 20%. Setting α to zero converts the algorithm into a pure greedy algorithm which always selects the best option. Setting it to 1 turns it into random selection [135].

Compared to the random restart local search, GRASP provides users a richer set of configuration options—in GRASP both the solution construction and local search behaviour can be controlled. The *RCL* size parametrization gives a way to choose the tendency to restrict the search around good solution components or allow the algorithm to venture into less promising directions. Balancing this with the local search intensity allows finding a good balance between focusing around good solutions and exploring wider areas of the search space [133, 135].

3.6.3 Iterated local search

Iterated local search, abbreviated ILS is another example of a very simple metaheuristic based on the idea of repeated use of the local search. It was first published by Baxter in 1981, who used it for solving the optimal depot location problem [128]. This idea was independently discovered by several other authors who used it under different names, such as *iterated descent* [438, 439], *large-step Markov chains* [440], *iterated Lin-Kernighan* [441], and *chained local optimisation* [442].

The pseudocode is given in the Algorithm 5. Along with local search, ILS uses a *perturbation operator*, to find suitable initial points for the next iterations of local search. The perturbation operator introduces changes to the previous point, usually by performing random modifications in the solution elements. Ideally, the perturbation operator would always modify the current point into a good initial point for the next local search iteration. The change should be sufficiently big to prevent the local search from returning to the previous point again the algorithm will be stuck in a local optimum for too long. However, the perturbation intensity also should not be too large to degrade the movements to random restart local search [127].

Iterated local search is a single-state method, since during the entire run of the metaheuristic, it is considering only one solution as a base for perturbation and next iterations of the local search. The decision whether the new local optimum $\mathbf{x}'^{*}$ is accepted as the current solution for the next iteration is done in the *accept* procedure. As seen in Algorithm 5, the decision where to move can be done based on the current and previous local optimum, and the history of

Algorithm 5 Iterated local search pseudocode

```
procedure ITERATED LOCAL SEARCH ( )
    $\mathbf{x}_{initial}$ = initialSolution()            ▷ $\mathbf{x}$ is the initial solution
    $\mathbf{x}^* = localSearch(\mathbf{x}_{initial})$            ▷ $\mathbf{x}^*$ is the current solution
    $\mathbf{x}_{best} = \mathbf{x}^*$            ▷ $\mathbf{x}_{best}$ is the best solution found so far
    repeat
        $\mathbf{x}' = perturb(\mathbf{x}^*)$
        $\mathbf{x}'^* = localSearch(\mathbf{x}')$
        if $f(\mathbf{x}'^*) < f(\mathbf{x}_{best})$ then
            $\mathbf{x}_{best} = \mathbf{x}'^*$
        end if
        $\mathbf{x}^* = accept(\mathbf{x}^*, \mathbf{x}'^*, history)$
    until end condition is met
    return $\mathbf{x}_{best}$
end procedure
```

previously visited points.

A simple acceptance criterion, called *better* accepts the newly found local optimum only if it outperforms the previous best-found solution. This way, a very strong intensification is supported as the algorithm will never consider areas around points that do not bring performance improvement, even in cases when the current point is only slightly worse than the best so far. Conversely, the *random walk* acceptance criterion always selects the last local optimum as the current solution, regardless of its quality. This encourages much more exploratory behaviour as even the points that are much worse than the previous one always get accepted. Between those two criteria, there are numerous possibilities for various middle-ground selection criteria, e.g. mimicking the gradual shift from diversification to intensification as in the simulated annealing metaheuristic [127].

While the pseudocode for the procedure *accept* in the Algorithm 5 allows using history, the metaheuristic is frequently able to achieve great results without using history. The two mentioned acceptance criteria, "better" and "random walk" do not use history beyond the information about the current and the previous point. In practice, even these simple acceptance rules are frequently achieving very good results and given their simplicity, they are a very popular choice. Nevertheless, there are indications that using information about previous runs improves performance [375]. A simple use of history can be implemented as a restart in cases when no improvement is found after a while. There are numerous other ways to use information about previous points, and in cases when state-of-the-art performance is needed, researching sophisticated ways to use history can allow making better decisions and improve the quality of solutions [127].

Iterated local search is similar to both GRASP and random restart local search, in terms of iterating the runs of the local search multiple times until the end condition is met. The most important difference is the way these metaheuristics evade being stuck in local optima—GRASP and random restart local search create a completely new solution in each iteration while ILS instead performs modifications of the current solution using the perturbation operator. These modifications move the current point relatively close to the current local optimum. Such behaviour illustrates the assumption that good solutions tend to be grouped nearby in the search space. As opposed to the random restart local search, ILS tries to utilise this assumption by keeping reasonably close to known points, while still avoiding being stuck in the same local optimum [127]. The metaheuristic is also similar to other single-point techniques, such as tabu search, where a *random shakeup* was described in the early literature as an operator analogous to perturbation in the iterated local search [129].

3.6.4 Variable neighbourhood search

Variable neighbourhood search, abbreviated VNS is another single-state metaheuristic based on iterating the local search procedure. While in the previously described metaheuristics it is generally assumed that the local search has a predetermined and fixed neighbourhood definition, VNS has a distinct feature of using multiple different local search operators, and a strategic way to select the appropriate local search operator. The variable neighbourhood search was first proposed in 1997 by Hansen and Mladenović [136].

The overall technique is very similar to ILS, as outlined in the pseudocode in Algorithm 6, where local search is iteratively used in such a way that the initial point for a local search is a slightly modified previous local optimum. The difference is that VNS performs local search and perturbation in more than one neighbourhood. During the metaheuristic run, the neighbourhoods for LS and perturbation are subject to systematic change with the intention of balancing intensification and diversification. The rationale for implementing multiple different neighbourhoods is that a local minimum in one neighbourhood might not be a local minimum in another. Further, a global optimum is optimal in all possible neighbourhoods. Therefore, using more than one neighbourhood definition adds an extra level of robustness and promotes discovery of solutions that might not be accessible by moving only in one type of a neighbourhood [136, 137].

How to define the neighbourhoods for use with VNS? The commonly used and simple way of creating different operators is having a single proximity definition and neighbourhoods of different sizes, e.g. including up to p closest points to evaluate. This approach was applied for solving the travelling salesman problem in the original article by Mladenović and Hansen [136], where each neighbourhood $\mathcal{N}_p(\mathbf{x}, v)$ is defined as the set of solutions that can be created by modifying arcs that connect p closest cities to the city v in the current solution $\mathbf{x}$. Increasing p also allows larger change in the solution $\mathbf{x}$. By using various values of the parameter p, it is possible to create a set of different neighbourhoods.

Algorithm 6 Variable neighbourhood search pseudocode

```
procedure VARIABLE NEIGHBOURHOOD SEARCH( )
    $\mathbf{x}_{initial}$ = initialSolution()                 ▷ $\mathbf{x}_{initial}$ is the initial solution
    $\mathbf{x}_{best} = localSearch(\mathbf{x}_{initial}, \mathcal{N}_1)$      ▷ $\mathbf{x}_{best}$ is the best solution found so far
    $k = 1$                                  ▷ current neighbourhood index, $k \in [1, k_{max}]$
    repeat
        $\mathbf{x}' = shake(\mathbf{x}_{best}, \mathcal{N}_k)$
        $\mathbf{x}'^{*} = localSearch(\mathbf{x}', \mathcal{N}_k)$
        if $f(\mathbf{x}'^{*}) < f(\mathbf{x}_{best})$ then
            $\mathbf{x}_{best} = \mathbf{x}'^{*}$
            $k = 1$                          ▷ Reset to the first neighbourhood
        else if $k < k_{max}$ then
            k = k + 1          ▷ Move to next neighbourhood, unless already in the last
        end if
    until end condition is met
    return $\mathbf{x}_{best}$
end procedure
```

Using the strategy of switching neighbourhoods as defined in Algorithm 6 is especially suited for neighbourhoods sorted ascendingly by their size. When this condition is satisfied, the algorithm tries to improve by using the smallest neighbourhood, and the fastest local search. If that fails to improve the current solution, the search area is expanded, therefore achieving diversification. By resetting to the most narrow local search as soon as improvement is detected, the algorithm switches to intensification again.

Along with simply defining a single local search across different neighbourhoods, in more sophisticated implementations, using more than one distance definition is encouraged. If each neighbourhood is populated according to a different distance definition, more diversity across different neighbourhoods is promoted. This way, even if two different neighbourhoods have similar size, they can contain completely different solutions [136], and the search can move in a completely different way, depending on the currently selected neighbourhood. A good example of different definitions of a neighbourhood can be found in [443], where de Paula et. al. propose three neighbourhood definitions for the parallel machines scheduling problem. Elements of each neighbourhood are created by a different type of modifications to the current solution, and each neighbourhood definition implies a different structure of its elements. Proposed neighbourhood definitions include all solutions that can be created by (*i*) swapping the jobs on one machine, (*ii*) swapping jobs on different machines, and (*iii*) moving jobs from machine with the highest load to the machine with the lowest load. Each neighbourhood in general has a different size, depending on the number of machines and the current job assignment. More importantly, each neighbourhood of the first type contains a different set of solutions than the other two, since swaps across one machine can never result in solutions for which swapping across different machines is needed.

Similarly to the definition of the local search that is highly dependent on the currently used neighbourhood, the *shake*$(\mathbf{x}, \mathcal{N}_k)$ procedure introduces random moves from the current solution $\mathbf{x}$, and returns a random element from the current neighbourhood $\mathcal{N}_k$. While the author uses the name *shake*, the behaviour and motivation for having this procedure is very similar to the *perturbation* operator in the ILS. This method diversifies the search and helps escape local optima, nevertheless, it is different from the perturbation in ILS since it assumes that the neighbourhood to use will always provided as an input. The basic version of the VNS always uses the *best* acceptance criterion, in which the current local search is accepted as the incumbent solution only if it outperforms the best solution found so far. Therefore, in the pseudocode, $\mathbf{x}_{best}$ represents both the best solution for and the incumbent solution for the current algorithm iteration [137].

3.6.5 Tabu search

The *tabu search*, abbreviated as TS uses the idea of using *search history*, to help avoid previously visited local optima. It keeps a *tabu list* of previously visited solutions, and explicitly prohibits re-visiting all the solutions in the tabu list. All moves that would cause the metaheuristic to return to solutions in the tabu list are called *tabu moves* . The length of the tabu list is a parameter. This optimisation technique was first proposed in 1986 by Fred Glover [129].

The pseudocode of TS is provided in Algorithm 7. At the algorithm start, the initial solution is constructed, the tabu list is created and the initial solution is added to the tabu list. While it is traditionally called tabu *list*, it is in fact implemented as a first-in-first-out queue. Each time a new element is added to the tabu list, the number of current elements needs to be checked, and if adding a new element would exceed the maximum size, the oldest added element must be removed.

In the main loop, the algorithm runs an extended version of the local search, denoted *localSearch*$(\mathbf{x}^*, \mathit{tabuList})$ in Algorithm 7. This extended local search method, as outlined in Algorithm 8 chooses the best neighbourhood element that is not in the tabu list. Preventing returns to recently visited solutions guarantees that the algorithm will not be stuck in a local optimum. Further, it prevents the algorithm from cycling in the same sequence of local optima for all potential cycles with up to l elements. In the TS metaheuristic, the result of this local search procedure is then selected as the current solution, added to the tabu list and then the next loop iteration starts. In terms of the acceptance criteria, the basic version of tabu search uses acceptance criterion that corresponds to the *random walk* acceptance in the ILS [129, 369].

In practice, keeping copies of entire solutions in the tabu list, and performing comparisons of the current candidate with each element of the tabu list can be impractical due to large memory and processing time requirements. Therefore, implementations usually do not store copies of previous solutions. Instead, they keep the history of the most recent transformations on the solution, and prohibit undoing these transformations. Likewise, it is possible to develop strategies

Algorithm 7 Tabu search pseudocode

```
procedure TABU SEARCH( )
    $\mathbf{x}^*$ = initialSolution()            ▷ $\mathbf{x}^*$ is the initial and currently accepted solution
    $\mathbf{x}_{best} = \mathbf{x}^*$            ▷ $\mathbf{x}_{best}$ is the best solution found so far
    $tabuList = initialiseQueue(l)$               ▷ Initialise the tabu list with maximum size $l$
    $tabuList.add(\mathbf{x})$                    ▷ Add the initial solution to the tabu list
    repeat
        $\mathbf{x}^* = tabuLocalSearch(\mathbf{x}^*, \mathbf{x}_{best}, tabuList)$
        $tabuList.push(\mathbf{x}^*)$
        if $f(\mathbf{x}^*) < f(\mathbf{x}_{best})$ then
            $\mathbf{x}_{best} = \mathbf{x}^*$
        end if
    until end condition is met
    return $\mathbf{x}_{best}$
end procedure
```

Algorithm 8 Tabu-extended local search pseudocode

```
procedure LOCAL SEARCH($\mathbf{x}^*$, $\mathbf{x}_{best}$, $tabuList$)
    $\mathbf{x}_{localBest} = tabuList[0]$        ▷ Current local optimum
    for all $\mathbf{x} \in \mathcal{N}(\mathbf{x}^*)$ do
        if $f(\mathbf{x}) < f(\mathbf{x}_{localBest})$ then
            if [not $tabuList.contains(\mathbf{x})$] or [$aspiration(\mathbf{x}, \mathbf{x}_{best}, tabuList)$] then
                $\mathbf{x}_{localBest} = \mathbf{x}$
            end if
        end if
    end for
    return $\mathbf{x}_{localBest}$
end procedure
```

where entire classes of transformations are prohibited. An example of such strategy could be to not allow any type of modification to last l variables that were modified. This way of “locking” large sets of variables and similar types of adding wide ranges of moves to the tabu list can in some cases become too restrictive. To ensure that the tabu does not prevent the metaheuristic from exploring good areas, a technique called *aspiration* is used. A simple and commonly used aspiration criterion is allowing moves that are in the tabu list if they have better quality than the best solution found so far. More sophisticated techniques that allow tabu moves if they cannot form a cycle have also been proposed [291, 444]. In Algorithm 8, the aspiration criterion is implemented in the *aspiration* procedure, that can override the tabu in the acceptance condition.

The size of the tabu list is the most important parameter of the metaheuristic. Longer tabu list means that it is less likely that the search will return to previously visited solutions and this is a favourable condition since cycling is an undesirable state for search algorithms. Still, the size of the tabu list should be balanced with the speed requirement, since too long lists can be too slow and too restrictive [369].

The metaheuristic is similar to GRASP and ILS, however it does not use the perturbation operator, nor construction by greedy algorithm to initialise inputs for the next iterations of the local search. Instead it simply does the random walk and always accepts the current local optimum. Further, the way it avoids local optima is different. Unlike ILS and GRASP, tabu search can guarantee the algorithm will not be stuck in cycles of local optima. This is not the case in basic versions of GRASP and ILS. Still, in good implementations, cycling will be avoided by different mechanisms, and the developer has the option of specifically designing these operators to ensure that cycling is avoided if needed [369].

3.6.6 Simulated annealing

The *simulated annealing* metaheuristic is inspired by the physical process of annealing, used in metallurgy to reduce defects in materials. This heat treatment improves the ductility and reduces the brittleness of the material [445]. The process consists of heating the material to high temperatures, then slowly cooling it under controlled conditions in a heat bath. Heating first diffuses the atoms in a random arrangement in the liquid phase. The slow cooling that follows allows reaching an energy equilibrium at each temperature, and as the final product, crystallisation in a perfect lattice which corresponds to the minimum energy state [121, 446].

This remarkable process, studied in the area of statistical physics has drawn the attention of the researchers in the area of optimisation, due to evident analogies with combinatorial optimisation. The idea to simulate the process of annealing was grounded in the fact that the process that can find the minimum energy state for metal specimens with large numbers of atoms could be employed to find minimum cost solutions to combinatorial optimisation problems with very large number of possible solutions [121, 370, 446].

The first papers describing the simulated annealing metaheuristic can be dated to work of Kirkpatrick in 1983 [119] and Cerny in 1985 [447]. Simulated annealing, abbreviated as SA is a single-state metaheuristic in which the algorithm creates a sequence of solutions, while keeping the best-found solution. The distinct feature of SA is the possibility of accepting solutions that degrade quality. Contrary to intuition, this allows metaheuristic to evade being stuck in local optima for too long and helps find the global optimum. The pseudocode of simulated annealing is given in the Algorithm 9 [121, 343, 370, 446].

Algorithm 9 Simulated annealing pseudocode

```
 1: procedure SIMULATED ANNEALING ( )
 2:     x = initialSolution()                      ▷ x is the initial and currently accepted solution
 3:     $\mathbf{x}_{best} = \mathbf{x}^*$                                   ▷ $\mathbf{x}_{best}$ is the best solution found so far
 4:     $T = t_0$                      ▷ Initialise the temperature parameter to the initial value
 5:     repeat
 6:         $\mathbf{x}' = perturb(\mathbf{x})$
 7:         if $f(\mathbf{x}') < f(\mathbf{x}_{best}))$ then                          ▷ Update the best so far
 8:             $\mathbf{x}_{best} = \mathbf{x}'$
 9:         end if
10:         if $f(\mathbf{x}') < f(\mathbf{x})$ then                           ▷ Update the current solution
11:             $\mathbf{x} = \mathbf{x}'$
12:         else if $random() < \exp(-\frac{f(\mathbf{x}')-f(\mathbf{x})}{T})$ then
13:             $\mathbf{x} = \mathbf{x}'$                           ▷ Allow acceptance of worse solutions
14:         end if
15:         $T = updateTemperature(T)$
16:     until end condition is met
17:     return $\mathbf{x}_{best}$
18: end procedure
```

The metaheuristic sequentially creates solutions from the neighbourhood of the current solution, using the perturbation operator as the one used in ILS. It is then decided whether to accept this new solution or not. Algorithm lines 10-14 define the acceptance criterion used by the basic version of the metaheuristic, where in each case when the new solution outperforms previous, it is always accepted. If the new solution is not better than previous, it is still possible that it will be accepted, and the probability that this will happen is equal to

$$P(\text{accept worse solution}) = \exp\left(-\frac{f(\mathbf{x}') - f(\mathbf{x})}{T}\right). \tag{3.2}$$

The above stochastic acceptance is implemented using the random number generator, where in the pseudocode, it is assumed that $random()$ function returns a random number in the interval from zero to one. The probability that a worse solution will be accepted depends on the temperature and drops as the temperature decreases. The $updateTemperature(T)$ function is slowly decreasing the temperature at the end of each iteration. Therefore, the probability that the worse solutions will be accepted is initially very high, which leads to the exploration phases in the early stages of the algorithm, which accepts virtually all solutions and performs a random walk in the search space. As time passes and the temperature drops, it becomes less likely that the algorithm will move in the direction that does not improve the solution, leading to an intensification phase. As the metaheuristic is close to finishing, the probability of accepting worse solutions becomes approximately equal to zero. In this phase, the algorithm for the greatest part accepts only improvements and degenerates into a variant of the local search. The algorithm runs until the termination criterion is met. Note that the cooling schedule must be adapted to the termination criteria, to ensure that cooling is slow enough [121, 343, 370, 446].

The most important parameters of the algorithm include the initial temperature and the cooling schedule. Very commonly used cooling rule is the geometric temperature update, that corresponds to the exponential temperature decrease. Using this type of temperature update, the temperature in the next iteration T_{k+1} is calculated by multiplying with a parameter $T_{k+1} = aT_k$. There are numerous other cooling schedules, including nonmonotonic updates of the temperature, in which temperature sometimes can increase as well [162, 448, 449, 450, 451]. More advanced implementation decisions include selecting the neighbourhood definition, and the acceptance probability function. While the exponential function provided in Equation 3.2 is most commonly used, there are other examples in the literature [370, 452].

For the basic version of the metaheuristic, it has been formally proven that it converges if infinite time is provided [370]. Compared to the previously described metaheuristics, there are several distinct features of simulated annealing. Unlike most metaheuristics, it does not have an explicitly defined separate local search method. Instead, it achieves intensification by accepting only the improving moves as the temperature drops and the termination condition nears. It uses the perturbation operator present in ILS and VNS to diversify search. When it comes to acceptance criteria, initially it is equivalent to the *random-walk* acceptance criterion in ILS, that gradually changes to *best*. As opposed to tabu search, SA is memoryless, and in the basic version it does not use information about search history.

3.6.7 Ant colony optimisation

The *ant colony optimisation* metaheuristic is a popular technique inspired by natural processes. It mimics the food searching behaviour that was observed in the Argentine ant (*Iridomyrmex humilis*), during experiments by Simon Goss et. al. [453]. These experiments have shown that the ants use an indirect form communication called *stigmergy*, to allow self-organisation of large numbers of individual organisms [454]. Each ant has very limited sight and other senses, however, collectively, they show highly sophisticated behaviour. A notable example is their ability to find shortest paths from their nests to the location where food is placed. They use *pheromone trails* to mark the path they used to reach the food source, as well as their return-trip.

Due to the differences in path length, the first ants to return to nest will also be the ones who found the shortest path. In their movements, ants prefer following the highest concentrations of the pheromone trail deposited by previous ants, which will cause even higher concentrations on that trail, as more ants follow it. This autocatalytic positive feedback behaviour was modelled by biologists and served as an inspiration for the ant colony optimisation metaheuristic [453]. It was first proposed by Marco Dorigo in 1991 [123].

The basic version of the ant colony optimisation was developed for the travelling salesman problem, and traditionally, the descriptions of the metaheuristic rely on an example TSP application. While this can be confusing for the newcomers that wish to apply it to a different problem, in practice this is not limiting as almost any combinatorial optimisation problem can be represented as a shortest path problem. In more precise terms, ACO can be applied to any optimisation problem for which it is possible to develop an element-by-element solution construction procedure [124, 351].

Using a problem independent notation, in the framework of ACO construction procedure, each solution is observed as a set of solution components. During solution construction, the algorithm must first select the first component, then keep adding the following solution components compatible with previous ones, until a complete solution is produced. In terms of the optimisation problem, similarly to the greedy algorithm in Section 3.5.2, a solution is a vector of variables $\mathbf{x} = (x_1, \dots, x_n)$, where n is the dimensionality of the problem. Each variable x_i must be set to some value v from the corresponding variable domain, or a feasible subset of the domain if the given problem is constrained. Let us assume that there is k possible values that can be assigned to x_i, and that the algorithm needs to choose which of k possible elements $\{v_{i1}, v_{i2}, \dots, v_{ik}\}$ to assign. Then, a *solution component*, denoted c_{ij} as a decision to set the value of x_i to the value v_{ij}. Indeed, solving each optimisation problem is a series of decisions on which values to set to a certain variable. In terms of ACO, it is a series of choices of solution components, as if assembling a physical object from individual parts [124, 351]. Note that, similar to the discussion for the greedy algorithm, discrete variables are assumed in this discussion, therefore, the basic version of ACO is limited to discrete problems. Still, extensions for continuous problems have been proposed [455].

To each solution component c_{ij}, in the framework of the ACO metaheuristic, a *pheromone value*, denoted τ_{ij} is assigned. This value indicates the quality estimate of each component, as a result of the collective work of large number of *artificial ants*, that update the pheromone values in such a way that components of good solutions are assigned with higher pheromone values. Along with the pheromone value, ACO can also use the apriori available information about each component, e.g. distance from the current to the next city in the travelling salesman problem. In the ACO metaheuristic, this information needs to be stored in the form of *heuristic function*, denoted $\eta(c_{ij})$. Pheromone information is gathered during the metaheuristic runtime, while heuristic information is based on the information available in advance that keeps constant while algorithm is working [124, 351].

The pseudocode of the metaheuristic is given in Algorithm 10. Initially, the pheromone values are assigned to the initial pheromone value τ_0. In the basic variant of the metaheuristic called *ant system*, τ_0 is set to zero. The loop begins by runs of *artifical ants*, which are probabilistic solution construction procedures similar to the random greedy algorithm used in GRASP. In each iteration, m ants perform solution construction. Unlike the randomisation used in GRASP, ACO uses a different approach, where the randomised biased solution selection is based on the current pheromone concentrations and the heuristic function. In each step of the solution construction the function $feasible(\mathbf{x}, x_i)$ gives possible feasible choices for the variable x_i, given the incomplete solution $\mathbf{x}$ that is currently under construction. Put in the language of ACO, the $feasible(\mathbf{x}, x_i)$ function enumerates all feasible components at the current step. Given k possible choices for the solution component, the probability that the component c_{ij} will be selected is given by the formula for probabilistic selection. Most commonly used is the *random proportional rule* formula

$$P\left(c_{ij}|\mathbf{x}\right) = \frac{\tau_{ij}^{\alpha} \cdot \left[\eta\left(c_{ij}\right)\right]^{\beta}}{\sum_{c_{il} \in feasible(\mathbf{x}, x_i)} \tau_{il}^{\alpha} \cdot \left[\eta\left(c_{il}\right)\right]^{\beta}}, \forall c_{ij} \in feasible(\mathbf{x}, x_i), \tag{3.3}$$

where α and β are parameters. The above formula assigns selection probability proportionally to the product $\tau_{ij}^{\alpha} \cdot \left[\eta\left(c_{ij}\right)\right]^{\beta}$. The parameters α and β are used to control the relative importance of the pheromone as opposed to heuristic information. Setting α to zero means that only the heuristic information will be used, and the construction will be reduced to a random greedy algorithm. Conversely, using $\beta = 0$ will ignore heuristic information, this way leading ants exclusively with information gathered by the walks of previous ants in the current metaheuristic execution [351].

Algorithm 10 Ant colony optimisation pseudocode

```
procedure ANT COLONY OPTIMISATION ( )
    $\tau_{ij} = \tau_0$ for all $i$, $j$                          ▷ $\tau_0$ is a parameter
    $\mathbf{x}_{best}$ = null                                     ▷ Uninitialised $\mathbf{x}_{best}$.
    repeat
        $currentSolutions = constructAntSolutions(m)$
        $localSearch(currentSolutions)$                            ▷ Optional step
        $updatePheromones(currentSolutions, \boldsymbol{\tau})$
        $\mathbf{x}_{best} = findBest(currentSolutions, \mathbf{x}_{best})$
    until until end condition is met
    return $\mathbf{x}_{best}$
end procedure
```

Each constructed solution can be further improved using the *local search* operator. While this step is optional, in a wide range of problems, the best results are achieved when a local search is used [124, 456, 457, 458, 459]. This indicates that in a typical project where combinatorial optimisation problems are solved using ACO, it is highly advisable to use local search to help ants find good solutions.

In the next steps, each solution is evaluated, and the pheromone is updated (*updatePheromone* procedure). It is done in two steps: pheromone evaporation and pheromone deposit. Pheromone evaporation is usually performed by multiplying each τ_{ij} with $(1-\rho)$, where $\rho \in [0,1]$ is a parameter called *evaporation rate*. Evaporation is implemented with the intention of "forgetting" very old solutions by decreasing the pheromone concentration for a fixed percentage in each iteration.

Pheromone deposit is performed by increasing the pheromone concentration for selected solution components. There are several variants of the ACO metaheuristic that differ in the precise recipe on how to deposit pheromone. In general, the rule for the pheromone deposit can be summarised as "add most pheromone to the components of the best solution". The simple ant system uses the following rule: given the objective function $f(\mathbf{x})$, for each solution $\mathbf{x}$ created by an ant in the current iteration, add $1/f(\mathbf{x})$ to pheromone values τ_{ij} that correspond to solution components of $\mathbf{x}$. An extension called *elitist ant system* adds an extra step of updating the best solution found so far. Another popular extension, $\mathcal{MAX}-\mathcal{MIN}$ ant system, restricts pheromone values to an interval $[\tau_{min}, \tau_{max}]$, initialises the pheromone to the highest value at the metaheuristic start ($\tau_0 = \tau_{max}$), and in every iteration, only the best ant updates the pheromone. The overall ACO pheromone update equation can be written as

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{\mathbf{x}\in S_u | c_{ij}\in\mathbf{x}} g(\mathbf{x}), \tag{3.4}$$

where the first summand calculates the pheromone evaporation, and the second one is pheromone deposit. In the formula, S_u denotes the algorithm variant set of solutions for which pheromone deposit is performed, and $g(\mathbf{x})$ is a function called *evaluation function*, commonly implemented as inversely proportional to the objective function, $g(\mathbf{x}) = \frac{1}{f(\mathbf{x})}$ [124, 343, 351].

Several differences are visible when comparing ACO to previously described metaheuristics. All previously described techniques keep one accepted solution in each algorithm iteration. ACO generates m solutions in each iteration, and is therefore a population-based method. Therefore, the computer running ACO must reserve extra memory for multiple solutions, as compared to previously described techniques. The metaheuristic is generally more complex, with more parameters to set-up and more decisions to make when implementing (should we evaporate pheromone, which of the pheromone update variant to use, which pheromone update function to use).

Conceptually, ACO is similar to the GRASP metaheuristic. Both techniques use randomised biased solution construction procedures whose goal is to produce a different solution each time, while also trying to select good solutions with higher probability. If local search is used, then the algorithms are even more similar. Still, ACO creates more than one solution at each iteration and uses the innovative framework of pheromone trails to store details about solution quality that can be reused in future construction procedures. Further, it brings a systematic way to integrate anything that is known before solving in the form of heuristic information [124, 343, 351].

3.6.8 Genetic algorithm and evolutionary techniques

The *genetic algorithm*, abbreviated GA is the oldest metaheuristic technique. The basic principles date back to Alan Turing's idea of using evolutionary mechanisms to build "learning machines" in 1950s [106, 460]. During 1960s, a variety of related techniques known under the umbrella term of *evolutionary computation* was developed with the goal of creating generic adaptive systems with various purposes [115, 390, 461, 462, 463]. These techniques had a high impact in the areas of artificial intelligence and optimisations. During the eighties, numerous conferences and specialised journals started to appear, and the volume of papers started to rise quickly [96, 373].

Evolution strategy by Ingo Rechenberg is the earliest evolutionary technique used for optimisation [111, 112, 113]. It could not be considered a metaheuristic in a modern sense since it is a simple method of iteratively introducing modifications to a single solution and keeping the changed version if the last modification improved the solution. This approach lacked a local improvement operator (requirement 3) and therefore also a balancing strategy between local search and the optima avoidance (requirement 5).

The complete genetic algorithm was first proposed by John Holland in 1975 [116]. It is inspired by the natural process of evolution, and the way biological species change and adapt to the varying environment [464, 465]. Genetic algorithm uses a *population* of candidate solutions of a problem. They are gradually changing in the process of simulated evolution that tries to adapt them to the best possible solutions to a given problem. Pseudocode of a basic genetic algorithm is given in Algorithm 11 [343, 373, 466].

Population

The first step of the metaheuristic is creation of the initial population. Each candidate solution is encoded as a *chromosome*, that is a set of *genes*. Each gene g_i corresponds to a variable of a solution $x_i \in \mathbf{x}$, and the set of possible values of a gene is called *alleles*. Each chromosome corresponds to a complete solution to the problem. Rules that convert a chromosome representation into a complete solution and vice-versa are called *genotype-phenotype mapping* in the GA jargon. The idea behind genetic algorithm is that simulating the way a species adapts to

Algorithm 11 Genetic algorithm pseudocode

```
procedure GENETIC ALGORITHM ( )
    $population = initialisePopulation(M)$                ▷ Population size $M$ is a parameter
    $population = localSearch(population)$                ▷ Used in memetic algorithm only
    $\mathbf{x}_{best}$ = findBest(population)                ▷ The best solution found so far
    repeat
        if crossover condition satisfied then
            $crossoverSelection = selection(population)$
            $offspring = crossover(crossoverSelection)$
            $localSearch(offspring)$                      ▷ Used in memetic algorithm only
        end if
        if mutation condition satisfied then
            $mutationSelection = selectMutationIndividuals(population)$
            $mutate(mutationSelection)$
            $localSearch(mutationSelection$               ▷ Used in memetic algorithm only
        end if
        $\mathbf{x}_{best} = updateBest(\mathbf{x}_{best}, offspring)$
        $population = selectNewPopulation(population, offspring)$
    until until end condition is met
    return $\mathbf{x}_{best}$
end procedure
```

the environmental conditions allows evolving generally poorly performing initial solutions into high-quality solutions that satisfy the objective function and the constraints in a great extent. Evaluating individual solutions is done using a *fitness function*, which assigns higher fitness values to higher quality solutions. This means that for minimisation problems the fitness function has an opposite sign than the objective function [343, 373].

Biologically inspired operators—selection, crossover and mutation

Artificial evolution is based on three operators of the genetic algorithm that are performed in the loop of the metaheuristic:

1. selection operator,
2. crossover operator,
3. mutation operator,

where the choice of the name for each operator indicates clear inspiration in analogous phenomena from nature that are studied by biologists. During each iteration, the current population is called *generation*. In each iteration, the *selection operator* defines the subset of the population that will have offspring. The *crossover operator*, sometimes also called *recombination operator* takes two solutions called *parents*, and creates a new individual, whose genes are recombined copies of the chromosomes of the parents. The new individual is called an *offspring*. Finally, the *mutation operator* introduces small random changes to the genes of an individual, similar to the perturbation operator of ILS. The process is repeated with the next generation, until the termination condition is met [343, 373].

There are numerous ways to implement each of these operators. Crossover creates one or more solutions based on typically two, however potentially more parent solutions. Simple examples of crossover are based on the binary representations of the solution where either entire segments or discontinuous sets of bits from each parent are joined in the offspring chromosome. However, this approach can be too simplistic, and in practical applications, the crossover operator is usually problem-specific [343, 373, 416].

Two popular choices for the selection operator include

- tournament selection, and
- fitness proportionate selection.

The selection operator should prefer the best solutions to allow furthering the genes of the best individuals into the next generation. Still, the probability of choosing poor performing individuals should not be zero since despite poor fitness, they might still carry some highly valuable genes. By balancing these two requirements, the algorithm will not go too far investigating poor parts of the solution space, while also not converging to mediocre solutions too soon [343, 373].

Tournament selection is a very simple algorithm where the selected individual is the best out of randomly selected t individuals. It can be configured by varying the tournament size—having a tournament with only a few selected individuals promotes diversity and with $t = 1$, it degenerates into random selection. Conversely, increasing t increases the probability that the best individual will be selected. When t is equal to the population size, then the tournament selection always returns the best solution [343, 373, 416].

Fitness proportionate selection, also called *roulette-wheel selection* is more complex, and it corresponds to the random proportional rule used in ACO. It guarantees that in the population, the selection probability is proportional to the individual's fitness. The term "roulette-wheel" comes from the analogy with a fictional roulette wheel, where the winning number corresponds to the selected individual. Unlike equal segment lengths in a common roulette, in the GA roulette wheel selection, each of the numbers has different width, proportional to the individual's fitness, therefore ensuring that the best individuals have a higher likelihood of winning [343, 373, 416]. A faster version of such selection under the name of *stochastic universal sampling* (SUS) can select several individuals with a single run [467].

Finally, when sufficient number of offspring is created, and the mutation is finished, the new iteration starts, with a new population. The original GA simply replaces old population with the new population, that contains the same number of the offspring as the previous population. Modifications of this simple approach include *elitism*—keeping the best, or several best solutions always present in the population [343, 373, 416].

Extensions and related techniques

The basic GA can be extended in several ways [114, 468, 469]. The local search operator is not present in the canonical GA proposed by Holland. Metaheuristic called *memetic algorithm* uses local search to improve individuals after recombination and mutation, under some conditions [432, 470]. As in the ACO metaheuristics, reports indicate that the local search great potential to improve solutions of pure GA. A related area called *genetic programming* uses the principles of simulated evolution to evolve computer programs instead of programming them manually. Genetic programming is based on an appropriate program representation such as syntax tree, and a population of programs represented in such way, on which selection, recombination and mutation are iterated. For this purpose, the fitness of an individual can be estimated based on the number of errors, required time, achieved accuracy or quality of results.

As compared to other metaheuristics in this Section, it is clear that the genetic algorithm has higher complexity, both in the effort needed to understand the metaheuristic as well as to implement it to solve a problem. It consists of several operators, and each of those operators can be implemented in several ways. Even after choosing the variant of the operator that seems suitable, there is a significant number of parameters that need to be tuned and balanced with each other, frequently in unclear ways [471]. How much mutation, how frequently and to how many individuals should we use? How elitist should the selection be? How large should the population be? Conducting detailed experiments, which can consume a lot of time is the only way to get answers to these questions since these are all problem-specific questions.

The method is similar to ACO in terms that it handles multiple solutions at each iteration. Genetic algorithm and most evolutionary computation techniques are population-based. Benefit of having a population is the potential for more diversity in the solutions and exploring larger areas of the solution space, however this comes at the cost of needing more memory. For all individuals in the population, the value of the objective function must be computed at some point. Therefore, genetic algorithm uses a lot of processing power per iteration, as compared to the single-solution methods.

3.7 Problems with limited budget of evaluations

As described in 2.4.8, problems with limited budget of evaluations (LBE problems) have been identified as an especially difficult subset of optimisation problems. In the literature covering metaheuristic techniques it is mostly implicitly assumed that the evaluation function is fast and cheap to execute. For conventional problems, this allows the algorithm to explore large parts of the solution space before returning the solution. It is not uncommon to evaluate thousands or even millions of solutions in various implementations before the algorithm finishes. Unfortunately, due to various limitations such as slow evaluation function and very high computational demands, evaluating large number of options is prohibitively costly in LBE problems. Metaheuristics cannot rely on evaluating large number of solutions in LBE problems, instead they might have access to e.g. only a thousand or a few hundred evaluations [150, 173, 175]. A separate issue is how to handle constraints if the LBE problem has them, some successful results can be found in [472, 473, 474, 475, 476, 477, 478, 479].

There are two general directions in solving such problems:

1. applying an optimisation algorithm directly to the expensive objective function [174],
2. developing a *surrogate model*, also called *meta model* that is cheap to evaluate and used to approximate the expensive objective function [173, 175].

The advantage of the direct approach is simpler implementation since there is no need to invest effort to build a surrogate model and integrate it into the general algorithm. Disadvantage of this approach is a need for specialised tuning and using as much problem specific details as possible since the algorithm will otherwise run too slow [174]. Nevertheless, such direct solutions are rare in the LBE literature. Most published research reports good results with surrogate models since they allow the algorithm to work quicker at a cost of loss of accuracy. Their advantage is the ability to get larger number of objective function estimations. A disadvantage of such approach is higher complexity of implementation which requires a surrogate model to be prepared and then integrated into the algorithm. Deciding when and how to use a fast surrogate, which points to select based on what is known, and when to use the expensive original can be a highly complex research question [173, 175].

When using a surrogate model for a new problem, the biggest obstacle is the uncertainty that the simplified model will represent the objective function faithfully. Such issues appear in the machine learning community and there are techniques to control model errors such as dividing the dataset into training and validation set, detection of overfitting and others. These techniques are essential when assessing the general reliability of any model, unfortunately they are typically developed for a different purpose, not for the direct use in an optimisation algorithm [8, 480, 481, 482].

When optimising using surrogates, it is not critical that the surrogate model provides absolutely accurate values of the true objective function, it is more important that the estimate is good enough to lead the algorithm in the direction of the optimum. Unfortunately, there is usually not much guarantee that a surrogate will do that. Situations when a surrogate mistakes a point as a local optimum is a notable issue called “false optima”, where the point is a locally optimal value for the surrogate model, but not the original objective function that needs to be optimised. Such errors can keep the algorithm in a false optimum for considerable time and waste a lot of resources. To prevent this, surrogate based meaheuristic usually have a strategy to systematically compare the surrogate with the expensive original objective function [173, 175, 480, 483]. While complex, there also exist studies that involve more than one surrogate [484, 485, 486]

3.7.1 Commonly used surrogate models

The surrogate based optimisation literature has provided investigations of various types of surrogate models, from polynomials, to statistical and machine learning methods. Some metaheuristic-specific techniques such as fitness inheritance have also been widely applied. Finally, some researchers combined various surrogates and developed methods for selecting the surrogate under various circumstances [173, 175, 487, 488].

The *polynomial models*, sometimes also called the *response surface model* method typically use second-order polynomials to estimate the original expensive objective function. The coefficients in the polynomials can be calculated using least squares method and gradient method. In both methods, the number of needed samples to get good results is proportional to the squared number of input variables. For problems with high dimensionality, the least squares method can have too high computational cost, and in such cases the least squares method is preferred [172, 489, 490, 491].

A very popular statistical method for approximating the expensive objective functions, called *kriging* was initially invented for application in mining industry. It was named by Daniel Gherardus Krige who achieved good results in applying statistical methods for estimating the unknown distribution of gold under ground, based on a limited number of test drills. The technique is also called *gaussian process regression*, *Wiener-Kolmogorov prediction*, and *spatial correlation modelling* [492, 493, 494, 495, 496, 497, 498]. Using kriging, a function values are modelled based on Gaussian processes, the assummption that a function values will be similar in nearby points, and that this similarity will drop as the distance between the points increases. Given n points where the function values are available, interpolating the function value in an unknown point has a computational complexity of $O(n^3)$. Kriging provides uncertainty estimates for each interpolated point [499]. The uncertainty estimates provide opportunities for active selection of points to sample using the expensive original function and balancing the model accuracy and exploration of promising regions. This idea is the basis for the *efficient global optimisation* (EGO) algorithm [500, 501]. Some applications of Kriging with metaheuristics can be found in [175, 483, 502, 503, 504].

Techniques used in *machine learning* are another natural framework for the problems of objective function approximation. Popular choices include *neural networks* and *support vector machines* [172, 175]. Neural networks are inspired by the research on brain. They are represented as collection of connected artificial neurons, which can be efficiently trained based on input data to provide an expected output when given a typical input [505, 506, 507]. *Support vector machines* are models that can be used for classification and regression analysis based on statistical models of the learning process [508, 509, 510, 511, 512, 513, 514]. Applications in the area of LBE problems can be found in [485, 515, 516, 517, 518, 519, 520].

In addition, techniques that use specific functionalities of metaheuristics have been developed to run with less evaluations while solving LBE problems. *Fitness approximation* is an example of such technique used in genetic algorithms. Using such approach, a child of two parents with known fitness can be assigned an average fitness of the child's parents or a weighted average based on the proportion of the inherited genetic material [521, 522]. While this procedure is very fast in estimating the fitness of an individual, critics highlight that it was proven to give satisfactory results only with very simple objective functions such as continuous and convex problems [523].

It is difficult to select the appropriate surrogate for a given problem. The choice of the best surrogate can in great deal depend on the problem. Further, the literature that compares various models is limited and sometimes focuses on simple problems. Therefore, the choice of the appropriate surrogate can be difficult, and practitioners are sometimes left to their intuition to select the surrogate if implementations and comparisons of several methods are not possible. Some studies that compare various surrogate models can be found in [524, 525, 526, 527, 528, 529, 530, 531].

Metaheuristics that have so far been successfully used with surrogate include in most part evolutionary computation [175, 525, 532, 533, 534, 535, 536]. While the method of using surrogates can be used with other metaheuristics as well, using other metaheuristics is more rare in LBE community. Some examples include simulated annealing, artificial immune systems, particle swarm optimisation [537, 538, 539, 540, 541].

Chapter 4

Bottom-up development methodology

The area of metaheuristics received great attention in the last decades as they have favourable characteristics of being highly generic, and having great potential for achieving good results [8, 105, 343, 542]. Research effort and progress has been so great that the scale and complexity that can be solved nowadays seemed out of reach only a few decades ago. Despite the progress, using metaheuristics still poses numerous challenges. A notable issue with metaheuristic projects is the fact that these methods are still complex and require considerable adaptation to specific problems before they can be used [8, 96, 234, 542]. Projects that are based on metaheuristic techniques require experienced experts, and therefore, the development can be slow and costly. A lack of development methodologies does not help mitigate that issue. Further, dozens of proposed metaheuristics and their variants can be confusing to the newcomers and requires a significant effort in learning [235]. Even under expert guidance, the process of implementing metaheuristics in some parts still resembles more an art than an engineering process grounded in rigorous science [96].

This Chapter presents a bottom-up development methodology devised to minimise the implementation effort. The proposed methodology is grounded in the fact that established metaheuristics share similar operators, and on the no free lunch theorem, which provides a formal basis for the algorithm efficiency comparison. Established metaheuristics, discussed in the previous Chapter are decomposed into the required operators, and the development starts with the simplest components, to which complexity is incrementally added. By gradually adding increasingly complex components to the metaheuristic, while performing performance evaluations in each step, the development can stop when the results are considered satisfactory, this way saving development time and cost if simple metaheuristics perform sufficiently well.

4.1 Motivation

The development of metaheuristics that can be observed in the last decades can be viewed as a result of two overlapping research efforts:

- researching general problem-solving behaviours, and developing adaptive systems that would be able to find solutions to various problems without explicit programming by humans, in the area of artificial intelligence [96],
- developing more general *heuristic* algorithms, reducing their complexity and reducing the required development effort in the areas of operations research and optimisation [8, 542].

There has been great progress in both directions. Techniques of genetic programming can evolve procedures for tasks of varying complexity without much intervention of a programmer [423, 543]. Likewise, the wide area of reactive search, hyperheuristics, and metaheuristic tuning techniques have a potential to relieve the developer from some parts of the development process [8, 344, 379, 544, 545, 546]. Metaheuristics are nowadays considered to be a standard way of solving especially large and difficult optimisation problems. They are popular both in academic research as well as in practical and commercial applications [96], and for numerous problems, they are achieving state-of-the-art results [150].

Despite considerable progress, we are still far from the ideas of fully autonomous software systems that solve complex tasks with minimum intervention from the human developers. Metaheuristics did achieve a very high degree of generality, nonetheless, successful applications require significant effort to adapt the metaheuristic to the specifics of the problem being solved [234]. Further, projects that use metaheuristics require experienced developers with high level of expertise. If high performance is needed, implementing metaheuristic techniques can become a complex research project [96, 235, 547].

When starting a project based on metaheuristics, mutually related questions like these come up naturally:

1. “Which metaheuristic should I use? ”,
2. “Which metaheuristic will give the best solutions for my problem? ”,
3. “Which metaheuristic will be easiest to implement? ”

The question, in a general sense, unrestricted to metaheuristics was first formulated in 1976 by Rice [548]. A large number of published metaheuristic techniques only complicates this problem further. While these are all fundamental questions, the existing literature provides limited support to answer them. Research projects usually focus on the details of a single method and do not address these problems directly. In the published work that does address the problem of best performing metaheuristic, several types of answers are provided:

- “This is not known in advance since metaheuristics by their definition do not provide performance guarantees” [8, 69, 150, 405],

- “It depends” [234],
- “You need to compare the performance of various metaheuristics to find out [8, 405, 549]”,
- “Use hyperheuristics, algorithm portfolios or machine learning as a procedure to find out” [344, 366, 550, 551, 552].

There is no doubt that all of the above answers are correct and based in deep understanding of the difficulties and complexity involved with building efficient optimisation algorithms. Experimental evaluation is indeed the only way to compare different techniques with certainty, nevertheless the methodology for rigorous testing and comparisons of metaheuristics is still developing [549, 553]. Hyperheuristics and algorithm portfolios are an attempt in establishing procedures that can do a part of the development tasks, this way avoiding a need for several labour-intensive development steps, however they add extra complexity, require a high level of expertise and have very high computational requirements. A big drawback of any experimental evaluation is the *aposteriori* essence of such methods—they can be used only after implementing two or more different algorithms.

Unfortunately, the above answers are not helpful at all to practitioners, experts in other areas and anybody else who wants to solve their optimisation problem, that might not be complex. This is especially notable if the practitioners do not have deep experience in the area and cannot reliably estimate the complexity of various techniques. Indeed, advice such as “Try out five metaheuristics to find which one works best” completely ignores the immense effort and expertise needed to first implement and then rigorously compare several different metaheuristics. In practice, software projects are regularly done under deadlines and with limited staff and computational capacity. Instead of first implementing five different techniques in order to compare them, then discarding the four algorithms that are not the winners, it would be much more economical to implement only one, that balances the implementation effort and good performance, depending on the project ambitions and resources and start out with a reasonable project strategy to achieve this goal.

In a typical practical project, the metaheuristic is often selected based on the existing publications and experiences of others in solving the given problem or similar problems. Published results reporting good results of some technique can be viewed as an encouragement to try and reproduce the same technique. The developer’s level of expertise in different techniques is an additional bias that is present even with highly experienced researchers and developers. For someone with years of experience building genetic algorithms, it is easier to simply do one more implementation of genetic algorithm, than to spend time learning the details of e.g. iterated local search. This is a myopic view since a simpler technique such as iterated local search can be much quicker to implement and could save development time. Furthermore, these personal biases are an important underlying cause of close segregation in research groups in the area of metaheuristics and optimisation in general that was highly pronounced until recently [554, 555, 556].

Bridging the gap between research and practice

The above discussion illustrates a prominent issue in the metaheuristics community: an apparent lack of development methodologies that are robust enough to allow building high-quality optimisation software, however also economical in terms of required staff, time and computational resources [234, 554, 557]. It is a symptom of a wider dichotomy between academia and software engineering practice. Despite the initial idea that metaheuristics will simplify the process of developing optimisation algorithms, a gap still exists between the output of research projects and what would be required for a direct application in the industry [345, 346, 558].

The academy is regularly focused on developing innovative techniques with the goal of further performance improvements, commonly referred to as the *up-the-wall game* [559]. It is only recently that there is an increasing research effort devoted to better understanding of the existing methods, including the reasons why certain techniques work on certain problems, even if there is no immediate performance gain [234]. Furthermore, practical problems commonly have a large number of complex traits and numerous constraints. Due to the difficulties in realistic modelling of these problems, and the desire to put more focus on the algorithms, academic research is often done on simplifications of practical problems. Finally, nearly all academic work assumes that the problem definition is final and that the requirements will never change [557]. While this is reasonable from academic perspective, it limits the direct applicability of devised methods to the real problems [283, 346, 560].

An excellent overview of the use of metaheuristics in the software industry is given by Edmund Burke and Emma Hart, who in [235, 364] state the following:

> *Many state-of-the-art meta-heuristic developments are too* problem-specific *or too* knowledge-intensive *to be implemented in cheap, easy-to-use computer systems. Of course, there are technology provider companies that have brought such developments to market but such products tend to be expensive and their development tends to be very resource-intensive. Often, users employ simple heuristics which are easy to implement but whose performance is often rather poor. There is a spectrum which ranges from cheap but fragile heuristics at one extreme and knowledge-intensive methods that can perform very well but are hard to implement and maintain at the other extreme. Many small companies are not interested in solving their optimisation problems to optimality or even close to optimality. They are more often interested in* “good enough—soon enough—cheap enough*” solutions to their problems.*

The above citation has been confirmed times and times again by the practical experience of the author of this thesis. Very often, the great effort needed to solve general classes of problem instances is not required as the project might be focused on a subset of smaller problem instances. In other cases, it is enough only to outperform humans that were previously doing the task of the optimiser to consider the project as a success.

Unlike in academia, a big source of complexity is highly specific real problems, that can be difficult to communicate effectively and formally define. A problem might have very complex constraints. In some cases evaluation of a solution can take a long time or require significant computational power. Finally, the changing requirements are a reality of the software industry, where significant changes to the problem formulation are expected during the course of a software project, e.g. when a product is sold to a different customer with a slightly different problem specifics. There has been a wide array of techniques to improve software development methodologies to allow better adaptability of software and more resilience to change, uncertainty and incomplete specifications, such as test-driven development and agile methodologies [561, 562, 563]. Instead of relying on a complete and fixed specification, these methodologies rely on small improvements and frequent communication with the customer, this way allowing quicker responses to any changes during the implementation process.

Building on the issues discussed above, a notable and as of 2019 still insufficiently addressed criticism of metaheuristics includes an apparent lack of a universally applicable development methodology [234]. This is such a notable issue that there exists an entire conference devoted to this and related issues. The web site of the *SLS2019: International Workshop on Stochastic Local Search Algorithms* [564] states:

> *Development of effective SLS algorithms*[1] *is a* complex engineering process *that typically combines aspects of algorithm design and implementation with empirical analysis and problem-specific background knowledge. The difficulty of this process is in part due to the complexity of the problems being tackled and in part due to a large number of degrees of freedom researchers and practitioners face when developing SLS algorithms.*
>
> *This development process needs to be assisted by a* sound methodology *that addresses the issues arising in the phases of* algorithm design, implementation, tuning and experimental evaluation. *In addition, more research is required to understand which SLS techniques are best suited for particular problem types and to better understand the relationship between algorithm components, parameter settings, problem characteristics and performance.*

[1] SLS algorithm is an abbreviation for *stochastic local search algorithm*, a synonym for metaheuristic.

The methodological publications are rare, with a few exceptions in [8, 547, 549, 557, 565]. While initiatives advocating better formalisation of the research and early proposals of design patterns do exist, this direction in the science of metaheuristics started only recently [365, 557, 566, 567]. It is now generally understood that metaheuristics cannot be viewed as a set of independent and distinct optimisation techniques. Instead, the *component based view* defines each metaheuristic as a set of components that can be reused and assembled in various ways. The component-based view was used as a justification for research into techniques that can procedurally assemble hybrid metaheuristics while minimising the need for decisions by the developers [344], nevertheless no study so far used it to investigate the potential to improve the development methodology of metaheuristics with the goal of reducing development time and labour. In [565], Zäpfel et al. mention that the top-down implementation approach is legitimate and often used, and that the bottom-up approach might be better suited for practice as well as teaching. Still, the authors do not go further in the analysis and do not mention the benefits and the potential that the bottom-up approach has to allow faster implementation and save the development cost. Much of the related research is also done in the area of methods for algorithm tuning and techniques for comparisons of metaheuristics [8, 549, 553, 557, 568].

Towards a general metaheuristic development methodology

The potential of the component-based view in practice and the implications to development methodologies remained largely unexplored. While there exists a number of software libraries with the goal of utilising the component-based view to provide reusable components for building metaheuristics, each of them has a slightly different development model and limitations. It is a common dilemma in software engineering: use a library for some task or do your own implementation of the functionality. General risks with third-party libraries include security, level of support, possible termination of the development, and unexpected bugs that might not be under control of the users of the library and can take considerable time to resolve. Finally, along with learning the theory, which is needed for the use of most libraries, a certain amount of time is also needed to learn the library that is to be used.

This Chapter focuses on cases when the external libraries are not used, and when for any reason, an implementation from scratch is required. It provides a proposal of a clear and simple development methodology whose goal is to develop efficient-enough metaheuristics using minimal development effort: expertise, time spent in development and learning, and computational capacity. Unlike the top-down approach that is implied in much of the literature, the proposed methodology is a bottom-up design. By starting with the simplest metaheuristic components, more details and more complex components are incrementally added. Using this philosophy, development can stop as soon as satisfying results are achieved.

4.1.1 Current practices in impelementing metaheuristics—a fractal of complexity

Metaheuristics are generally last-resort methods, that are used to solve problems for which no simpler and faster methods are known [343]. A most important source of complexity in metaheuristic implementations is the difficulty of the problems that need to be solved. Very commonly, they are used for $\mathcal{NP}$-hard problems, for which no efficient algorithms are known. A traditional top-down approach would usually include some variations of the following steps:

1. Select the metaheuristic to implement,
2. Implement the general framework for the metaheuristic, the main loop and e.g. the population handling in GA or the pheromone graph in ACO,
3. Decide which operators will be used,
4. Perform tuning.

As seen in the overview available in Chapter 3, metaheuristics vary in the complexity, ranging from those that can be summarised in only a few sentences, to those who have several intricate operators.

The above process has a very high number of combinations. For each metaheuristic, there are several operators (out of which some are optional). For each operator there exists several flavours of the operator (e.g. selection in GA can be made using SUS or tournament selection). For each flavour, there can be several parameters (e.g. if tournament selection is used in GA, the tournament size needs to be set). The above discussion illustrates the need for a large number of good decisions that need to be done when implementing a metaheuristic. All these decisions are problem instance-specific since good values for a certain problem instance might perform poorly on other problem instances and vary even more when used on a completely different problem [8, 124].

All the operators must act harmoniously with each other in order for the metaheuristic to work well. If this is not achieved, the algorithm performance will not be satisfying or the algorithm might not work at all. Finding good combinations of structural decisions and parameters is not an easy task, since interactions between parameters can be difficult to comprehend [471], and the efficiency of different configurations of metaheuristics needs to be evaluated experimentally. Despite high complexity, the process of testing several different configurations is very important since this allows finding a good balance between intensification and diversification, as discussed in Section 3.4.

In practice, metaheuristic development is typically an iterative process based on resource-intensive experiments, and guided by the developer's experience, intuition and preliminary experimental results [96]. Any change to the problem definition, e.g. adding an additional constraint or changing the objective function can mean that the entire process needs to restart

[8, 557]. This process in a great deal holds more resemblance to an art than to engineering [96].

4.1.2 Tuning metaheuristics

Each metaheuristic itself can have numerous parameters. In GA, e.g. population size, elitism and tournament size need to be set. In tabu search, the length of the tabu list needs to be set. In GRASP, the length of RCL needs to be set. The problem of choosing the metaheuristic parameters is difficult due to two facts:

- metaheuristics generally return a different solution each time [105],
- good parameters are not problem-specific, they are problem-instance specific [8, 124, 557].

Stochasticity of the results produced by metaheuristics implies that the experimental evaluation cannot be performed on a single run only. Instead, a statistical analysis of multiple runs on the same instance is needed to draw reliable conclusions from the experiment. The second issue is the fact that good performance on one problem instance does not imply good performance on other problems the metaheuristic needs to solve. Therefore, an experimental evaluation of a single configuration requires multiple runs on multiple problem instances. Assuming that we use 30 runs per instance and that we have 10 instances as the representative set for a problem, we need 300 runs to evaluate each algorithm configuration. Clearly, detailed tuning is resource-intensive and becomes intractable if large number of configurations needs to be evaluated or when each run is long.

To help speed up the process of tuning, it is possible to use procedural tuning [8, 546], where different techniques such as racing algorithms, surrogate-based modelling and heuristic search techniques have been proposed. Still, due to the vast number of different configuration options, and the exponential number of combinations when numerous parameters are involved, some filtering and prioritisation by a human developer is almost always necessary.

4.1.3 Metaheuristic design patterns

Software design patterns are reusable solutions for common problems that occur in software development. They are inspired by similar ideas in a completely different area—architecture, where in [569] Alexander considers high-quality architectural solutions across long periods of time and suggests to call them “patterns”:

> *(...) each pattern describes a problem which occurs over and over again in our environment, then describes a core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*

In their seminal book [570], Gamma et. al. define *software design patterns* in object-oriented software development as

> *(...) descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.*

In addition to the definition, the authors provide a catalogue of patterns divided into several categories. Their work helped shape the contemporary software industry, and nowadays design patterns are a commonly used accumulated wisdom related to difficult problems in software design [571].

In the area of metaheuristics, there has been a similar initiative to document design patterns that help solve common problems that are specific to the development of metaheuristics [572]. Goals of such research are to document reusable solutions, allow wider access to this catalogue, and standardise the notation by e.g. using the unified modelling language [573]. The GECCO[2] conferences in 2014 and 2015 had dedicated workshops that were advocating design patterns in metaheuristics [574, 575].

In [576], Lones identifies several concepts used in nature-inspired metaheuristics, such as local search, hill-climbing, variable neighbourhood search, multi-start, adaptive memory search (tabu lists), population, intermediate and directional search, and search space mapping (used in ACO). The single point search algorithm pattern is presented in [577], where algorithms such as ILS and VNS are presented in a common framework with the goal of creating hybrid metaheuristics. Ben Kovitz and Jerry Swan discuss various types of search history usage, which they call "tagging" in [578]. The "structural stigmergy" is a proposed name for a complex interplay of several other components [579]. Composite operators are discussed in [580]. Improving the ways the solutions are presented when humans are evaluating solutions is defined as "Interactive Solution Presentation" pattern in [581], where several different ways to improve the process and reduce the fatigue of involved people is suggested. Using various types of surrogates for modelling "slow" objective functions is discussed in [582].

While the above work correctly identified various common techniques used in metaheuristics, the impact these design patterns had has so far been limited. They correctly identify the issues with the limitations of the traditional top-down view, and the need for documenting the patterns. Nevertheless, they do not proceed further in investigating the potential that such decomposition has to speed up the development of metaheuristics and typically limit the discussion to their potential for constructing hybrid metaheuristics and hyperheuristics.

[2] GECCO is an abbreviation for the Genetic and Evolutionary Computation Conference

4.2 Theoretical foundations

This work is in large part inspired by the no free lunch theorem and the convergence proofs for various metaheuristics. The *no free lunch theorem*, presented in more detail in Section 2.5.3 is a rigorous mathematical proof with deep consequences for the field of optimisation. It states that the average performance on the set of all possible problems is equal for any two algorithms [145]. In the optimisation community, it has been used to illustrate the need for more specialised algorithms [220], and invest more effort in e.g. metaheuristic tuning [8]. It also implies that for no two metaheuristics, it can generally be said that one is more efficient than the other. However, in the interpretation that advocates specialisation, it can be argued that a better metaheuristic would be the one that can be adapted more closely to the relevant problem, this way highlighting the importance of the low-level operator adaptation by e.g. choosing the appropriate neighbourhood definitions and implementing efficient problem-specific moves.

Similar theoretical results arise from the area of convergence analysis. For some metaheuristics, such as GA, ACO and SA, it has been proven, that given enough time, the probability of converging to the optimum is equal to one. Interestingly, the same proof exists for the pure random search, which is a trivial algorithm [368]. These formal proofs do not give any conclusions about the required time to reach the optimum, which might be millions of years, therefore they have limited consequences for real applications. Still, both the no free lunch theorem as well as the convergence analysis show that from a purely theoretical point of view, using more complex metaheuristics does not give any guarantee that the results will be better.

Finally, there is a consensus in the research community that all metaheuristics fit the intensification and diversification framework (I&D) [162]. The I&D framework is described in more detail in Section 3.4, and it is an important step towards a unified view of metaheuristic algorithms in which all metaheuristics are governed by an interaction of these two ways to move in the solution space. Most components of the metaheuristics can be placed in this spectrum, between pure intensification (moves guided by the objective function) and pure diversification (random moves not guided by the objective function). Further, it is common that a metaheuristic has one component that is used for most of the intensification, and conversely, one component that does the most of the intensification. The unified view on metaheuristics has greatly inspired this work that tries to use this universality principle to propose a unified development process to build any metaheuristic from the I&D framework.

4.3 Practical foundations

4.3.1 Implementation complexity and development effort

The goal of this Chapter is to provide a development methodology that can develop efficient algorithms, while minimising the *development effort*. For the purpose of this thesis, the development effort is defined as the total effort required to learn, implement and tune a metaheuristic. The definition of the development effort will always be somewhat vague since it involves individual aspects related to the developer skills, expertise, previous experience with different metaheuristics etc., as well as the subjective perception of difficulties of certain tasks. Typical ways to evaluate this effort can be the total number of man-hours needed to complete a project, total CPU-time needed to tune the algorithm, total makespan[3] of the project, or total development cost.

As discussed in Chapter 3, GA and ACO, as compared to single-point methods have more operators and more degrees of freedom in the ways to implement them. All this means that generally, the effort required for the implementation of GA and ACO will be higher with regard to all three criteria mentioned above. GA requires longer development time due to a large number of operators, ACO has an added complexity of implementing artificial ants and the pheromone graph that corresponds to solution components. Both GA and ACO have a much wider set of structural options and parameters, therefore the tuning process will be more demanding. More complex implementation and more demanding tuning imply generally longer makespan or higher cost. Therefore, SA, TS, ILS, GRASP and random restart local search will be referred to as *simple methods*, while GA and ACO will be discussed as *complex methods*, where simple and complex are related to the development effort.

To illustrate the above discussion, let us assume that we have a single optimisation problem and that we want two development teams to solve it using different techniques: Team GRASP, who works on implementing GRASP, and Team GA who is implementing the genetic algorithm. Team GRASP needs to devise two components a solution construction procedure and a local search procedure. Team genetic algorithm, must develop at least five: a solution construction procedure, mutation, crossover, selection, general population handling, and preferably also local search.

When it comes to tuning the algorithm parameters, team GRASP needs to decide on two parameters: the restricted candidate list size and the local search parameters. Team GA needs to tune seven: mutation intensity, frequency of mutation, crossover parameters, the tendency towards selecting best individuals, elitism parameters, population size, local search parameters. Clearly, there is a great number of combinations of these parameters, with potentially complex interactions among them, while the basic GRASP has only two, which simplifies tuning.

[3] *Makespan* is the time that elapsed between the start and the end of the project

Finally, GA and ACO are population-based, therefore they have higher memory requirements and need to evaluate more solutions per each iteration. This implies higher computational requirements for running and tuning these metaheuristics. The added memory requirements might be prohivitively big with high population size and large problems that require a lot of memory.

4.3.2 Empirical studies of metaheuristic efficiency

Comparing different metaheuristics on the same problem is labour and resource-intensive. Perhaps this is the reason for limited research in this direction. A notable example of such investigation is presented in [320], where a group of authors in the Metaheuristic Network project [560] compares the performance of evolutionary algorithms, ant colony optimisation, iterated local search, simulated annealing, and tabu search. All these methods were applied to solving the university course timetabling problem with several artificially generated problem instances. Problem representation and local search code were shared with all metaheuristics in an attempt to provide fair comparison. The results indicate that no heuristic is a clear winner on all instances, even when similar instances are considered. This illustrates the difficulty of identifying the best metaheuristic on wide problem classes. Still, some trends can be observed. In the small problem instance, ILS, followed by SA is the best. With medium instances, SA is the best, followed by ILS. On large instances, in terms of hard constraints, Tabu search is the best, followed by ILS.

Similar results were observed in [583], where François et al. compare genetic algorithms, tabu search, several ant colony optimization techniques, and (interestingly) recurrent neural networks to optimise fuel reload patterns in boiling water nuclear reactors. The goal of the optimisation was to increase the produced energy while satisfying the thermal and reactivity constraints. The best average performance was achieved using the recurrent neural network, and tabu search was the second best. The best performing individual solution was found using tabu search.

In [584], Sönke Hartmann and Rainer Kolisch compare tabu search, genetic algorithm and simulated annealing for solving the resource-constrained project scheduling problem. For simulated annealing and genetic algorithm, the authors also compare the performance with different problem representations. In addition to the metaheuristics, several direct heuristics were compared. In this work, genetic algorithm and simulated annealing with the “activity list” representation worked best. The authors note that their experiments indicate that the choice of an appropriate representation has been far more significant than the choice of the metaheuristic. Further, they indicate that initial solution generator procedures are an important component in metaheuristics, and that tabu search was the best technique when the number of allowed evaluations was limited.

Antosiewicz et al. in [585] compare genetic algorithms, simulated annealing and tabu search with less established techniques—quantum annealing, particle swarm optimization and harmony search on the travelling salesman problem under a limited budget of evaluations. Their results indicate that simulated annealing and tabu search clearly outperformed other techniques. Further, tabu search was the fastest to converge.

4.3.3 General experience

In his invited talk at the EU ME meting 2009 conference, based on his experience in the Metaheuristics Network project [324] Thomas Stützle identifies the following as the most important contributors to successful implementation:

- Creative use of general ideas and insights into the algorithm behaviour and its interaction with the problem specifics,
- Expert developers,
- Sufficient time that permits implementation and detailed tuning.

More importantly, similarly to [584], Stutzle highlights that efficient local search neighbourhoods and other underlying heuristics were highly important for the success of the algorithm, while the strict adherence to the rules of a specific metaheuristic was less important.

From the perspective of this work, it is remarkable that simple techniques such as ILS, SA and TS were consistently outperforming GA and ACO in three out of four studies presented above, while in [584] GA and SA results were similar. In the area of metaheuristics, any generalisation is very difficult, as is the objective algorithm comparison [553]. Excellent results of simple techniques, and the understanding that efficient components can be more important than the selection of the metaheuristic indicate a very important premise of this work: *more complex metaheuristics are not a guarantee of better performance*.

4.4 Standard metaheuristic components

Chapter 3 brings a brief overview of several commonly used metaheuristics. Their analysis uncovers numerous common components that they share. The three ubiquitous components are:

1. Initial solution generation procedure $initialSolution()$,
2. Perturbation operator (diversification component) $pertrub(\mathbf{x})$,
3. Local search operator (intensification component) $localSearch(\mathbf{x})$.

Each operator definition can vary depending on the metaheuristic. Further, depending on the metaheuristic, the same operator might be named differently. The mutation operator in the genetic algorithm is functionally equivalent to the perturbation operator in the iterated local search and the shake operator in VNS. The basic functionality of introducing random moves undirected by the objective function is the same, therefore in this work, all such operators will be called *perturbation*. Likewise, all components that provide a local optimum as a result will be called *local search*.

The bottom-up methodology presented in this work is based on building reusable components that can be assembled into various metaheuristics. Components are implemented as an evolution of previously developed components if possible. A principal technique used to achieve savings in the development effort is the classification of the effort into small incremental changes and complex modifications that require either writing the component from scratch or a complete rewrite, or a significant addition of new functionality. In the diagrams that show the evolution of each component from simplest to more sophisticated, a full-line arrow is used for the complex modifications, while quick to implement incremental improvements are drawn in a dashed arrow.

4.4.1 Initial solution generation procedure

The *initial solution generation procedure* is the component that provides the initial solution for the single point algorithm or a population of solutions for population-based methods. It is generally considered that this procedure must be fast to allow sufficient time for the metaheuristic to work. Usually, pure random variable choice, denoted $initialSolutionRandom()$ or a simple greedy algorithm, denoted $initialSolutionGreedy()$ is used

Variants of the greedy approach are used in GRASP and ACO, where unlike other discussed techniques, the solution construction procedure is fully specified, and both are based on greedy solution construction. In GRASP, the canonical greedy algorithm provided in Algorithm 1 is modified by adding a restricted candidate list as a technique to include biased randomness. The size of the restricted candidate list is a parameter that controls the intensification of the procedure, as described in more detail in Chapter 3.6.2. In the remainder of this work, the greedy randomised solution construction used in GRASP will be denoted $initialSolutionGRASP()$. While more complex than random selection, the implementation of GRASP is still a small incremental modification of a greedy algorithm.

Creating solutions in ACO is more complex. In ACO, solution creation is iterated in each algorithm step to create a population of solutions. It uses the randomised greedy construction philosophy similar to GRASP, however using a different approach to selection of variable values. Implementing the ACO solution initialisation requires additional data structures to store heuristic information and the *pheromone graph*. Pheromone graph is a solution representation specific to ACO and it might take a while to develop, especially to beginners and if the problem is not tightly related to graph problems. In the ACO solution construction, each solution construction procedure represents an artificial ant, that instead of basing the decisions on the in-

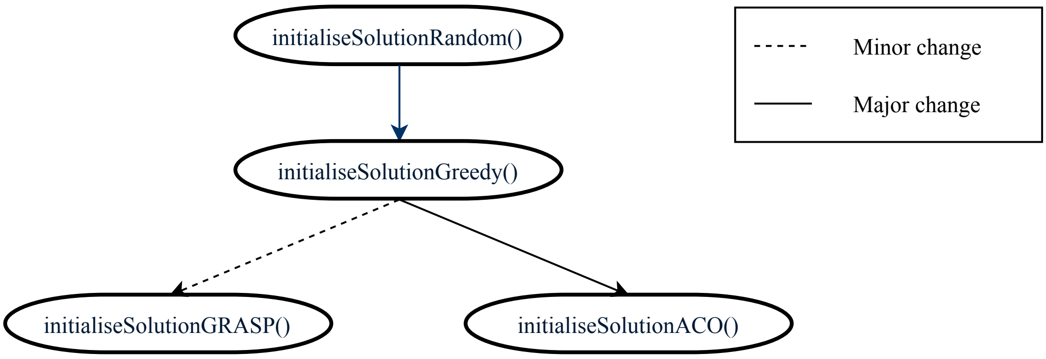

initialiseSolutionRandom()
initialiseSolutionGreedy()
initialiseSolutionGRASP()
initialiseSolutionACO()
Minor change
Major change

Figure 4.1: Initial solution construction operators evolution

cremental objective function evaluations chooses solution components in a biased random way, such that the components with higher pheromone concentration and higher heuristic information have a higher probability to be selected. This procedure will be called $initialSolutionACO()$ in the rest of this work.

Initial solution generator procedures are always constructive, since they provide a complete solution from scratch. Figure 4.1 shows a recommended evolution of the initial solution construction procedure. In almost all cases, random initialisation will be the quickest to implement. There are several other more complex to implement initial solution generation procedures are possible. A full-line arrow between operators indicates that the more complex operator needs to be developed from scratch or as a significant improvement from the initial operator. Conversely, dashed arrow between operators is used when an operator can be developed as an incremental, small modification of the initial operator. A typical improvement over a random initialisation would be a greedy approach, however it requires additional and problem-specific development effort. Building on the greedy algorithm, it takes a simple and incremental improvement to modify the algorithm into the GRASP solution construction, while the ACO solution construction using artificial ants requires more extensive development of features specific for ACO metaheuristic.

4.4.2 Local search

Local search is a widely used idea in metaheuristics. It is so efficient and has such a high potential to improve the results that it is by definition used in nearly all metaheuristics, and even the metaheuristics that do not explicitly define local search as their component generally recommend its use (e.g. ACO) or can be extended to include local search (e.g. SA and GA) [442, 470]. Basic algorithmic template and description of related ideas for local search are briefly presented in Section 3.5.4. Local search is mostly an intensification component.

In most metaheuristics, there is no precise specification on how to implement the local search, instead it is problem-specific and closely related to the neighbourhood definition. It can

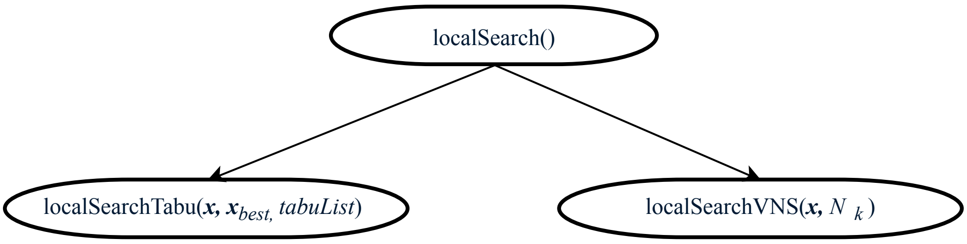


Figure 4.2: Local search operators evolution

be deterministic or stochastic. The basic version can be reused in almost any metaheuristic, however to be usable with TS and VNS, the local search operator needs to be extended. In tabu search, the local search needs to be extended by the tabu list functionality and ensure that it will not go back to the solutions in the tabu list. The VNS metaheuristic requires implementation of several different local search procedures, so that each uses a different neighbourhood definition. This can be implemented by e.g. parametrising the current local search operator, so that the neighbourhood size becomes a parameter, by implementing several procedures that use different neighbourhood definitions or by combining these two approaches.

The evolution from the simplest to more complex operators is shown in Figure 4.2. The VNS local search, denoted $localSearchVNS()$ has two parameters: the neighbourhood definition to use and the neighbourhood size. Having several neighbourhood definitions with neighbourhood size as a parameter is generally a good practice, unrelated to VNS. Specific variants of such local search can be plugged in into all other metaheuristics and tuned so that the best performing neighbourhood definition size is selected. While VNS local search is reusable, the Tabu local search, denoted $localSearchTabu()$ is usually restricted to the use with the tabu search metaheuristic, except with hybrid metaheuristics. It has the tabu list and current best-found solutions as arguments, in addition to the point around which to search.

4.4.3 Perturbation operator

The *perturbation operator* results in the diversification of the search. It introduces random modifications to the solution without any regard to the objective function. It is used in SA, ILS, VNS (where it is called a *shake* function), and GA (where it is called *mutation*).

The implementation of the perturbation is closely related to the neighbourhood definition of the local search, and the intensity of the perturbation should be balanced with the intensity of local search (if used) and other intensification components. While this operator can be developed independently, due to close ties with the local search, it can be a good strategy to implement the local search first. After the local search is finished, the perturbation can usually be implemented in a trivial incremental way, by simply disabling the component that directs local search according to the objective function and accepts first random change in the solution. This is illustrated in Figure 4.3, where dashed line between the operators indicate that only minor effort is needed

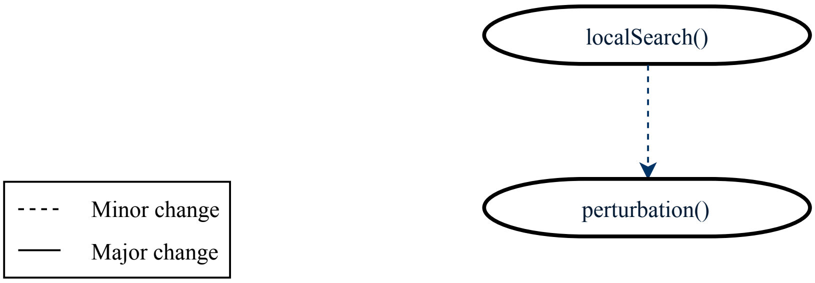


Figure 4.3: Perturbation operator evolution

to evolve local search to perturbation and convert the random restart local search into iterated local search.

4.5 Bottom up development methodology

As described in 4.2 and 4.3, there exists no evidence in the available theoretical study, nor in the empirical comparisons that more complex metaheuristics provide better performance and higher quality results. At best, the results indicate similar performance, and at worst, there exist empirical studies where simple techniques such as SA and ILS outperformed complex metaheuristics such as GA and ACO.

These results bring the question of whether it is reasonable to use complex metaheuristics like GA and ACO, when the same or better results could be attained using much simpler methods. Moreover, the literature indicates that performance-wise, selecting the right metaheuristic is less important than having efficient problem representation, neighbourhood definition and the operators based on problem-specific speedups. The time could be better spent by e.g. improving the local search procedure used in a simple metaheuristic such as ILS, than taking the time to implement and tune a complete GA.

The *bottom-up development methodology for the development of metaheuristics* is an attempt to resolve the above issues by providing a well-defined set of steps when solving a problem metaheuristically. It is a set of development principles with the following goals:

- Introduce a generally applicable, simple to use metaheuristics development methodology,
- Reduce the complexity of the development,
- Sistematise the existing established metaheuristics in a common framework for the development and exchange of operators,
- Reduce the development time, number of staff, computational resources and cost,
- Allow balance between implementation cost and the quality of results.

These goals are especially suited for situations when there is limited time for the algorithm implementation, limited staff and limited computational capacity. It is a great option to bring more order in metaheuristics implementation in both the practical applications as well as the academic study, since efficient development is important in both fields. The bottom-up development methodology uses the following design principles:

- utilising the component-based view of metaheuristics
- building *complete* components, in terms of each component being capable of returning a fully specified solution,
- building reusable and compatible components, that share the solution representation code,
- starting the development with the simplest components,
- incrementally adding more complexity,
- assembling available components into complete metaheuristics,
- stopping with adding more complexity when the results are sufficiently good.

Using the bottom-up approach, the development starts by implementing the first complete component: the simplest version of the initial solution generator. It continues by implementing the local search, and finally the perturbation. In each step, the metaheuristic into which the components are assembled can be selected based on the preferences, developer experience and specific features of the problem. Most importantly, stopping as soon as the good performance is achieved and achieve savings in the development time if the problem can be solved using simple techniques.

4.5.1 Assembling metaheuristics from components

As described in more detail in Chapter 3, each metaheuristic is a loop that performs a sequence of different operators (components), while remembering the best-found solution at all times. Initial solution generator, local search and perturbation are common components, however not all metaheuristics use all of them. Table 4.1 provides a breakdown of the components used per various established metaheuristics. In the table, a component is indicated as required by a bullet (•), and components that are not needed are indicated by a blank space (e.g. simulated annealing does not use the local search). Bullets also indicate components that can be shared across metaheuristics. For example, the local search used in GRASP can also be used in the iterated local search, without any modifications. Still, there are some metaheuristics that require specialised operators. These are denoted as “specific” in the table, e.g. GRASP and ACO impose specific requirements on the solution construction procedures. While GRASP solution construction could be used in other metaheuristics, the other direction does not hold. Likewise, tabu search uses a specific version of the local search that is not compatible with any other metaheuristic, unless a hybrid technique is being developed. Variable neighbourhood search requires more than one local search procedure, which is denoted as “multiple”. Finally, SA, ACO and GA do not require local search in their canonical definitions, however embedding it

Metaheuristic	Initial solution gen.	Local search	Perturbation
Pure random search	•		
Random restart local search	•	•	
GRASP	specific	•	
Iterated local search	•	•	•
Variable neighbourhood search	•	multiple	•
Tabu search	•	specific	
Simulated annealing	•	optional	•
Ant colony optimisation	specific	optional	
Genetic algorithm	•	optional	•

Table 4.1: Example of required components for various metaheuristics

in these metaheuristics is encouraged since it can significantly improve their performance. Such cases when local search can be added is denoted as "optional" in the table.

From Table 4.1, it is visible that the initial solution generation procedure is needed for all the considered techniques. Building the initial solution generator and only the local search procedure provides the possibility to assemble a wide range of metaheuristics. This is further detailed in Table 4.2, where a list of metaheuristics that can be built with several given combinations of parameters is provided. ACO and TS are in parentheses in the table since they require more sophisticated versions of either solution construction or the local search, therefore requiring more effort than building random restart local search and GRASP from the same components. As presented in the table, having the initial solution construction and local search allows greater flexibility in the metaheuristic selection than having only the initial solution construction and perturbation. With all three operators, all of the above techniques can be assembled. Typically these components need to be connected in a loop and adding some metaheuristic-specific code is always needed. In simpler metaheuristics such as ILS, this is simple and typically includes connecting the components according to the specification of the metaheuristic and keeping the best solution found so far. However in complex metaheuristics such as ACO and GA it requires more work.

4.5.2 Bottom-up workflow

The bottom-up workflow starts with the simplest and most basic components and gradually adds more complexity, in incremental improvements. After each incremental development, the current algorithm must be able to produce a complete solution to the problem. The solution must not be of high quality in the early stages, however it must be a complete assignment of values

Implemented components	**Possible metaheuristics**
Initial solution generation procedure	Pure random search
Initial solution generation procedure + Perturbation	Pure random search Simulated annealing
Initial solution generation procedure + Local search	Pure random search Random restart local search GRASP (Tabu search) (Ant colony optimisation)

Table 4.2: Metaheuristics that can be assembled with different components

to each planning variable. This allows a *rapid prototyping* approach, where early proposals of the system can be presented very quickly after starting the project.

After each incremental improvement, preliminary tuning, testing and performance checks need to be performed. If it is a commercial project, it is recommended to have the customer involved in each improvement, to allow early detection of issues. As soon as sufficient or near-sufficient quality of the devised algorithm is attained, the final detailed tuning needs to be done, and then the development can stop. The process can be viewed as a walk in a graph of possible metaheuristics where each metaheuristic is represented by a node. Iitially, the process starts with a random initial solution construction procedure, used in a pure random search. From there, the recommended path is to first add the local search, then develop perturbation. With these operators, the process moves from pure random search to random restart local search and then to the simplest complete metaheuristics: iterated local search, grasp and SA+LS.

Earliest developments

The development starts with the two most basic components:

- implementing the objective function and the constraints,
- implementing the initial solution generation procedure.

Both components are needed to allow complete functionality of creating new solutions and estimating the solution quality. Regarding the initial solution generator procedure, the simplest version of the generation procedure is needed. In most cases it is the random initialisation of all variables.

It is critical to implement the objective function as early as possible, especially in commercial projects that have numerous complex constraints. Having the implementation of the objective function at the start of the project helps with the implementation of all other components since it defines what the result of the algorithm will be. The objective function must be combined with the appropriate solution representation, and it must be as fast as possible. Therefore, the precise formulations of the objective function and constraints shape nearly all details in the optimiser, from the earliest stages where the most efficient solution representation and neighbourhood definition are selected to the final touches of various tuning procedures. Having the implementation of the objective function early allows testing the initial solutions, and it allows the developers to gain insight into the features of good solutions.

Another big advantage of having the objective function early is the fact that it is a precise formal definition of the goals of the project. By including the customer in the process when commercial projects are developed, discussing the results of the objective function and the way different solutions are compared significantly increases the clarity of any communication when features of the project are specified. Agreeing on the precise formulation for the constraints can be a difficult management and communication issue, especially if people were performing the task of the optimiser previously. The knowledge of people who were manually solving the optimisation problem can be characterised as *tacit knowledge*, something intuitively clear, however difficult to explain, formulate and implement in a software system.

Finally, having both the objective function and the initial solution generator defined and implemented as the first thing in the project allows a *rapid prototyping* approach to be used from the very start of the project. Solutions constructed by a random initialisation procedure will most likely be of very poor quality. Still, they are complete solutions, that can be used in other components of the system (e.g. the graphical user interface, and export modules that convert it into various human-readable reports). Most importantly, it allows the users of the product to be involved and track its evolution from the very start, this way allowing faster responses to any changes that might be needed.

Simple techniques

Given the initial solution construction procedure developed in the previous step, it is possible to very quickly develop search techniques, that while rudimentary, still have the most basic traits of metaheuristics. By simple iteration of the initial solution generation procedure in a loop, and keeping the best solution so far, it is possible to simply develop the *pure random search*, probably one of the simplest optimisation techniques. Pure random search, described in more detail in 3.5.1 generally does not perform well, however running it allows extensive tests of the existing code as well as an early presentation of the results in the rapid prototyping approach.

When the initial solution construction procedure is ready and tested, the development can continue by developing the next component. The next component to be devised can be either the local search or the perturbation operator. Despite the fact that local search is generally more difficult to implement than perturbation, unless the problem to be solved is very simple, it is recommended to implement the local search first. There are three reasons for this: (1) the local search, allows significantly better results earlier in the project, (2) local search is sufficient to build a wider array of metaheuristics, (3) perturbation is a trivial incremental step after the local search operator is built.

Still, if the developers opt for the development of the perturbation operator, it allows creating a first complete metaheuristic. By perturbing a solution in a loop and accepting the solutions according to the Metropolis criteria, it is possible to assemble the *simulated annealing* metaheuristic. Simulated annealing is a simple technique and in a narrow sense, simulated annealing itself can be considered a local search technique. In the Handbook of metaheuristics, Delahaye et al. state: *It is therefore necessary to see the annealing as a mechanism for approaching the global solution of a combinatorial optimization problem, to which it will be necessary to add a local search method allowing an optimum to be reached exactly* [370]. Therefore, it is common to complement it by the local search procedure to improve the results. In simple projects, however, even the canonical SA might be sufficient to reach good results. If this is empirically proven during testing, then the development can stop with SA.

Adding the local search

Given the

- initial solution construction procedure and a
- local search procedure,

it is possible to construct several search techniques, as seen in Table 4.2. The simplest one is the random restart local search. With some more development effort, it is possible to implement GRASP. If further development effort to devise metaheuristic-specific components is invested, two additional techniques—tabu search and ACO can be constructed.

Random restart local search can simply be assembled by running a local optimiser on the new random solution, and repeating the process in a loop. Random restart local search will generally provide results superior to the pure random search. Still, as described in Section 3.6.1, it is considered as a basic method due to the lack of focus. Local optima tend to be clustered in combinatorial optimisation problems, therefore random restarts as the basis for local search are too dispersed to provide good results.

In simple problems, especially if there is a large density of good solutions, this approach might be sufficient. If not, implementing algorithms with better control of the search process is recommended. *Simulated annealing with local search* is a popular extension of the simulated annealing, with the ability to provide better results than the canonical SA [370, 442]. Another possible direction is choosing either GRASP or tabu search and moving in the direction of these two metaheuristic-specific operators.

The *GRASP* metaheuristic can be implemented by adding an improvement to the initial solution construction procedure. It needs to be converted from a pure random initialisation to a randomised greedy algorithm. The recommended development course is to first develop a greedy construction algorithm, then add the randomisation element based on the restricted candidate lists, as specified in Section 3.6.2. It can be an attractive option if we have access to a greedy algorithm or the problem definition allows trivial development of a greedy construction procedure.

By extending the local search algorithm with the *tabu list* framework of tabu search, it is possible to improve random restart local search to tabu search. This requires that the basic local search procedure is extended to the $localSearchTabu(\mathbf{x}, \mathbf{x}_{best}, tabuList)$ procedure adjusted to the tabu search metaheuristic that has a description of the solutions that are prohibited as an argument. In a basic version of the metaheuristic, a simple list of prohibited solutions needs to be provided, however usually it is a specification of variables that must not be changed or classes of transformations that should not be done. This implies that that solution comparison support must be developed and that the corespondig classes of transformation can be checked. For simple problems this is trivial, however in e.g. scheduling it involves working with complex data structures and might require considerable and error-prone implementation effort. Additionally, the aspiration criteria also need to be implemented in the extended local search procedure to ensure that moves that improve the solution are not missed.

While GRASP and tabu search can be a good development direction, especially when the developer knows these techniques well, the bottom-up development methodology highlights that developing better low-level operators can be a better investment than developing sophisticated metaheuristics with complex operators and parameters that are difficult to tune. Developing the third standard problem-specific component, the *perturbation operator* is much simpler, since it can be a trivial simplification of the already existing local search codebase. Further, adding perturbation allows development of any metaheuristic, since this way both the intensification and diversification component is available. Therefore, implementing perturbation after local search it is the recommended direction in this development methodology since it is simple and allows assembling the developed components into virtually any metaheuristic.

Local search and perturbation

Given the

- initial solution construction procedure,
- local search operator,
- perturbation operator,

any established metaheuristic from Chapter 3 and any other metaheuristic that fits into the intensification and diversification frame [162] can be developed. Simpler metaheuristics can be assembled by simply connecting the developed operators in a loop. More complex metaheuristics require coding additional metaheuristic-specific elements. A good example of the simple technique is the iterated local search as well as the variable neighbourhood search. Conversely, the genetic algorithm uses all three operators as well, nevertheless the implementation of a genetic algorithm also requires implementing the problem-specific crossover operator and population handling.

The *iterated local search* metaheuristic can be built by perturbing the result of each local search in a loop, while keeping the best result so far. The most common acceptance criteria ("random walk" and "best") are trivial to implement, as discussed in 3.6.3. Along with simulated annealing complemented with local search and GRASP, it is the simplest metaheuristic in terms of development effort that is also capable of producing great results. The *variable neighbourhood search* can be implemented as a natural extension of ILS. To implement VNS, several different local search procedures are needed, so that each has a different neighbourhood, as detailed in 3.6.4.

Complex methods

The complex methods considered in this work are the genetic algorithm and ant colony optimisation. They require several components that are not used in other metaheuristics, in this way limiting the reusability. They are population methods, that require more memory, this way requiring more resources, which can be a problem when solving large problems and a large number of evaluations which is a problem when solving problems with a limited budget of evaluations. They can be difficult to tune.

The *genetic algorithm* can be developed based on initial solution construction and perturbation, however, extending it to a *memetic algorithm* by using the local search as well can improve performance (see Section 3.6.8). Along with these standard operators, genetic and memetic algorithms need population handling code and a selection operator, as well as the crossover operator. Since it uses numerous specific elements, it is not possible to create genetic and memetic algorithm by simple modifications of other metaheuristics.

The *ant colony optimisation* metaheuristic is most similar to GRASP. It is a multistart method based on randomised greedy solution construction. While the basic version uses only the solution construction procedure, it is recommended to extend it using local search. The metaheuristic does not use the perturbation operator. Using the bottom-up approach, ACO can be built by extending the greedy algorithm, where solutions are built based on the heuristic information and pheromone traces of artificial ants. This requires a pheromone information data structure to be implemented, usually as an appropriate graph. The highly specific solution construction can be complex to tune since it depends on several parameters such as the pheromone update function, algorithm variant, the range of values of the objective function, evaporation.

4.5.3 Achieving high performance

In this methodology, it is highlighted several times that each prototype should be tested to check if sufficient performance is achieved. The sufficient performance can be anything from providing any solution at all, to outperforming people who were doing the task manually, to outperforming worlds top solvers. An important goal of this methodology is to eliminate the unneeded complexity and unneeded development effort and highlighting the potential of simple metaheuristics to provide great results. This does not mean that applying the proposed methodology cannot produce top quality solutions. By allowing the developer to move in the direction of higher complexity when the current results are not satisfactory, it allows building arbitrarily complex methods, including exotic metaheuristics that are not explicitly mentioned in this work, as long as they fit the I&D framework.

Another important recommendation is that once all three basic components have been built, the development of the low-level operators is favoured instead of using more complex metaheuristics. As an illustration, let us consider a scheduling problem that involves scheduling the staff in a set of 1-hour tasks, so that some employees always must be grouped in the same team. Also, let us assume that we have a very early version of the ILS metaheuristic with a basic local search operator that moves one person at a time across different tasks. What would be better—to invest time to build a more complex metaheuristic such as ACO, or invest time to improve the local search operators. From the author's experience, as well as some other researchers, the answer is almost always—"improve the operators" [585].

With enough time, any metaheuristic will eventually arrange the employees in a way that does not split the teams across different tasks. Nevertheless, the above version of the local search is inefficient. Moving a single person at a time, without considering that some need to be grouped will have a local neighbourhood in which groups will frequently be split, especially if a group is big. A simple improvement to our ILS algorithm would be modifying the solution representation so that the groups are viewed as a single unit. Then the local search and all other operators would always move these groups together. This way, the algorithm would be able to focus on the difficult parts of the problem and it will not have to waste time discovering trivial problem specifics.

While any metaheuristic will eventually adapt the solution to respect the constraints, metaheuristics can be slow in such discoveries. Instead of burdening the algorithm with the task of discovering the knowledge that can be simply encoded into the operators, it is much better to add this knowledge to the operators. Metaheuristics are very general problem solvers, therefore more complex search procedure will typically add fewer benefits than adding more problem-specific features to the operators. In the above scheduling problem, using ACO instead of ILS will have the same problem as long as the solution representation is inefficient and groups are split.

4.5.4 Agile development

The methodology proposed in this work is more compatible with contemporary software development methodologies such as *agile software development* and *rapid application development* (RAD). It is more consistent with the shorter release cycle, iterative product development and gradual product evolution, which are the basic ideas in the agile software development methodologies. It is suitable for the prototyping approach in which prototypes of an optimisation algorithm are continuously improved in cooperation with the users or customers. Finally, complexity is a common issue in implementations of metaheuristics, therefore the proposed methodology puts a great emphasis on keeping the implementation as simple as possible.

These practices can allow greater flexibility in the development and project management, and embrace the fact that in the real world, software specification is often subject to change. Instead of viewing the implementation of the algorithm as a single big task, it facilitates the component-based view that promotes splitting a big task into smaller ones. Finally, it avoids the frequent practice of implementing complex metaheuristics, when the same task could be done with much simpler.

Comparison with traditional top-down and waterfall approach

Nearly all literature in metaheuristics implies the waterfall approach, in which there is an implementation phase and production phase, and the implementation effort is nearly always ignored, mostly implicitly, and in rare cases, also explicitly [557]. This corresponds to the waterfall software development methodology in which a problem is first completely specified, then the implementation is done in predefined steps, so that the next step proceeds only after completing the previous. This approach works well in e.g. construction engineering, however it is criticised as not sufficiently flexible for software development. While this is acceptable in academic study focused on e.g. improving the process of tuning, such an approach ignores the problems described in this work: unjustified complexity in the algorithms which promotes slow and expensive development.

Consider e.g. the guidelines for implementing ACO published in 2004 in the excellent book [124] by Marco Dorigo (redacted):

> 1. *Represent the problem in the form of sets of components and transitions or by means of a weighted graph, on which ants build solutions.*
> 2. *Define appropriately the meaning of the pheromone trails. This is a crucial step in the implementation of an ACO algorithm and, often, a good definition of the pheromone trails is not a trivial task*

and typically requires insight into the problem to be solved.

3. *Define appropriately the heuristic preference for each decision that an ant has to take while constructing a solution.*
4. *If possible, implement an efficient local search algorithm for the problem to be solved.*
5. *Choose a specific ACO algorithm and apply it to the problem being solved, taking the previous aspects into account.*
6. *Tune the parameters of the ACO algorithm.*

The above workflow is an example of a waterfall process of implementing a software project, common in literature about metaheuristics. Since the publication of this book, the waterfall approach has been widely abandoned in software engineering, in favour of more agile methods . Understandably, given that the book is about the ant colony optimisation algorithm, it assumes that the developer will implement ACO.

The drawbacks of the approach presented above is the fact that a complete solution to a problem can be produced only after step 4 (or 5 if local search is not used). Before that, several complex and problem-specific steps must be done, which indicates that this might not be a quick process. It also implies that the development of components such as graphical user interface either must wait or must be done with limited set of artificial solutions until steps 4 or 5 are done. If local search is used, this approach ignores the fact that experiments with simpler methods can be done as soon as we have developed the local search. The literature is still widely segmented into different metaheuristics. Therefore almost all methodological recommendations in the literature are similar—based on the assumption that the method to be implemented is already selected and then based on the fact that the problem is fully specified and that the problem definition will never change.

Contrary to the waterfall model above, using the methodology proposed in this work, the user can be presented with early prototypes as soon as the initial solution generating procedure is finished. Since random initialisation is trivial, this can be done very quickly. The graphical user interface as well as e.g. export to OpenOffice format can be implemented in parallel as both teams can be provided with numerous complete solution objects. The integrations team can plan the web service endpoints for the system and test all layers of the application. No component is implemented unless necessary and there is no need to wait for very long development cycles to see a result of the improvement that is currently being worked on. There is no complexity and development time invested unless justified by performance reasons. Finally, the process assumes that a customer might change the problem definition during the process. By attempting to detect parts of the system that need to be changed as soon as possible, using the prototyping approach, it allows implementing changes while the system is yet incomplete and there was not much effort spent into all the final details.

4.6 Conclusions and future research directions

This work is an effort to reduce the dichotomy between academia and software industry, and reduce the gap between the output of a successful academic research project and the needs of practitioners. Current research is still in great deal segregated in independent units focused on individual metaheuristics and improving efficiency. Further, the current research typically does not devote much attention to the issues of high effort that is required to achieve good results in practice. Conversely, the software industry is typically not interested in the best possible results and highly complex (and expensive) methods. Instead, it needs the balance between quality and the cost of development. The development methodology proposed in this work can provide this balance, and great savings in the generally expensive process of metaheuristics development. Finally, the methodology is more compatible with modern agile software development methodologies, allowing small incremental changes and a quicker response to changing requirements. Instead of building complex method-focused systems, the methodology emphasises minimum complexity that allows satisfying performance, and a component-based view, where individual components can be assembled and reassembled in different ways.

The next chapter gives a description of the application of this methodology to solving three difficult problems:

- Workforce scheduling in small inbound call centres,
- Improving the carsharing reservation service, and
- Improving the profitability of carsharing services by variable trip pricing.

In all three cases, adding more complexity stopped when sufficiently good results were achieved. The results show that the ILS metaheuristic was sufficiently flexible in solving all three problems in a satisfactory way. These results are consistent with findings of [320], which indicated that even the very simple metaheuristics could be highly efficient problem solvers.

A very interesting future work direction is resiliency to changing requirements, which is dealt with in a limited way in this work. Development of techniques that increase the robustness of metaheuristics to adding constraints and modifications in solution representation would help increase adoption of metaheuristics by the practitioners. Additionally, development of such techniques can provide valuable insight into the behaviour of various metaheuristics and the interactions between the metaheuristic and solution representation, which would add great value from the academic point of view.

Chapter 5

Applications

This Chapter describes three case-studies in which the bottom-up development methodology proposed in Chapter 4 is applied to build metaheuristics for solving three difficult problems:

- Call centre workforce scheduling problem in small and medium call centres,
- Carsharing reservation system service quality and profit optimisation,
- Dynamic pricing optimisation problem in one-way carsharing.

Each of the above problems is a discrete, constrained, deterministic optimisation problem. The first problem, call centre workforce scheduling problem is a highly constrained scheduling problem, while in the second two, the constraints are few and very simple. Contrary to the constraints, the first problem has a simple objective function that can be calculated very quickly, therefore allowing short solving times of typically 30 seconds per problem. The second two, and especially the last problem have a slow objective function. These problems are solved using an approach where each solution is evaluated by simulation. For large problems, this process can last several minutes, therefore limiting the total number of possible evaluations. The first and the last problem are single-objective problems, whereas the second problem is a multiobjective problem.

The first problem, *call centre workforce scheduling problem* is a complex scheduling problem, consisting of a large number of constraints. The problem is solved using two metaheuristics: GRASP and ILS, and the results are compared. The work was presented on the EvoApplications, European Conference on the Applications of Evolutionary and bio-inspired Computation held in Porto, Portugal 30 March - 1 April 2016. The detailed report is published in the conference proceedings available in [395].

The second two problems are related to *carsharing*. Carsharing is a service that consists of a fleet of vehicles available at various locations across the service area that can be used by a large group of the service members and is typically charged by minute or by the hour [586]. The carsharing reservation system improvement problem solved using the ILS metaheuristic is a multiobjective problem in which the profit of the provider and the service quality provided need to be balanced. Along with the successful solution of the problem, the research also resulted with a first technique that can provide long-term reservations in one-way free-floating carsharing systems. The simulated results significantly outperform the reservation times available in the contemporary carsharing providers. The results are published in the Transportation Research Part C: Emerging Technologies journal [182].

In the third problem, the carsharing system is improved by introducing variable pricing for customer trips in a one-way station based carsharing systems. To the best of author's knowledge, this is the first application of ILS metaheuristic to a problem with a limited budget of evaluations. Variable pricing is an attractive option to guide customer behaviour used in various industries, including transportation, however their use in carsharing has been limited. It was hypothesised that by lowering the prices for trips that help the system and raising the prices for the trips that are not favourable under current conditions, it is possible to improve the system performance. These improvements could include better fleet balance, as well as higher total profit, nevertheless this potential of pricing was not proven in rigorous research of one-way carsharing, nor the practice. The simulation results indicate that significant profit and balance improvements can be achieved using this approach. For example, the proposed metaheuristic was able to turn a simulated Lisbon carsharing provider struggling with losses into a profitable service that produces more than 1000 € of profit per day. The results of the research are published in the journal Transportation Research Part B: Methodological [181].

By applying the development methodology on three diverse problems, it is shown that the methodology is flexible enough to allow solving difficult problems, including those with slow evaluation function, while still resulting in simple metaheuristics. The remainder of this Chapter first brings the results of applying the bottom-up development methodology on the call centre scheduling problem. Then, in Section 5.3 the carsharing reservation problem is described in detail. The chapter concludes with the description of the ILS metaheuristic for the one-way carsharing variable pricing problem.

5.1 Workforce Scheduling in Inbound Customer Call Centres [1]

Nearly all types of human organisations occasionally face scheduling problems. Staff scheduling is a classic operations research problem that consists of assigning a set of employees to a set of working times, subject to various constraints and with the goal of finding the schedule of

[1] This section is based on the previously published paper: "Workforce Scheduling in Inbound Customer Call Centres with a Case Study", in the proceedings of Evo* 2016: Applications of Evolutionary Computation, 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, Part of the Lecture Notes in Computer Science book series (LNCS, volume 9597). Copyright is held by Springer International Publishing Switzerland 2016.

the best quality. Such problems are $\mathcal{NP} - hard$ [199, 329, 587] in their very simplest forms, which imposes tractability issues and renders them difficult to solve efficiently. Contact centre staff scheduling is an example of this type of a problem [339, 587, 588, 589].

Call centres are an instrument many companies use for communication with their clients. They are commonly used to provide technical support, perform sales, handle various customer inquiries etc. They typically consist of a centralised pool of trained staff members called *agents*. When a customer dials the number of a company, if there are available agents in the centre, her call is answered immediately. However, if all staff members are busy at that moment, the customer will have to wait in the queue until an agent becomes available. Long waiting times can cause customer dissatisfaction and it is critical to keep the waiting times as low as possible. This relates to the typical goal of a call centre, achieving high *service levels*. On the other hand, to keep the operating cost reasonable, hiring too much staff is also undesired.

In a traditional call centre, staff members are communicating exclusively via telephone. With the increasing popularity of the Internet, many companies added support for other means of contact, such as e-mail or chat, extending call centres into their contemporary generalisation called *contact centres*. Contact centres support several communications channels, and typically include e-mail, online chat and telefax aside from the usual phone. Based on the number of supported channels, centres can be *multi–channel* or *single–channel*. Centres that are answering, but never actively initiating communication are called *inbound* contact centres. Conversely, centres that are only initiating communication are called *outbound* contact centres. If both answering and calling are performed, the service is classified as a *mixed* contact centre. With regard to the staff skills, a contact centre can be *single–skilled* or *multi–skilled*. In a single skilled centre, all of the employees have the same training and theoretically, provide a homogeneous service level, independent of the agent. In a multi–skilled centre, there are various profiles of agents, depending on their skill sets. Good workforce schedule needs to organise existing staff into schedules that keep the service level as high as possible while taking legal and organisational constraints, as well as personnel preferences into account. Such staff schedules will not have too much staff members available during low intensity hours since it is expensive to have idle agents at work. Conversely, good schedules will not have deficit of agents at peak hours to keep the customer satisfaction high. Ideally, all of the incoming calls will be answered immediately and staff utilisation rates will always be high.

In this work, a call centre scheduling algorithm is proposed. It is suited for the needs of a small to moderately sized single–skill inbound call centres. The system is based on a flexible constraint management framework that allows easy addition of new company–specific constraints and two robust, scalable metaheuristics. The system is used in two steps. First, forecasting is performed, based on the call history in order to estimate the distribution of calls during the next scheduling period. The forecast demand is used to calculate the distribution of staff during time, that ideally meets the demand and the service levels during the scheduling period. After the forecasting and staffing curve estimation is done, schedules are generated using an optimisation algorithm that searches for the schedule that best fits the forecasted calls. Related work proposes a diverse range of techniques such as dynamic programming, linear, quadratic and mixed integer programming and relaxations of the linear programs [325, 327, 332, 333, 334, 335, 336, 337, 338]. Several heuristic methods are also proposed [590]. However, the aforementioned work is focused on the optimisation part and less on the constraints imposed by the organisation. As an exception, a hybrid heuristic approach with several techniques, including an algorithm inspired by simulated annealing is described in [339] and applied to a real–world problem. Constraint handling appears to be performed through the means of the objective function, therefore defining them as ranked soft constraints that can be violated in final solutions. In [327, 332], an integer program is solved using an iterative cutting plane method with evaluations based on simulation while in [591] the the problem is framed as a mixed integer stochastic program. A detailed literature review on staff scheduling, including specifics of call centres, is available in [325].

Using the incremental metaheuristics building methodology proposed in previous Chapter, two metaheuristics were built: GRASP and iterated local search (ILS). These two metaheuristics were, to the best of the author's knowledge, never successfully applied to this type of problems. As compared to related work, this problem is a *highly constrained* example of a *real world* problem. A flexible constraint handling system is a highly prominent feature of the system. Moreover, the implemented approach deals with the constraints differently than any other proposed method the author is aware of. Instead of implementing them in the objective function as in [339], devised a rule based assurance system is devised in each component of the algorithm. In that manner, hard constraints are guaranteed to be satisfied in all of the solutions throughout the execution.

As noted in [592], a noticeable gap was observed between the research output from academia and the needed expertise that can be directly utilised by the industry. Solving real problems is difficult since it necessitates modelling and handling various kinds of constraints which are usually simplified in academic problems. This work aims to contribute to bringing the worlds of academia and practical implementations closer together.

5.1.1 Problem Description

The call centre workforce scheduling problem (*CCWSP*) is a scheduling problem whose goal is to maximise the *service level* while satisfying all the imposed constraints. The service level is typically defined as a combined measure consisting of (*i*) a percentage of answered calls (more is better) and (*ii*) percentage of dropped calls (less is better) during a time period. In the developed algorithm, schedules are typically generated one month ahead, with adjustable target service levels. Times are quantized, with the minimum quantum duration given as a parameter and 30 minutes is used in this case study. If needed, finer granularity could be used. This naturally means that the optimisation problem might be more complex.

The problem is defined as an ordered triplet:

$$CCWSP = (S, s_d(t), C),$$

that consists of a set of staff members S, the desired staff number during time $s_d(t)$, calculated based on the demand, and a set of constraints C. The set of staff members is a fixed set of workers at the contact centre, with their permanent workplace designated for each staff member. Seating and workplace assignment is therefore not a part of the optimisation problem. Each staff member is defined by its identifier in the system and the relevant staff data is stored in a suitable database.

The ideal staff distribution $s_d(t) : \mathbb{N} \to \mathbb{N}$ is a function which defines a minimum number of employees needed at time t, in order to achieve the desired service level. As the frequency of calls varies throughout the days of the month, staff number that is needed to handle such demand will also vary. In this case study, based on the preliminary interviews with a call centre in Zagreb, Croatia the demand peaks are usually experienced afternoon, and they are most prominent on Mondays, as seen on Figure 5.1. Much fewer calls are placed during weekends, requiring noticeably less workforce.

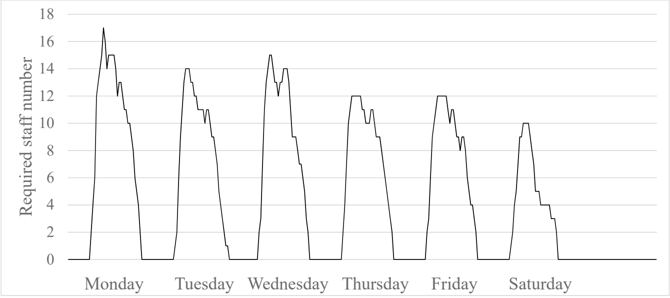


Figure 5.1: An example staff distribution curve for one week (Mon-Sun) [395]

The set of constraints is derived from the labour law in Croatia, company specific policies and regulations. Furthermore, it's easy to notice that many of the constraints were defined out of a desire to keep the employees satisfied, by giving them as much flexibility as currently possible. Most of the constraints are related to a single worker but there are some, that are dealing with an aggregate number of working staff during weekends.

Contact centre constraints include:

- Contact centre open hours, e.g. centre is open 7:00–21:00 during working days, 07:00–17:00 during Saturdays and closed Sundays and holidays.
- Shifts intervals (e.g. morning shift arrivals from 07:00 - 10:00).

- Maximum allowed daily work time (legally defined to be 8 hours).
- Minimum daily break, defined as the time between consecutive presences at work, (currently 12 hours as defined in the Croatian labour law).

Basic staff constraints include:

- Suitable arrival times (some workers can only work in the morning while some can work anytime – during the morning, afternoon and night shift).
- Suitable working days (e.g. some workers work Mon-Sat, some Mon-Fri).
- Duration of the work time (can be adjusted for each day of the week and holidays, e.g. worker can work 8 hours Mon-Fri and 7 hours on Saturdays).
- Prearranged absences (vacations, free days, sick leaves).
- Prearranged presences (a manager might arrange some activities in advance).

Some company specific constraints include:

- Workers arrive to work at the same hour throughout Mon-Fri.
- The weekend schedule is independent of the work day schedule.
- Simple night shift rotation rules are supported for the night shift staff.
- The night shift order is defined as an input, with a different staff member assigned to the night shift in each period. This defines night shifts as a specific case of prearranged presences.
- Employees working on Saturdays need to have at least two working Saturdays, if needed (during 5 Saturday months), an employee can work three Saturdays, but only if he or she didn't work three Saturdays during the previous month.
- Shift work (soft constraint) – employees working in both morning and afternoon shifts need to have at least 5 days in each, to ensure they are entitled to their monthly shift work bonus.
- To ensure equal service levels during weekends, number of staff members during Saturdays needs to be roughly equal.
- Unpopular shifts constraint: an employee cannot be scheduled in unpopular shifts (e.g. 14:00) during two consecutive weeks.
- Highly unpopular shifts constraint: a list of short time periods that are considered the most unpopular among staff. A staff member can be assigned to work during the unpopular shift only one week per month at maximum. In this work, the most unpopular arrival times were the mid–day periods at 10:00, 10:30, 11:00 and the late afternoon at 14:00.

Solutions that satisfy all of the above constraints are called *feasible solutions*. Violation of any constraint other than the shift work constraint renders the schedule unsuitable for use and such solutions are therefore considered *infeasible*. While these constraints are highly specific for the case study call centre, the implementation of the constraint management system is modular and parametrised, in order to enable easy integration into similar call centres and further customisation. The input to the scheduling system is an XML document which includes information about employees, ideal distribution and the constraints for the scheduling period. While such files can be edited directly, end users are accessing the system through a suitable graphical user interface.

5.1.2 Methodology Overview

Similarly to related approaches, the process of scheduling the contact centre workforce is performed in three stages: (1) forecasting, (2) staffing and (3) scheduling. Unlike in [335], scheduling and rostering is both done in the same stage and performed by the metaheuristic. In the remainder of this Section, the first two stages will be briefly described, while the scheduling algorithm is elaborated in detail in the following Section. The typical workflow starts by the contact centre manager telling their staff to enter their preferences and absence days in the scheduling system through the staff version of the user interface. Call centre administrators then make further adjustments such as setting up the target service level and constraints, using the administrator version of the user interface. In this stage, the S and C component of the contact centre scheduling problem are defined.

The ideal staff distribution $s_d(t)$, however, isn't known in advance and depends on the number of calls received during the upcoming scheduling period. The call forecasting is based on the inbound call records from the activity history. Currently, apply a simple forecasting model is applied, where it is assumed that the calls for the current month will match the inbound calls during the same month previous year. To further refine the forecasting and account for the differences among days of the week, the forecasting system automatically aligns working days for the forecasted month with the working days of the reference month. In that way, working days, weekends and fixed non working days are aligned. For example, call forecast for April 2016 based on April 2015 call history, will be shifted two days ahead as April 1, 2015 is Wednesday and April 1 2016 is Friday.

The greatest variation in the volume of calls is dependent on the day of the week and the season of the year. Aside from already mentioned weekend–work day load intensity differences, late December peaks and low–intensity summer weeks are common in this work and this pattern repeats every year. Therefore, even such a simple model provided a good estimate of the future calls.

After forecasting, the staffing is performed. It consists of determining the ideal staff distribution for the upcoming period. Since the assumption is that the call centre has a single queue and is single–skilled, there is an analytical solution to calculating the number of necessary staff. This calculation is computed using the Erlang-C [588] formula. An example of an ideal staff distribution curve calculated by the system for an example real–world week is shown on Figure 5.1. As discussed earlier, it is visible that Monday has a considerable peak but also that only a few staff members are needed to successfully handle the weekend calls. The user interface allows further manual staff distribution curve adjustments as specific events such as promotions of new services might change the demand.

After defining an ideal staff distribution, it is needed to find the workforce schedule that matches this distribution as closely as possible. Due to the complex constraints and tractability issues, devising a direct algorithm for the problem is difficult. Therefore, it was decided to use a metaheuristic approach.

The bottom-up development methodology proposed in this work was used to first build a random restart search based on the initial solution generator. After this, the local search operator was used to produce the GRASP metaheuristic. Surprisingly for the author, the call centre experts that were consulted for this work considered even this simple metaheuristic as working well enough. For research purposes, after a simple modification component was built, some experiments were performed with the iterated local search algorithm.

Constraint assurance and Prototyping Approach

While general ideas about the constraints were agreed upon before the project started, they were not detailed enough to allow simple translation into scheduling rules that can be implemented in a software product. The goal was to produce a scheduling system that is able to respond to a plethora of possible events in a highly sophisticated way — in essence, acting as a replacement for the human that was in charge of scheduling. Agreeing on the definition of the necessary constraints for the workforce scheduling system is a difficult management and communication issue. The knowledge of the persons who were manually scheduling the workforce can be characterised as *tacit knowledge*, something that is intuitively clear, but difficult to explain, formalise and automate. Therefore, a prototyping approach was chosen. Prototypes of a scheduling system were presented on regular meetings with the call centre management and further improvements were discussed and agreed upon. Since the definition of the problem varied through time, a flexible constraint assurance system was needed.

The author decided to use a system in which all of the constraints except one are satisfied throughout the search process. The only exception is the soft shift–work constraint that was added later and is a combinatorial optimisation problem on its own, in current version partially solved by postprocessing. The devised system is a modular object oriented set of classes and interfaces written in Java and based on the *observer design pattern*. Such approach, that was to the best of the author's knowledge first used in [593] is a natural way to implement complex interactions of various constraints as scheduling events are occurring. In this approach, each staff member is a subject, to which an arbitrary set of constraint objects (observers) can be added. After scheduling an agent to a certain time slot, all of the observer objects (constraints) assigned to him/her are notified. After receiving the scheduling event notification, each observer updates the list of suitable times for that staff member to reflect the effects of each schedule change.

Using the described architecture, the lists of allowed time intervals are constantly maintained by the observers. In each search step, the algorithm can easily determine which quanta are suitable to schedule a staff member by performing a simple querying of the feasible times list.

For example, the unpopular times constraint removes other unpopular times throughout the previous and next week for that staff member if it receives a notification about scheduling in an unpopular period. Removed time quanta are "invisible" to the search algorithm in the remainder of the run, thus further reducing the size of the search space. If a new type of a constraint is needed, it is easy to implement – it is enough to provide an implementation of the observer interface and simply plug it in the rest of the system.

5.1.3 Scheduling Algorithm

In this Chapter, a *GRASP* (Greedy Randomised Adaptive Search Procedure) [134] and iterated local search [594] metaheuristics to solve *CCWSP* are elaborated. The GRASP metaheuristic uses a solution construction component and local search to build different locally optimal solutions while keeping a record of the best so far. *Iterated local search* (ILS) is a simple but effective metaheuristic that systematically explores close proximities of known good solutions using the local search operator while avoiding being stuck in local optima by the means of a *perturbation operator*.

The complete solution algorithm works in three stages: (1) preprocessing, (2) work day scheduling, (3) time scheduling. In the first step, preprocessing is done to find misconfigurations in the input problem and handle prearranged presences and absences. Working days are then split into *statically* and *variably* scheduled days. For statically scheduled days, the schedule is known in advance. Such days are not subject to optimisation and are not modified by the algorithm.

After the preprocessing, weekend working days are chosen using a direct, feasibility–preserving heuristic that determines which Saturdays and Sundays a person is working. When all the working days are known, the solution representation data structures are built. Since staff always comes at the same time Monday – Friday, in most cases Monday can represent the remaining four working days as well. Such preprocessing significantly reduces the dimension of the search space. After building the solution representation, the working days are known, however, the schedule is still empty as variable working times are yet to be determined by the optimisation algorithm. It is clear that scheduling a staff member at the time t during the day d has consequences for the work times during the remainder of the month due to a complex set of imposed constraints. To ensure that complete schedules are feasible, the robust and extensible constraint assurance system described in the previous Section is used. As already mentioned, in the current implementation, throughout the search process all except the shift work constraint is always satisfied.

Objective function

An ideal timetable will have exactly $s_d(t)$ staff members available at time t. As ideal staffing curves frequently have short spikes and irregularities (Fig. 1) and the typical working time is around 8 hours, it's generally impossible to achieve absolute accordance to the ideal curve. The objective function evaluates how close the real staff distribution is to the ideal one.

In the proposed objective function, a penalty is defined for each candidate schedule as a sum of penalties for all time quanta in the schedule:

$$p = \sum_{t=0}^{t_{max}} (s_d(t) - s(t))^2,$$

where t_{max} is the last quantum in the scheduling period and $s(t)$ is the number of staff in the generated candidate schedule, as opposed to the goal number of staff members $s_d(t)$ during time t.

Note that the number of missing staff is squared. The rationale for such objective function is the decision to emphasise bigger deviations from the desired curve. For example, let us compare a schedule with one man–hour missing during 5 hours, one per each hour and a schedule with 5 man–hours missing during one hour. For that day, they both have a total of five man–hours missing. However, a lack of one person is hardly noticeable in the call centre while lack of five people has a significant impact on the service quality of a small centre with 30 employees. Squaring the number of missing persons helps emphasise this effect. A bigger difference from the ideal staff distribution means bigger penalty and lower solution quality. The best possible solution would perfectly follow the ideal distribution and have a penalty of zero.

Random solution generator

The random solution generator uses the described constraint assurance system to produce an initial solution in which all of the working times are chosen randomly from a set of feasible times. The scheduling is done sequentially, for all of the working days in the schedule, for each staff member that works on that day. As each scheduling decision is made, the constraints handling system is keeping updated information about which working times are suitable for the employee. As the schedule is having more working times determined, less and less time intervals are suitable, since many constraints contradict each-other. In the case of a failed constraint setup for an employee, that employee's schedule might be empty due to a constellation of conflicting constraints which caused the feasibility checker to determine that no working hours are suitable for the staff member. In such cases, users are advised to carefully inspect the setup for that person or ask for help from the support team.

Local search and perturbation operators

Local search is a simple greedy operator that tests if moving a staff member earlier or later during the day helps make a better schedule. It can operate only on feasible times and does nothing if the current time is the only one that's feasible. Local search is performed for each staff member, for each day in the search space. The local search operator has one parameter: number of passes through the entire schedule. If the number of passes is set to 2, the local search algorithm will be performed twice.

To promote unbiased discovery of good solutions, the order in which local search is applied on staff members is randomised. In that manner, unfair schedules are avoided, since initial solutions tend to have a different distribution than the goal staff distribution. For that reason, some favouritism was discovered as the agents that were first to be optimised had a tendency to be scheduled to unpopular afternoon times when the demand is, in general, higher.

The perturbation operator introduces random changes in the solution, which might both improve or decrease the solution quality. Perturbation operator has one parameter: the percentage of staff members for which the schedules will be modified. For each staff member, and for each time in the dynamic schedule, a random choice of a feasible work time is performed.

Random restart search

After implementing an initial solution generator, a trivial random restart search metaheuristic can be easily implemented by running it in a loop. It is frequently used as a baseline to check initial versions of more advanced algorithms. The more advanced algorithms should, at their very least, be able to significantly outperform random restart search. As the end condition, a timeout is used, currently fixed to 30 seconds, as described in Section 6.1.

GRASP and Iterated Local Search

An initial solution generator and the local search are sufficient to assemble an implementation of a GRASP metaheuristic [134, 343, 364, 595]. It starts by building a random initial feasible solution, on which the local search operator is applied. After that, the procedure is repeated, all the time keeping a record of the best solution found so far. The algorithm pseudocode is given in Algorithm 1. As the end condition, a timeout is used, currently fixed to 30 seconds, as described in Section 5.1.4.

Adding the perturbation operator to initial solution generator and local search makes it possible to assemble the iterated local search metaheuristic. It uses the perturbation operator instead of generating entirely new solutions to escape local optima. As with the previous two approaches, a timeout of 30 seconds is used. The algorithm pseudocode is given in Algorithm 2. In this implementation, perturbation is always applied to the last local optimum, therefore, the author selected the *random walk* acceptance criterion for the iterated local search [343, 364, 594].

5.1.4 Results

The proposed metaheuristics are implemented in Java 1.8 programming language. During the experimental evaluation, a series of experiments was run on a problem instance that is considered to be an appropriate representative of an usual scheduling problem in a small call centre. The problem has 32 staff members under a typical workload for a one month period. In that example, the call centre is open 7:00–21:00 during working days, 7:00–20:00 on Saturdays and closed Sundays and holidays. Most staff members have 7 hours work time Mon–Fri and 5 hours work time during Saturdays. Total of 13 staff members is using their absence days, 58 days in total, with most of them absent for a week (6 days). There are no public holidays in this month. Time quantum duration is 30 minutes. This example is representative for most other examples that were encountered during the development.

All experiments were run on a computer equipped with a 2.4 GHz Intel Core i7-4700HQ processor and 16 GB of RAM. The algorithm was running using Java 1.8.0, subversion 25-b18 runtime environment. First, the initial tuning of the termination criteria was performed. Then, a detailed parameter tuning for the local search intensity and perturbation parameters (ILS only) was done with the best–found termination run time. Finally, ILS and GRASP performance was compared with the baseline random restart search and with each other's performance.

Termination condition

It was decided to use the allowed execution time as the termination condition. This termination condition allows a consistent user experience and near–real–time way of using the system. Before doing the tuning process, the author wanted to decide on the allowed time and performed an initial tuning step to investigate the influence of execution time on the performance. Typically, when running a metaheuristic, there is an initial period of fast improvement that soon starts to slow down. After a certain point in time, the improvement rate becomes slow to none and the task of termination time tuning was to determine the time which will allow the algorithm to converge but not waste the effort if improvement rates drop too low. Initial experiments have shown that increasing the GRASP running time from 1 to 10 seconds gives a quality increase of only

6% and that 30 seconds runs give approximately the same quality as the 10s runs. Therefore, the termination criterion of 30 seconds is selected as a tradeoff between comfortable waiting time and scalability.

Parameter tuning

Most metaheuristics have various parameters that need to be set to a certain value. While a quick setup with parameters chosen by the developer's intuition might work well, parameter tuning based on experimental evaluation is an essential part of the metaheuristic implementation. It ensures that the parameters are adapted to the problems representative of those being solved by the final version of the algorithm and, therefore, provide the best possible performance in a production setting [542].

During the GRASP tuning, experimental evaluation consisted of determining the best local search intensity. Running the local search multiple times after each initial solution is constructed might improve the solution. However, focusing on local search too much will cause the search procedure to get stuck in local optima too frequently and not exploring the search space thoroughly enough. A total of 8 experimental setups was evaluated ranging from 1 to 500 iterations, with 30 algorithm runs for each of them, as displayed in Table 5.1. The best average performance was achieved with only 5 iterations, proving that the local search operator is able to reach the local optimum in a rather low number of passes through the solution.

Iterated local search tuning consists of finding a balance among the local search and the perturbation operator. The intensity of the perturbation should be high enough to ensure that the local search operator doesn't return back to the initial solution, yet not too high, to prevent the algorithm from degrading to random restart local search [594]. For each of the 4 different perturbation configurations, 8 different local search intensities are evaluated, leading to a total of 40 different setups. The average performance for 30 runs of each setup is shown in Table 1. Similarly to the GRASP local search tuning, the results show that the local search converges to the local optimum after a relatively low number of passes. For the perturbation intensity equal to zero, the algorithm degrades to infinite local search around the initial solution, leading to clearly suboptimal solutions. Changing 20% of the solutions (the percentage of staff members affected with perturbation) with 10 iterations of LS has produced the best results. Setting the perturbation to higher values produces worse solutions, but only slightly, since the LS operator is able to reach good solutions even with very high perturbation rates. The reason for such effect might be the fact that a large number of restrictive constraints prevent the operator from changing the solution to a great extent.

Table 5.1: Metahruristic tuning results [395]

GRASP tuning	
LS passes	Average penalty
1	14802
5	**14582**
10	14760
20	14951
50	15188
100	15278
200	15716
500	16263

ILS tuning				
1-5	perturbation intensity			
2-5 LS passes	0.0	0.2	0.5	1.0
1	18675	14158	14284	14528
5	18293	13975	14188	14188
10	17944	**13952**	14094	14098
20	18277	13957	14209	14181
50	18916	14108	14180	14149
100	19699	14100	14223	14253
200	19193	14020	14331	14398
500	18972	14349	14580	14530

Performance comparison

The best performing configurations of GRASP and ILS are also compared with the baseline random restart search as well as with each other. The results obtained in 30 experimental runs are shown in Table 2, including the median penalty (relevant for the statistical test). Additionally, average, minimum, maximum and sample standard deviation of the penalty is provided to give a better illustration of the distributions. From the table, it is clear that both GRASP and ILS are producing better results than the baseline random restart search and that ILS is better than GRASP. This is formally verified using Wilcoxon–Mann–Whitney test to check the H0 hypothesis that distribution functions of the algorithm performances for GRASP and ILS are the same as for random restarts. Both tests resulted in p–values below $1 \cdot 10^{-10}$. Using the same statistical test, it is shown that performance differences of GRASP and ILS are statistically significant ($U = 18$, $p = 1.773 \cdot 10^{-10}$).

In order to test how representative was the example used for tuning, the performance on a larger set of input examples was tested: April, May, July and November of 2014 and January 2015. Note that these examples weren't available at the time of initial tuning. Overall, while still being better, the median performance of ILS is only 2% better than in the case of GRASP. On a case–by–case basis, ILS had better median results in 4 out of 5 examined problem instances, while GRASP was more successful in one.

The described system has been successfully applied to several difficult problem instances representative for real-wold call centre scheduling. Creating such schedules by hand is highly

Table 5.2: Random restart, GRASP and ILS performance comparison [395]

	Penalty				
Metaheuristic	Avg.	Median	Min	Max	Std. dev
Rand. restart	30072	30267	27384	31340	876
GRASP	14563	14558	14180	14902	173
ILS	**13952**	**13937**	**13548**	**14456**	232

labour intensive. Further, when developing schedules by hand even if satisfying the constraints, it is difficult to assess the impact on the service quality. The devised system produces schedules that are ready for use in less than a minute, therefore a software system based on the developed algorithm has a potential to bring great time savings.

5.1.5 Future work and Conclusion

This Chapter presents applying two metaheuristics: GRASP and ILS to solve a call centre scheduling problem. To ensure suitability for small enterprises, minimalism, simplicity and, therefore, reduced development effort were main implementation strategies. The algorithm is attractive for cases when complex and expensive suites featuring support for multi–channel multi–skilled centres is not needed. Despite the simplistic design goals, the devised problem definition is still quite broad and applicable to a wide range of call centres. The devised constraint handling system provides a rich set of implemented rules and in case some of the requirements are not covered by existing rules, it's architecture provides clearly defined interfaces for the addition of new ones.

It is also demonstrated that the proposed bottom–up design approach produced surprisingly good results even in the early stages of implementation. While the original plan was to connect the components in some of the more complex metaheuristics such as ant colony optimisation or genetic algorithm, even the initial GRASP version produced results that were satisfactory. The author believes that using a simple algorithm in the aforementioned manner led to great savings in software development costs. An essential component of this approach is an independent constraint assurance system that facilitated rapid prototyping. Since defining the constraints during a scheduling project is a difficult communication issue, prototyping approach that gives users a chance to see the constraints in action after short development cycles significantly simplified the requirement analysis and functional specification for the project.

The implemented system has a great potential to help the contact centre management to save time on scheduling and focus on other important tasks. With an addition of a suitable graphical user interface, call centres could have an integrated and easy to use system that could allow them base their schedules on real data. Break scheduling could further improve service quality and as a major feature, multi–channel and multi–skilled contact centres could also be supported. These types of contact centres are especially interesting research area since there is no known analytical way to calculate their ideal staff distribution and existing approaches are based on heuristics or a simulation to evaluate the schedule quality.

5.2 Carsharing

Carsharing is a type of mobility service that provides short-term car rental [182, 586, 596]. Such services provide a fleet of cars distributed across the coverage area that can be used by the service members. Unlike traditional rent-a-car, the typical rental durations are short and charged by the minute or the hour. They are typically privately owned and marketed as a membership-based service. Service price includes the costs of fuel, cleaning, insurance, maintennance and management of each vehicle [182, 586, 597].

In modern carsharing systems, user interaction is commonly done using a smartphone application or a web site of the provider. In more rare cases, users could request a vehicle using a phone or other suitable channels. From the users' point of view, using the service is performed in the following sequence of steps.

1. User searches for the closest available vehicles.
2. User checks if the nearby vehicles found in the above step are suitable for the trip, based on personal preferences and trip requirements such as car brand, number of seats and the distance of the location where the free car is parked.
3. If a suitable vehicle is found, the user books the vehicle.
4. User goes to the vehicle, unlocks it using his membership smartcard or the mobile app and starts the trip.
5. The user is driving, and the service is charged e.g. by minutes or hours used.
6. After the trip is finished, the user parks the vehicle in a suitable location, locks the vehicle and checks out.
7. The returned vehicle again appears as available to all other users that can then book it.

Carsharing can provide the flexibility and accessibility of a private vehicle, without costs and responsibilities of owning a car. Using carsharing can be an alternative to both private vehicle ownership and public transport [586, 598]. To policymakers, carsharing systems are interesting due to their potential to reduce pollutant emissions and reduce the need for parking spaces and costly expansions of the public transport service coverage [599, 600].

5.2.1 Similar services and discriminating features

Carsharing is a part of recent trends in which various *shared mobility* services are becoming more popular and have wider availability, especially in urban areas [597]. Such services include bikesharing, ridesharing and others. *Bikesharing* involves a fleet of bicycles shared by the service members, and used in a similar way cars are used in carsharing. *Ridesharing* has a different operating model, in which rides of users with similar origins, destinations and start times are grouped in the same vehicle.

Despite the similarities in the general idea, carsharing is different from the traditional *rent-a-car*, which provides longer rental periods, usually charged by day or week. Traditional car rentals often have complicated procedures for taking and returning the vehicle, and carsharing has very simple and quick self check-in and check-out. The locations of the rent-a-car service centres, clustered around big commercial centres and terminals as compared to carsharing cars distributed across the city further indicate that carsharing and rent-a-car do not share the customer base, nor can they support equal trip purposes. Rent-a-car is well suited for a multi-day trip between several cities e.g. during a tourist visit to a foreign country, while a typical carsharing trip could be a trip to a popular restaurant in the city that could be an attractive option both for visitors as well as city residents.

5.2.2 Classifications of carsharing services

Carsharing systems can be classified into round-trip carsharing and *one-way carsharing* systems. The *round-trip* carsharing systems are the traditional type of carsharing, where each vehicle must be returned to the initial location after use. This type of carsharing limits the possible purposes of such trips, as the requirement to return the vehicle to the same location where it was taken limits users to round-trips, such as shopping. This type of carsharing is not suitable for travellers that have a considerable period of activity at their destination, or the travellers who do not need the return trip, e.g. a trip to the cinema or the daily commute [596, 601]. In the one-way carsharing, trip can end at any location, not necessarily the same one where the trip started. This type of carsharing is much more flexible for the users who can use it for all purposes, including the daily commute to and from work. While the greater flexibility is favoured by users, providing this type of carsharing is much more complex and expensive [181, 182, 597, 601].

A notable issue in one-way carsharing is the *vehicle stock imbalance* problem. The demand for vehicles across the service area varies [596, 601, 602]. Since users are freely moving the vehicles, due to varying demand, some areas might have more arrivals, while in others the number of departures might be larger. During the morning rush hour, it is likely that areas around the business districts will have more incoming than outgoing trips due to the daily commuters arriving to work [602]. This varying demand can lead to the accumulation of excessive number of vehicles in locations where the demand is low, and a lack of vehicles where there is a great demand for the service. The imbalance problem can in some cases even cause the lack of the parking space in the areas where there is a lot of incoming trips [181].

Based on the allowed set of locations where a vehicle can be returned after use, carsharing can be classified into *station-based* and *free-floating* systems. In station-based carsharing, the provider defines a set of specific locations (stations), and the vehicles can be returned only to the stations of the provider. Usually such providers purchase and reserve private parking places or garages in several locations across the city. Conversely, the users of free-floating carsharing can leave vehicles in any legal parking area in the city. Such providers typically arrange a business deal with the city authorities to allow the carsharing vehicles city-wide parking in the public parking places.

Depending on the type of propulsion system used in the cars, there are specifics related to the vehicle use and management required in carsharing. A carsharing fleet can be based on *internal combustion engine vehicles*, *electric vehicles*, *hybrid vehicles* and some combination of the three [603]. The internal combustion engine vehicles have a long range that can be travelled without refueling, and the refueling process is quick, however they are notorious for their environmental impact. Electrical vehicles are considered to be a greener alternative to the combustion engine cars, however they are more expensive, and have a shorter range [603, 604]. Further, charging the batteries of electric cars can take a considerable time during which the car cannot be used, unless battery swap techniques are used. Due to these specifics, carharing systems that use electrical vehicles have a higher management complexity [605, 606, 607, 608].

5.2.3 Historical overview

The earliest known carsharing project was started in 1948 in Switzerland. The project was called "Sefage", a short for "Selbstfahrergemeinschaft", which could be translated from German as "self drive community" [586, 597, 609, 610]. Established in the city of Zurich, it can be described as a cooperative, whose members were sharing access to vehicles as a group of friends, without a strict agreement. This community was founded mostly motivated by economic reasons. At the time, cars were expensive, and owning one was considered a luxury [597, 611]. The project was successfuly functioning during 50 years.

In 1951, French engineer Jacques d'Welles discussed the potential negative effects of privately owned cars and their increasing numbers in cities. To solve the problem of excessive privately owned vehicles, he proposed a development of a small electric city car. He further suggested that these cars could be shared by a large number of members, to further reduce the total number of needed vehicles [597, 612]. The proposed concept was strikingly similar to the modern electrical vehicle projects such as the Bluecity carsharing company in London that provides an entirely electric fleet of Bolloré Bluecar vehicles [613, 614].

Similar *public car* initiatives, mostly motivated by the well being of the citizens, and later as an effort to reduce pollution were started in 1970s and 1980s. A notable example in Montpellier, France, called “Procotip”, started in 1971 was the first known one-way carsharing system that had 35 cars and 19 stations. The members were buying tokens to be inserted in the “Tipmetre” devices that resembled parking meters. The project was active during two years, and in 1973, the project filed for bankruptcy. It failed mostly due to technological issues and a lack of suitable control systems.

A similar project called “Witkar” (Dutch for White car or white coach) was a one-way electrical vehicle carsharing project in Amsterdam, started in 1973 [597, 615]. Such an endeavour sounds cutting-edge even at the time of writing of this thesis, almost 50 years later. The service was providing access to 35 custom electrical vehicles in five stations in the centre of the city. The vehicles were an innovative custom designed electrical mini-cars or coaches, developed with the goal to be small, environmentally friendly and suitable for the city commute, with room for two persons, 1.76 m long, 1.42 m wide and 1.95 m high [597, 616].

The long term goal was to achieve centralised control over the vehicles with minimal labour. For this purpose, the cooperative purchased a Digital Equipment PDP-11 minicomputer and developed software support for vehicle release, checks of available parking space at the destination. The system was also used to track the battery status in vehicles, advise on battery replacement if possible and control the vehicle stock balance. In cases of vehicle deficiency or excess vehicles at some stations, the system would advise the staff about the issue, that could then organise trips to rebalance the vehicles. The system was also used for payments: it would check the user creditworthiness and to avoid the need for handling cash, it could directly charge the users’ bank account by regularly exchanging the PDP computer tapes with the Amsterdam savings Bank where all members needed to have a bank account [616]. The service was active during 12 years and had a total of four thousand users [597, 617]. It was discontinued due to the lack of govenment support and technological issues. The service had problems with vehicle stock imbalance due to allowing one way trips. While both the Witkar and Procotip services closed, they achieved great accomplishments in advancing the urban transportation. They were the pioneering projects in carsharing, that were ahead of their time in numerous ways. The available car technology, computers and software, as well as transportation research were simply not advanced enough at at the time to support successful services of such complexity, and both projects also lacked the sufficient support from the government and the cities in which they were tested [586].

During the 1970s and 1980s, several additional early projects were also started in Sweden (cities of Lund, Örebro, Gothenburg), United Kingdom (Suffolk), Switzerland (Zurich), Germany (Berlin) [586, 618], and in the North America. Development of carsharing was slow and steady during this period. In 1991, the European Car Sharing Association (CSA) was founded, with the goal to support the carsharing development and lobbying across Europe. In 1997, CSA had around 70 operators as members [586].

During the last two decades, a notable increase in the number of providers and users has been observed. Carsharing turned from dispersed and small experimental initiatives to a widely accepted transportation service. These developments have been especially fast in Asia and North America. In North America, the number of carsharing users increased from 210 thousand members in 2007 to 1.9 million members in the year 2017 [597, 618, 619]. Globally, as of 2016, carsharing operators were present in 46 countries on six continents, with more than 15 million members and over 150.000 shared vehicles in total [620].

5.2.4 Carsharing technology

The development of information technology has been crucial for the development and the increasing popularity of this mode of transportation. The ubiquity of smartphones, the flexibility of developing fully customised applications for smartphones, vehicle access secured by contactless smart cards and mobile phones, precise geolocational services such as GPS are the basic "ingredients" used by virtually all modern carsharing providers. These technologies greatly simplified the business of running a carsharing project and increased the simplicity of use and the security and reliability of the operation.

Another critical technological element that is routinely used by most carsharing services are specialised software packages that provide support for the service operation and automate processes such as booking, billing, vehicle tracking, fuel and battery level monitoring and others, this way integrating the fundamental technological components such as GPS and mobile phone apps into a comprehensive suite for running a carshairng system. Along with the basic features such software packages can include more advanced logistical and optimisation solutions that can automate tasks such as vehicle stock balancing, relocation movements optimisation and others [586, 597, 611].

5.2.5 Potential benefits of carsharing

Carsharing is associated with a number of potential benefits, both for the users as well as the environment, and some studies also report the wider social benefits [603]. Transportation is causing notable negative effects, including the greenhouse gas emission, air pollution, and traffic congestions. Studies indicate that carsharing has a potential to reduce the greenhouse gas emissions [597, 621, 622, 623], reduce the vehicle ownership rates [598, 623, 624], reduce the car use and total kilometers driven [599, 622, 624, 625]. In addition to these benefits, up to 35% of the carsharing users either sold, delayed buying or considered selling their car [623, 626, 627]. In [624] it is demonstrated that carsharing users also have a lower drive-alone rates. Finally, carsharing is ineresting to the policymakers due to its potential to complement the public transport network and expand its reach by filling the gaps in the existing public transport service coverage [597, 623, 628].

The most important advantage to the users is affordable access to the benefits of having a car without all costs and responsibilities of owning one, especially when the user does not travel a lot. It is very convenient and cost effective for occasional use [599]. There exist indicators that lower vehicle usage also encourages carsharing users to use healthy modes such as walking and cycling [622].

5.2.6 Commercial carsharing providers

According to the carsharing market report for 2019, authored by the Canadian consultancy firm Movmi [629] and the CSA Carsharing association [630], there exist only two carsharing providers with global reach: Zipcar [631] and Share Now [632]. ZipCar is a station-based carsharing provider headquartered in Boston, USA, and providing services in a total of 384 cities. It was acquired by Avis Budget Group in 2019. Share Now [632] provides free-floating carsharing in a total of 30 cities. It is a joint venture that merged Car2go (owned by Daimler AG) and DriveNow (owned by BMW). Before the merge, Car2go was the top provider in terms of the number of cities where it is available.

In Croatia, as of 2019, there is only one carsharing provider called Spin City [633], available in Zagreb. It is a free-floating service with a fleet consisting of 20 combustion engine cars and 10 electric vehicles. In addition to the usual carsharing services, it also allows the use of prepaid packages and more traditional rent-a-car services, charged daily.

5.3 Optimising long-term vehicle reservations in one-way free-floating carsharing systems [2]

Despite numerous benefits reported in 5.2.5, carsharing is still a service that received a wider adoption only decades ago and is still evolving. Better technical support and innovative transportation solutions such as the most recent appearance of one-way systems increased the applicability of this service [597]. Despite all progress, an important obstacle to the broader adoption is the fact that the service is still more difficult to access than for example a taxi. Aside from being dispersed at attractive locations around the city to allow walk-ins, the taxi service typically offers the dial-a-ride, e-hail, and booking services which add additional value and increase the suitability of the service for different purposes.

[2] This section is based on the paper: “Long-term vehicle reservations in one-way free-floating carsharing systems: A variable quality of service model”, published in the journal “Transportation Research Part C: Emerging Technologies”, Volume 98, January 2019, pages 298-322”, copyright 2018 Elsevier Ltd.

A possible way to increase availability and user satisfaction in one-way carsharing systems could be providing vehicle reservations. Reservations are available in a wide range of services and industries: reserving a table at a restaurant, seats in a theatre or booking hotel rooms are nowadays ubiquitous everyday actions. Reservations are available in other transportation services as well: virtually all of the air traffic is reserved ahead, and most taxi providers allow their users to reserve a ride [634, 635, 636, 637]. Reservations can give the providers useful information, such as daily, weekly and seasonal demand patterns, and the way users respond to various campaigns. Knowing the demand ahead helps these services to plan their operations and organize the resources to improve efficiency. Therefore, the operators commonly encourage users to perform reservations as soon as possible. Pricing incentives are a frequently used way to achieve early user response. As a notable example, booking a hotel room or a flight just one day ahead is almost always much more expensive than doing it some months in advance.

Providing vehicle reservations in carsharing can be a highly challenging issue though, and has hardly been addressed in the literature. The topic of using resource reservation as a management strategy in carsharing has been mainly explored for parking at the destination when there is a shortage of parking spaces [602, 638]. Unlike the airline and hospitality industries, where the reserved resources are under complete control by the provider, this is not the case in carsharing. The shared fleet movements are dynamic and difficult to predict, due to varying demand. For a carsharing service provider, knowing reservations a few days ahead, i.e., where a vehicle is going to be picked-up, does not help much in running the enterprise as relying on daily user trips is not enough to provide the guarantee that a vehicle will be available at the reserved location and time. Instead, some other mechanism needs to be used to support the reservation service and ensure that the user will have the reserved vehicle at the place and time he/she desires.

A simple and effective strategy that can be used to enforce reservations is *vehicle locking*. In this approach, the user selects a vehicle close to the desired location and the departure time. After this, the vehicle is considered locked and inaccessible for use by any other member, similarly to a waiter in a restaurant putting a “reserved” label on a table. A prominent drawback of such approach is that it lowers the vehicle utilization rates and the revenue produced by the locked vehicle. This is such a notable issue that many one-way carsharing providers do not have reservation services at all, or if they do, they offer it under highly restrictive conditions. For example, the global operators Car2Go and ZipCar allow reservations for one-way trips, but only up to 15 or 30 min before the trip start [639, 640]. Some other services allow longer reservations, however, charge for them by the minute [641, 642]. The utility of such service is therefore highly limited as reserving a vehicle for a trip to the airport a week ahead or a trip to work tomorrow morning is not possible or at best, is expensive. These restrictions substantially decrease the quality of service being provided by a mode that is supposed to serve a higher share of demand in the future.

Relocation operations are vehicle movements initiated by the service provider and performed by a team of employees. So far, relocations have been used mainly to solve the vehicle stock imbalance problem, both in the station based and free-floating carsharing systems. Relocation trips do not generate revenue and represent a cost for the company due to the fuel and staff expenses. However, research has shown that such investment can lead to higher overall profits by providing the ability to fulfill more demand. It is possible to find several optimization and simulation methods dedicated to this problem [643, 644, 645, 646, 647, 648, 649].

To the best of the author's knowledge, no research has been done to demonstrate the drawbacks of the vehicle locking method for providing carsharing reservations, nor in providing a more efficient alternative that can cope with identified disadvantages. In this work, an innovative reservation enforcement method named Relocations-Based Reservations (R-BR) that complements vehicle locking with relocations operations in a free-floating one-way carsharing system is proposed. It is hypothesised that this approach will allow longer reservation times while keeping the vehicle utilization rates and revenues reasonably high.

Methodologically, this work is based on a simulation-optimization approach. A custom microsimulation environment is built to investigate the user-operator interactions under different conditions related to reservations. A carsharing company might not support reservations at all or might use various strategies to ensure that reserved vehicles will be at the requested location at the required time. Companies might also sometimes reject reservations and users will not use the service unless an available car is close enough to reach it by walking. The developed model has similar properties to others that have been proposed in the literature to study the management of carsharing systems such as [643, 650, 651, 652].

The reservation quality of service (*QoS*) is defined using two parameters: (1) time in advance allowed for making a reservation, denoted h, and (2) the radius around the trip origin, denoted r, where the reserved vehicle is guaranteed to be available at the time of the client departure. It is assumed that users would like to be able to reserve a car anytime they want, therefore longer h means better user satisfaction. Conversely, it is also assumed that users would like to walk the shortest possible distance to the reserved vehicle [653]. More formally, to improve user satisfaction, it is desirable to maximize h and minimize r.

Applying an equal setup everywhere in the service area might not be optimal. Tactically increasing the service quality in certain zones of the city and decreasing it in others has the potential to improve the profitability of the service and increase the accepted demand, without impacting the service quality too much. Based on this idea, the Variable Reservation Service Quality Problem (VRSQP) is defined: given a set of zones in a city, with the possibility to choose a separate service quality level in each zone, the goal is to find the best set of (radius, time ahead) parameters in order to maximize the objective function (denoted Z). The objective function is defined as a weighted sum of individual goals: profit (maximized), satisfied demand (maximized), allowed time between the moment of reservation and trip start (maximized) and the radius around the user (minimized). These goals can be contradictory in some cases, which makes the VRSQP a multi-objective optimization problem. By choosing the appropriate weight for each of the four individual goals, it is possible to model the operator preferences: some businesses might be entirely profit-oriented and set to ignore all other goals, others might prefer a more balanced approach where profit is not improved if it causes large drops in service quality.

Choosing a setup of the geographically varying pairs of r and h is a complex problem. The author implemented an Iterated Local Search (ILS) metaheuristic [594, 654] in a simulation-based optimization approach for finding good and realistic solutions to the VRSQP. In this setup, the simulator acts as an evaluator for the variable service quality layouts proposed by the ILS algorithm. Based on the evaluation from the simulator, the algorithm creates increasingly better solutions and discards those that produced bad results in the simulation.

The methodology is applied on several problem instances: two extreme hypothetical cities (small town and large major city) and a case study of Lisbon Municipality, Portugal, with four different demand levels. Two key experiments are performed:

1. Comparing the vehicle locking and the R-BR method under a constant *QoS* in the entire service area,
2. The R-BR method is further optimized under a variable *QoS* using the ILS.

5.3.1 The relocations-based reservations (R-BR) method

Let us imagine that a user calls at 17:00 and wants to reserve a vehicle to be available the same day at 21:00 at a specific location of the city. In the vehicle locking approach, the operator searches for the closest vehicle to the desired location. If the closest vehicle is within the acceptable radius (r) from the location, the reservation is accepted, otherwise, it is rejected. If the reservation is accepted, the current closest vehicle is marked as locked and in that way, reserved for the user. In the example in the Figure 5.2, the closest vehicle that was found will be locked at 17:00 and will remain in its location until the desired departure time (21:00) when the user picks it up. Notice that this reservation process would be the same had the user searched a specific vehicle himself by using a smartphone or a laptop with internet access.

Using the Relocations-Based Reservations (R-BR) method, when a client makes a reservation, no action is taken immediately. At that moment, the reservation is checked for feasibility, as there exists the *QoS* limit of accepting reservations no more than h minutes ahead. After the reservation is accepted, all vehicles in the network continue to be available as if no reservation has taken place until the *response time moment*, denoted as t_a. Response time moment is the time before the desired departure at which the system starts processing the reservation and activates the relocations enforcement mechanism. At that point a decision needs to be made: lock

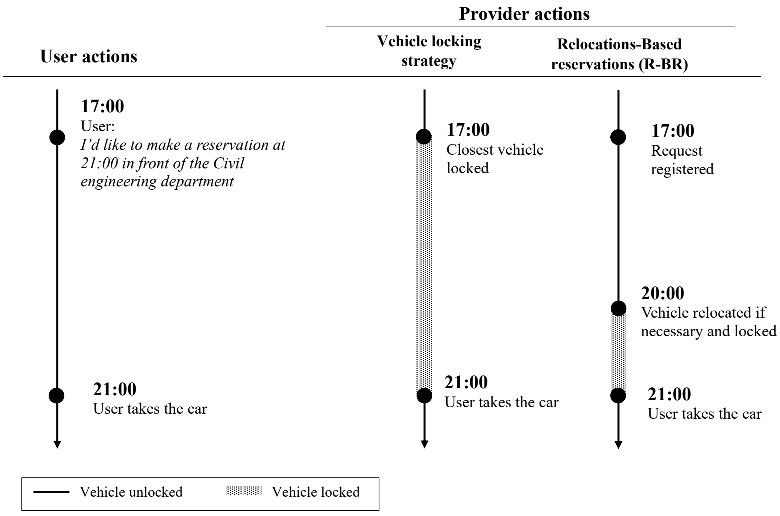


Figure 5.2: Vehicle locking and relocation strategies for allowing reservations [182]

some nearby vehicle or use a relocation movement. This decision is made based on the location of the closest available vehicle to the client trip origin. If the nearest vehicle is within the acceptable *QoS* radius (denoted as *r*), that vehicle is locked until the user takes it at the desired departure time.

When the vehicle is relocated, it will be locked until the user takes it. If there are no vehicles available for relocation a taxi must be provided to the client since he/she was expecting a vehicle. In the example shown in Figure 5.2, the vehicle locking system caused the vehicle to stay idle for 4 h whilst using the R-BR method, the car would be idle much shorter (only up to 1 h in the example). This approach is sketched in Figure 5.3. Note that the values of *r* and *h* can be set globally, equal for the entire service area or they can vary depending on the origin zone as in the VRSQP problem.

An important aspect that needs to be decided is how long before the reservation does the system need to respond. If the response time is too long, there is the risk of having low vehicle utilization rates, similar to the ones obtained with vehicle locking, if it is too short, the system risks having unreliable service where delays can happen. The author proposes propose that this parameter should be set in such a way that the system still has enough time for a relocation, even under the most pessimistic traffic conditions for the particular case-study city. While a more realistic value could be used in the function of actual traffic conditions, in this work the parameters are set to the most conservative estimates due to the fact that such forecasting could be unreliable. Last-minute cancellations of accepted reservations would undermine the user trust, and therefore selected higher reliability is selected instead of slightly better profit. A key issue when applying R-BR method is choosing a right balance of *QoS* parameters *r* and *h* and other performance indicators such as profit and satisfied demand. Any change of these will affect users who in general want to be able to reserve as early as possible and want their cars to

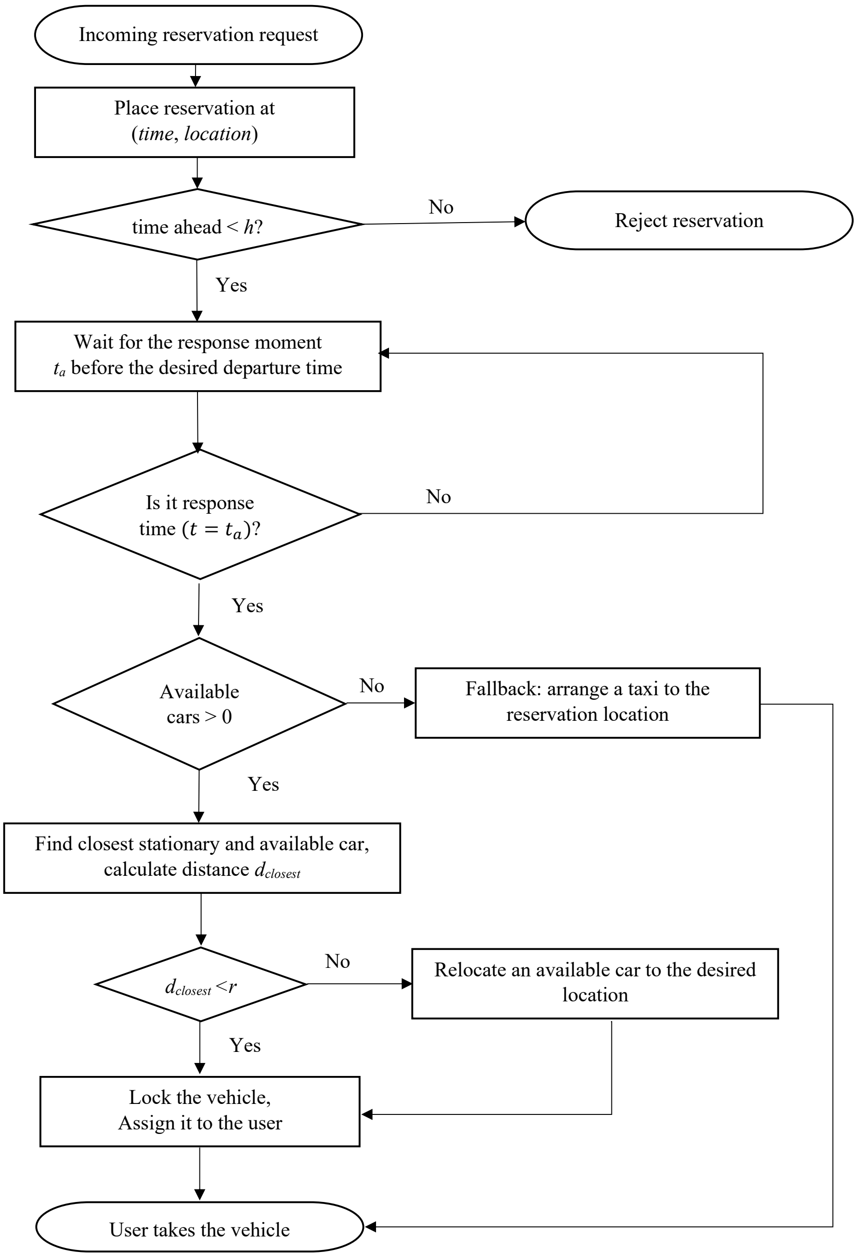


Figure 5.3: The proposed relocations based reservations enforcement strategy [182]

be as close to them as possible. Achieving a high quality of service can require more effort from the provider, and it has the potential to cause drops in profitability. In this work, two algorithms are proposed to help set these parameters:

1. a simple *QoS*-sweep algorithm to choose the best global service quality (equal in the entire service area regardless of the origin location) and
2. an ILS metaheuristic to choose these parameters when they can vary, depending on the origin zone.

Vehicle stock balancing

The simulation environment developed in this work supports two different types of relocations:

1. Reservation support relocation movements,
2. Balancing relocation movements.

The key application of relocations in this work is the first one: relocations are used to bring a vehicle to the location of a reservation if there are no nearby cars. The second type is a traditional application in carsharing, used to improve the vehicle stock balance and increase the probability that a vehicle will be close to the average user, including the users doing walk-ins [643]. Even though the focus of this work is on the first type, in the simulator, both can be used independently or complementary. In both cases, the relocation decisions are based on the current state of the simulated fleet (reserved, available and occupied vehicles), and the demand forecast data in the rectangular grid across the city, during several time periods.

While the first type of movements has a reservation support purpose, they can nevertheless be used to improve balance. Consider the situation where a type 1 movement is needed because there are no vehicles close to the user for a reserved a ride. A relocation movement will be performed to move a vehicle to the departure location. Depending on the choice of the car to relocate, it could increase or decrease system balance. Always choosing the closest car is the myopic cheapest move, however, it does not take the vehicle stock balance into account and has the potential to worsen it. Conversely, performing relocation trips that are best from the balancing point of view could lead to a large number of long and expensive relocation trips. For the reservation support movements, the middle-ground approach is used, where cars are relocated from the closest zone with a vehicle stock surplus. In the case where there are no surplus zones in the system, the closest available car is relocated.

For the second type, a simple strategy is implemented, where a number of balancing trips is periodically dispatched. Balancing effort intensity is parametrized by two parameters: (1) balancing trips per period, denoted b_n and (2) balancing period duration, denoted b_p. All the balancing trips are started at the same time at the beginning of each balancing period. Increasing the number of balancing trips per period and shortening the period (increasing the dispatch frequency) will improve the balance, however, the costs of operating these relocations will increase.

The balancing algorithm uses the forecasted demand during several days and the city area divided into a rectangular grid to determine which zones have a surplus of vehicles and which have a deficit. Depending on the severity of the deficit/surplus, the zones are prioritized into suppliers and demanders, and relocation movements from suppliers to demanders are produced. Cost of relocation movements is calculated based on the cost per minute driven for relocation trips, denoted C_r .

Note that faithfully modeling different balancing relocation strategies is a separate, complex issue that is still under research on its own [643, 644, 645, 646]. Considering that a highly realistic simulation of relocations is not the goal of this work, and to ensure faster execution of the model, the details related to advanced optimization techniques for selecting relocation movements are intentionally omitted, as well as the details of running the appropriately sized workforce. it is assumed that at any given moment, a staff member can immediately be available in any part of the city to start the relocation if needed. This is a simplification of the real systems which often work with their own staff team who require some time to reach the vehicles to be relocated and have a varying number of available staff throughout the day. However, an another assumption in this Chapter is that the relocations are performed by out-sourced people who do each relocation operation one service at a time and are paid by minute of relocation rides. When other staff payment models are used, some approximations are needed. For example, an operator might observe that approximately 35% of their relocation costs are spent traveling to the relocation movement origin and take that into account when calculating the value of the C_r parameter.

5.3.2 Variable reservation quality of service (QoS)

Varying the service parameters such as prices according to the local conditions is widely used in transport services [181, 655, 656, 657]. To further tailor the one-way carsharing operation to the demand, aiming at increasing the profit of the company while allowing reservations and keeping the service quality high, a variable QoS model across a city is added to the simulation. Given varying trip patterns across the zones of a city, tailoring the reservation parameters has the potential to improve the system efficiency. In the variable service model, the service area is divided into N zones, with individual QoS parameter values in each trip origin zone:

$$QoS_i = (r_i, h_i), \tag{5.1}$$

where r_i is the maximum allowed distance of the reserved vehicle (radius around the user) in the i-th zone and h_i is the maximum allowed reservation time ahead of the trip start in the same zone.

The Variable Reservation Service Quality Problem (VRSQP) is defined in general terms as the problem of finding the optimal set of *QoS* parameters for the zones in the city, for which an objective function that describes the operator preferences is maximized. While the profit of a carsharing company is a good foundation for comparing the solution quality from the perspective of the operator, it is clearly not enough to guarantee that service of good quality is being provided to the travelers. Large radius r might lead to savings in relocation trips and higher profits, nevertheless, the users prefer it to be as small as possible. For many users, walking half a kilometer to reach the vehicle makes little sense, especially if their trip is going to be short. Likewise, users prefer to be able to place reservations with as few restrictions as possible, therefore the longer the allowed reservation time, the better the service quality offered to the clients. Finally, the satisfied demand is the fourth importat factor which determines the solution quality, higher demand acceptance levels mean that a better user coverage was achieved.

Therefore the problem of finding the optimal reservation parameters is defined as a combination of all or some of the following individual objectives (1) maximizing the profit, (2) maximizing the reservation times, (3) minimizing the radius around the user where the reserved vehicle will be available, and (4) maximizing the satisfied demand. All of those can be combined into a scalar function as follows:

$$Max(Z) = w_P \frac{P - P_{min}}{P_{max} - P_{min}} + w_h \frac{\overline{h}}{h_{max}} + w_r \left(1 - \frac{\overline{r}}{r_{max}}\right) + w_d \frac{d_{sat}}{d_{tot}} \tag{5.2}$$

where P is the profit for a given solution, P_{min} and P_{max} are the lower and upper profit bound estimates, $\overline{h}$ is the average reservation time limit across all zones of the city, h_{max} is the maximum allowed reservation time, $\overline{r}$ is the average radius across all zones of the city, r_{max} is the maximum radius, d_{sat} is the number of satisfied trips, d_{tot} is the total demand (maximum potential number of carsharing trips) and w_P, w_h, w_r and w_d are the weight factors determining the relative priority of each function component during optimization. Since the radius is to be minimized, in the objective function it is converted to a maximization objective by subtracting $\frac{\overline{r}}{r_{max}}$ from 1.

The operator has complete freedom to choose the relative importance of each performance indicator and even to entirely exclude them from consideration. For example, a profit-oriented business might optimize only profit ($w_P = 1, w_h = w_r = w_d = 0$), not caring at all about satisfied demand or cars being close to the users. Some other operator might be in a middle of a marketing campaign during which they want to increase satisfied demand and brand exposure, even at the cost of slightly lower profit. A third provider might choose a balanced approach where service quality drops are acceptable, but only if they are justified by a high profit increase.

Note that the components of the objective function are normalized to an interval of [0, 1]. Therefore, if the weight factors are chosen in a way that their sum is equal to one and the profit limit estimates are correct, the entire objective function will be normalized to that interval as well, thus being possible to be represented by a percentage. The bound values h_{max}, r_{max} and d_{tot} are known in advance as they are the input to the optimization. However, the upper and lower bounds of the profit, P_{max} and P_{min} are not known in advance and need to be estimated.

In the simulation model, a fixed trip database is used and it is assumed that the input demand is constant. Giving the optimizer too much freedom when choosing the service quality would certainly break this, e.g., frequently placing a car 3 km away from the user who reserved it would give a very bad impression and discourage users from using the service, this way also lowering the demand and invalidating the profit calculation based on the constant demand assumption. To achieve ceteris paribus conditions in the optimization process, especially related to the input demand volume, the simulation framework provides two ways to control the algorithm's freedom to modify the solution: (1) hard *QoS* limits h_{max} , r_{max} , h_{min}, r_{min} and (2) soft objective function weights.

By imposing a hard limit using the algorithm parameters, it is guaranteed that service quality will never reach nonsensical values that would impose significant changes in the demand: the operator might request the cars always to be placed 500m or less from the user. The soft configuration of the objective function further adjusts the degree of the algorithm's freedom. Unlike the hard limits, using these soft weight parameters defines only the tendency to give higher importance to some performance indicators over the others, without guarantees on the final values. Combining both the hard limits and precisely defining the tendency to prioritize certain parameters, it is possible to ensure that the optimization objectives of the operator are met and that the service quality variations are sufficiently small to prevent having a notable impact on the demand.

5.3.3 Solution algorithm for the variable reservation service quality problem (VRSQP)

In the simulation-based optimization approach used in this work, simulation is combined with the ILS metaheuristic [594, 654] to solve the Variable Reservation Service Quality Problem (VRSQP). The metaheuristic is used to devise a set of *QoS* parameters that are provided to the simulator as an input, and the simulator is used to evaluate profit and satisfied demand. Based on this feedback, the algorithm iteratively decides which changes in the *QoS* across the city to perform to enhance the objective function further. This way, the simulator is acting as an evaluator for the solutions suggested by an algorithm to solve the VRSQP.

Reservation simulator

For this research, the author devised a custom discrete-event microscopic simulator to investigate various reservations-related decisions and reproduce the user/operator interactions under different conditions. The main design goals for the simulator were:

- Ability to model user decisions while performing walk-ins or reserving a vehicle while taking into account the spatial effects of moving vehicles in a minute-to-minute simulation environment;
- Ability to model the provider behavior when deciding whether to accept or reject incoming trips;
- Ability to choose reservation demand as a percentage of walk-ins vs. reservation requests;
- Ability to estimate performance indicators such as the revenue, operating costs, profit, and percentage of satisfied and rejected demand.

The proposed simulation-based methodology assumes the existence of a carsharing trip database with origin/destination coordinates, start times and trip durations for each trip. Trip estimation can be performed by surveying users or specialized mode choice simulations. Further, the demand forecast can be based on historical data on fleet utilization and individual trips.

The simulation is based on a list of walk-in and reservation trips synthesized from the initial trip database by a component called *mode divider*. The number of reservations is parametrized by the reservations percentage parameter (ρ), defined as a ratio of the number of trips that are reserved ahead and the total number of trips. The mode divider uses the Monte Carlo method to divide the input demand into walk-ins and reservations and set up the reservation times.

Summarized, the inputs to the simulator are (1) walk-in trips (2) reservation trips, (3) initial vehicle locations, (4) *QoS* parameters (arrays of r and h across the service area), (5) maximum comfortable walk-in distance c_{wd}, (6) forecasted ideal vehicle stocks in the service area during periods of time B_{ideal}, (7) balancing relocations dispatch period b_p, and (8) balancing relocations trip number per period limit b_n. The pseudocode of the simulator is available in the Algorithm 12. The key component of the simulator is the vehicle location record data structure, denoted as VLR. It is a dictionary which contains the vehicle status and locations for each minute in the simulated period. Supported values of the status variables for a vehicle are: (1) “stationary and available”, (2) “stationary and locked”, (3) “moving by user”, (4) “moving by a staff member”. Vehicles are available to start new trips and to be reserved only when in status (1) “stationary and available”. In all other cases, they are already assigned to a user or in use by a staff member. The simulation starts by loading the initial vehicle locations and initializing the vehicle time record. Results of a simulation run are estimates of the carsharing provider profit and satisfied demand as well as the complete record of all vehicle movements (OD locations and trip start and end time for each performed trip).

The simulation model filters the trips from the trip database for each minute t sequentially.

Algorithm 12 Carsharing reservation simulator pseudocode [182]

```
procedure RESERVATION SIMULATOR(walk-ins, reservations, initial vehicle locations, $QoS$, $c_{wd}$, $B_{ideal}$, $b_p$, $b_n$)
    VLR=initialize vehicle location record(initial vehicle locations)
    for each $t$ in the simulation period do
        initialize set $WID_t$ containing all walk-in demand starting at $t$
        initialize set $RD_t$, containing all reservations to respond to at $t$
        for each walk-in trip $w_i$ in $WID_t$ do
            get the closest stationary and available vehicle $c_{sa}$
            if closest vehicle distance > $c_{wd}$ then
                reject walk-in
            else
                accept walk-in
                calculate walking duration $t_w$
                lock vehicle $c_{sa}$ from $t$ until $t_{start} = t + t_w$
                Update VLR: set status of the vehicle $c_{sa}$ to "moving by user"
                    from $t_{start}$ until $t_{end} = t_{start} +$ duration of the trip
                Update VLR: set status of the vehicle $c_{sa}$ as "stationary and available"
                    at the destination location of $w_i$ from $t_{end}$ onwards
            end if
        end for
        for each reservation trip $res_i$ in $RD_t$ do
            processReservation($res_i$, $B_{ideal}$)
        end for
        if $t$ mod $b_p$ = 0 then
            dispatchBalancingTrips($B_{ideal}, b_n$)
        end if
    end for
    calculate profit
end procedure
```

Walk-ins are accepted or rejected by the user, as defined by the comfortable walk-in distance (c_{wd}) parameter. If a walk-in is accepted, it is assumed that the user will reach the vehicle by walking from his current location (trip origin) to the closest vehicle and that he will start walking immediately after sending the request. Walking duration t_w is estimated under the assumption that the walking speed is 5 km/h and that the walking distance is the Euclidean distance multiplied by a random number in the interval $[1,2]$ to take the impact of the street layout into consideration. Note that c_{wd} is a parameter used to define user behavior with regard to walk-ins and that it is not related to reservation service quality parameter r which applies only to the reservations.

Reservation enforcement (Type 1 relocations) are handled by the processReservation (res_i, B_{ideal}) function in Algorithm 12. There are two possible implementations this function can be redirected to: locking and relocations. The locking version is simple and straightforward: if close enough, lock the closest vehicle from the moment the reservation is made, until the departure, otherwise reject the reservation. The relocations strategy is implemented as detailed previously in Fig. 2, and uses the ideal vehicle stock B_{ideal} to choose vehicles to relocate. The carsharing operator resorts to a taxi service as a backup to ensure the reservations are satisfied even in cases where the fleet is overloaded and there are no free vehicles. The user will be charged the standard service price and the carsharing company will pay the taxi. This way, the service is paying the difference between normal carsharing fees and taxi rides. These trips are considered to be satisfied demand as the service ensured the trip can be performed under the same pricing conditions. Such trips are undesirable as a taxi is typically more expensive and these outsourced trips generate losses.

The balancing trips (type 2 relocations), if balancing is used (i.e., if $b_n > 0$) are dispatched in regular time intervals which is denoted as $dispatchBalancingTrips(B_{ideal}, b_n)$ function in Algorithm 12. Balancing trip assignment is performed based on comparisons of the vehicle stock in the currently running instance and their ideal distribution. Vehicles are relocated from zones with the highest surplus to the zones with the highest deficit as in [643], where a set of all zones in a time interval t is denoted W_t. In case more relocations than b_n are needed to fully balance the system, the simulator will have to choose the distribution according to probabilistic priorities. For each cell with a surplus (supplier) and for each cell with a deficit (demander) origin and destination probabilities are calculated according to the following equations:

$$Prob\_O_{i_t} = \frac{St_{i_t} - B_{ideal\,i_t}}{\sum_{j \in N, St_{j_t} - B_{ideal\,j_t} > 0} St_{j_t} - B_{ideal\,j_t}}, \; \forall i_t \in \mathrm{W}_{\boldsymbol{t}}, \; St_{i_t} - B_{ideal\,i_t} > 0, \tag{5.3}$$

$$Prob\_D_{i_t} = \frac{B_{ideal\,i_t} - St_{i_t}}{\sum_{j \in \mathrm{N}, B_{ideal\,j_t} - St_{j_t} < 0} B_{ideal\,j_t} - St_{j_t}}, \; \forall \mathrm{i_t} \in \mathrm{W}_{\boldsymbol{t}}, \; B_{ideal\,i_t} - St_{i_t} < 0, \tag{5.4}$$

where $Prob_{O_{it}}$ is the probability that cell i will be an origin for the balancing trip at time t,

$Prob_{D_{it}}$ is the probability that cell i will be a destination for the balancing trip at time t, $B_{ideal_{it}}$ is an ideally balanced number of vehicles in cell i at time t, St_{i_t} is the vehicle stock in zone i during time t. Based on these probabilities, the origin and destination are assigned in each trip. The equations are inspired by the random proportional rule used in the ant colony optimisation metaheuristic, as described in 3.6.7.

While real systems would most likely include undesirable effects, such as no-shows and late cancellations, in this work, they are not taken into account. Each area is likely to have slightly different no-show patterns, depending on the local culture and user habits and such data is difficult to obtain. Nevertheless, each provider would probably devise some type of penalty strategy (e.g., charging a no-show fee or forbidding the user to re-book the same vehicle after not taking it on time) to discourage such behavior and compensate for the financial losses, similar to practices in taxi reservations [658]. For this reason, it can be argued that the effects of no-shows can be neglected for the purpose of this work.

Several decisions in the simulator are based on random behavior, for example, user-car walking time estimation and dividing the demand into walk-ins and reservations. The simulator has two modes: non-deterministic and deterministic. In the deterministic mode, random value generators are always initialized with the same seed thus producing the same list of random numbers.

The profit (denoted P) calculation is performed by going through the VLR and calculating the revenue (R) and costs (C) of vehicle operations:

$$P = R - C \tag{5.5}$$

The revenue component is calculated as a sum of individual revenues of user trips, based on the service price per minute, denoted as π, and the given trip duration estimates, denoted duration$(trip_i)$:

$$R = \sum_{i \in \mathbf{L}} \text{duration}\,(trip_i) \cdot \pi \tag{5.6}$$

where $\mathbf{L}$ is the set of all user trips. The costs are calculated as a sum of fixed costs (C_f) and variable costs (C_v):

$$C = C_f + C_v. \tag{5.7}$$

Fixed costs do not depend on the number of performed trips and are calculated as a sum of daily parking costs and daily vehicle depreciation costs for all the vehicles in the fleet:

$$C_f = A \cdot C_{park} + A \cdot C_{veh}, \tag{5.8}$$

where A is the fleet size (number of cars),C_{park} is the cost of parking per vehicle per day in the city and C_{veh} denotes the costs of depreciation per vehicle per day. Variable costs are calculated as the sum of costs of vehicle maintenance, relocation and taxi:

$$C_v = \sum_{i \in \mathbf{L}} \text{duration}\,(trip_i) \cdot C_{mv} + \sum_{i \in \mathbf{L}'} \text{duration}\,(trip_i) \cdot C_r + \sum_{i \in \mathbf{G}} (C_{taxi_{start}} + \text{ distance}\,(trip_i) \cdot \mathrm{C}_{taxi_{km}}), \tag{5.9}$$

where $\mathbf{L}$ is the set of all user trips, $\mathbf{L}'$ is the set of all relocation trips, C_{mv} is the cost of vehicle maintenance per minute driven, C_r is the cost of a relocation operation per minute driven, G is the set of all trips redirected to taxi, distance$(trip_i)$ is the estimated distance of the trip i, $C_{taxi_{start}}$ is the taxi start price and $C_{taxi_{km}}$ is the price of driving 1 km in a taxi.

Iterated local search metaheuristic

The set of feasible solutions Q is defined by a tuple $Q(N, r_{min}, r_{max}, h_{min}, h_{max})$, where N is the number of zones. It contains all possible values of the r and h parameters within the allowed radius $[r_{min}, r_{max}]$ and time $[h_{min}, h_{max}]$ intervals for each zone. For the purpose of using a heuristic to solve the problem, both radius and reservation time values are discretised, with minimum resolution steps of r_{resol} and h_{resol} as parameters which, along with the allowed intervals for r and h, define the solution space:

$$|Q| = \left(\left(\frac{r_{max} \ - \ r_{min}}{r_{resol}} + 1 \right) \cdot \left(\frac{h_{max} - h_{min}}{h_{resol}} + 1 \right) \right)^N \tag{5.10}$$

Note that the size of the feasible solution space grows as the radius and time resolution increases and especially quickly as spatial resolution is increased (number of zones N). Further, it should be emphasized that r_{min}, r_{max}, h_{min} and h_{max} are algorithm parameters for establishing possible ranges of variables, that in general do not correspond to actual values the algorithm will produce in the solutions. They are defined in order to allow users control over the values of the QoS – allowing the radius to be larger than 500m does not make much sense as this has a potential to place cars too far from the users to be practically accessible. The algorithm guarantees that solutions will not have radius r larger than rmax in any of the city zones.

For each QoS set, it is possible to calculate the profit and the accepted demand by running the simulator with these specific reservation parameters in the city zones. To ensure that the evaluation of different solutions can be compared, the deterministic mode of the simulator is used. The pseudo-code of the algorithm is given in 13. The overall algorithm has five parameters: execution time and the allowed QoS radius and time intervals. The LSO and PO operators have more detailed parameters as described below. The algorithm begins by generating an initial solution $QoS_{initial}$, with h and r across zones initialized to random values in the allowed intervals, $r \in [r_{min}, r_{max}]$, $h \in [h_{min}, h_{max}]$. The local search is then applied to this solution, resulting in the first local optimum QoS . The main algorithm loop then runs until the allowed time passes (time). The loop consists of the perturbation operator generating the current perturbed solution

QoS' and then applying the local search to the perturbed solution to create a new local optimum $QoS'*$. The best-found solution is kept at all times and the random walk ILS movement strategy is used, meaning that the next initial solution for the PO is always the current local optimum.

Algorithm 13 Implemented iterated local search (ILS) metaheuristic for VRSQP pseudocode [182]

```
procedure ITERATED LOCAL SEARCH($time$, $r_{min}$, $r_{max}$, $h_{min}$, $h_{max}$)
    $QoS_{initial}$ =generate initial solution($r_{min}$, $r_{max}$, $h_{min}$, $h_{max}$)
    $QoS_{best}$ = $QoS^*$ = local search($QoS_{initial}$)
    repeat
        $QoS'$ = perturb($QoS^*$)
        $QoS'^*$ =localSearch($QoS'$)
        if evaluation for $QoS'$ is greater than the evaluation for $QoS_{best}$ then
            $QoS_{best}$ = $QoS'$
        end if
        $QoS^*$ = $QoS'^*$
    until time expired
end procedure
```

Local search operator

The LSO used in this work is a simple method that tries to increase and then decrease both the reservation distance and reservation time in the solution. After trying all the options, it chooses the change that caused the biggest improvement in the objective function value or retains the original value if no improvements have been produced. It has eight parameters: the initial solution QoS; the distance and time steps rstep and hstep define the increments of the variables that the local search will perform; the *partToSearch* value needs to be in the interval [0, 1] and it determines the approximate percentage of the QoS table which can be changed by the operator; the parameters r_{min}, r_{max} , h_{min}, h_{max} define the allowed interval for the reservation distance and reservation time. The pseudocode of the operator can be found in Algorithm 14.

The operator visits each element of the QoS table in the main loop. The functionality of determining the part of the table to change is implemented by a random number generator which accepts modifications of solution elements with probability equal to the *partToSearch* parameter. The operator is non-deterministic and this way, it can achieve more diversity in the search. For each table element that the local search is modifying, the operator first tries to adjust the reservation radius. It first adds r_{step} to the current distance value, then it subtracts r_{step} and evaluates both modifications. If neither the adding nor the subtracting of the step value improved the solution, the table remains unchanged. If any of these produced an improvement, the table is updated so that the new r value for the current element is either the added or subtracted value, depending on which one caused a greater quality increase in the objective. The analogous procedure is performed with the reservation time-ahead parameter h.

Algorithm 14 Implemented local search for VRSQP pseudocode [182]

```
procedure LOCAL SEARCH($QoS$, $r_{step}$, $h_{step}$, $r_{min}$, $r_{max}$, $h_{min}$, $h_{max}$, $partToSearch$)
    for each element $i$ in the service quality table $QoS_i$ = $(r_i, h_i)$ do
        initialize random number $e \in [0,1]$
        if $e > partToSearch$ then
            continue with next element $QoS_{i+1}$
        else
            while improvement achieved do
                $r_{down}$ = $r_i$ - $r_{step}$, unless this would make $r_{down} < r_{min}$
                $QoSR_{down}$ = $(r_{down}, h_i)$
                $r_{up} = r_i + r_{step}$, unless this would make $r_{up} > r_{max}$
                $QoSR_{up}$ = $(r_{up}, h_i)$
                update $QoS$ to the element of$\{QoS, QoSR_{down}, QoSR_{up}\}$
                    for which $Z$ is maximal
            end while
            while improvement achieved do
                $h_{down} = h_i - h_{step}$, unless this would make $h_{down} < h_{min}$
                $QoSH_{down} = (r_i, h_{down})$
                $h_{up} = h_i + h_{step}$, unless this would make $h_{up} > h_{max}$
                $QoSH_{up} = (r_i, h_{up})$
                update $QoS$ to the element of $\{QoS, QoSH_{down}, QoSH_{up}\}$
                    for which $Z$ is maximal
            end while
        end if
    end for
end procedure
```

The procedure ends when the main loop has iterated through all table elements. By systematically investigating the effects of distance and time variation and combining the effects of small changes, the local search can produce notable improvements to the initial solutions, especially when iterated with the perturbation operator in the ILS metaheuristic.

Perturbation operator (PO)

The perturbation operator introduces random changes in the part of the *QoS* table elements. The operator has eight parameters: the input *QoS* to modify, number of changes to make c, distance and time change steps r and h, and allowed distance and time intervals defined by r_{min}, r_{max}, h_{min}, h_{max}. In total, the algorithm performs c change attempts of the randomly selected cells in the *QoS* table. It might change up to c cells or less if some cells are changed multiple times or left unchanged due to reaching the allowed limits of their values.

After choosing an element to change, the distance for the current element is modified. The algorithm first decides on the direction of the change: increase or decrease. To perform this choice, a random Boolean value ω_r is produced by the random value generator. If r is $true$, then the distance in the current element will be increased for the distance step r, if it is false it performs the decrease, unless the change would take r out of the $[r_{min}, r_{max}]$ interval. The analogous operation is performed for the reservation time-ahead using the Boolean variable h.

Algorithm 15 Implemented perturbation operator for VRSQP [182]

```
procedure PERTURB(QoS, $c$, $\Delta r$, $\Delta h$, $r_{min}$, $r_{max}$, $h_{min}$, $h_{max}$)
    repeat
        choose a random element of the $QoS$ table $QoS_i = (r_i, h_i)$
        choose two random Boolean variables $\omega_r$ and $\omega_h$
        if $\omega_r$ is $true$ then $r_i = r_i + \Delta r$, unless it would make $r_i > r_{max}$
        else$r_i = ri - \Delta r$, unless it would make $r_i < r_{min}$
        end if
        if $\omega_h$ is $true$ then
            $h_i = h_i + \omega_h$, unless it would make $h_i > h_{max}$
        else
            $h_i = hi - \Delta h$, unless it would make $h_i < h_{min}$
        end if
        $c = c - 1$
    until $c = 0$
end procedure
```

The fact that both the perturbation as well as the local search operator act only on a randomly selected subset of the table, combined with various possibilities for parameter selection, provides possibilities to adjust the algorithm to work as needed: so that the perturbation is low enough in order to keep the algorithm focused but still high enough not to prevent the algorithm from being stuck in local optima.

Table 5.3: Hypothetical city features [182]

Problem instance	Total size	Trip number	Average trip time (min)	Average trip distance (km)	Fleet size A (cars)	t_a (min)
Town (500)	5 x 5 km	500	5	2.6 km	10	40
Metropolis (40,000)	50 x 50 km	40,000	51	24.6 km	2000	180

5.3.4 Computational experiments

To test the performance of the proposed methodology, experiments were performed on two sets of benchmark problem instances: hypothetical cities and case-study city. The demand for the hypothetical cities is generated using a custom built trip generator with trip patterns based on typical features for two extreme cases: town and metropolis. The case-study city in this work is the Lisbon municipality in Portugal, whose data was obtained from an agent-based model of Lisbon carsharing mobility [659]. Both categories include trips from a single working day of carsharing with reservation-ahead times assumed to be less than 18 h. Even though only one day is simulated, reservations from "the night before" and any time before t=0 in the simulation are allowed as long as there is enough time for a system response during the simulation period. In real systems that do not have simulation limitations, periods much longer than 18 h might be feasible. Trips that start at $t < t_a$ require response too early during the simulation start, and are rejected by the system.

Several experiments with up to three days of trips have shown that the results tend to be approximately proportional to the simulated number of days. Since the current limit of 18 h allows simulating overnight reservations, simulating only one day is sufficiently representative for the purpose of this work. When simulating largest datasets, scalability issues start to occur as simulation times get longer. Due to the analyses that indicate that there is no loss of generality when running one day and given the speed benefits of it, in this work, one day of trips is simulated.

Hypothetical cities

Problem instances for hypothetical cities are generated using a custom generator, built specifically for this study to enable generating carsharing trips in different environments. Two different hypothetical cities are used: town and metropolis: the town is representative of a small urban area with several tens of thousands of people, while the hypothetical metropolis is a large, densely populated urban area, typical of some of the largest cities in the world. Throughout this Chapter, problem instances are named based on the city name followed by the number of trips in the parentheses, e.g., Town (500). The detailed features of the hypothetical cities used in this work are shown in Table 5.3, where geographical size, trip number, average trip duration and distance, number of cars and response time t_a are defined. In the case of a small town, response time is much shorter than in the metropolis since even in the most conservative estimates, it takes up to 40 min to reach any location in the service area. In the metropolis, this is estimated to take up to three hours.

The generation process involves both trip generation and distribution. It takes into account the total volume of trips and the temporal and spatial demand variations, all of which can be specified in the input. Generating trips based on an input distribution is implemented using a technique inspired by the fitness proportional selection operator used in genetic algorithm and the pseudorandom proportional rule in the ant colony optimization metaheuristic [124, 343, 660].

Modeling trip intensity is done analogously: for each time interval, an individual value can be set up for the relative probability that an individual cell will be an origin and a separate value for the probability that the same area will be a destination. This allows modeling of the demand variations. The city consists of a highly populated central business district and broad residential areas on the outskirts. The southern part of the service area is water surface, therefore all relative chances of being an origin or destination are zero in these zones. As typical in the mornings, residential areas have more outgoing trips than the business areas. Business areas have more incoming trips during the morning. Initial vehicle locations are devised based on the trip origin probabilities during morning rush hour — this way the initial number of vehicles is proportional to the number of origins across zones to match the morning demand.

Case-study city: Lisbon

Realistic problem instances are based on the carsharing trip forecasting case study of the Lisbon municipality in Portugal [659, 661, 662] and an extensive mobility survey, performed by the Lisbon Municipality [663, 664]. Results from these studies were also used to decide on the fleet size and initial vehicle distribution. Unlike the rough estimates included in the hypothetical city models, this model takes into account the precise transportation habits of the local population and microscopic effects occurring on a high-resolution network of nodes.

The Lisbon area has been dealing with several mobility issues, including congestions and lack of parking space. Innovative transport solutions, including carsharing, are one of the alternatives that could help reduce mobility problems in the city. Details of the Lisbon problem instances are reported in Table 5.4. .

Table 5.4: The city of Lisbon features [182]

	Geographical size (km)	Trip number	Average trip duration (min)	Average trip distance (km)	Fleet size A (cars)	t_a (min)
Lisbon (3,000)	11.6 x 11.0	3,000	14.3	4.3	80	60
Lisbon (6,000)	11.6 x 11.0	6,000	14.5	4.5	159	60
Lisbon (12,000)	11.6 x 11.0	12,000	14.5	4.4	318	60
Lisbon (25,000)	11.6 x 11.0	25,000	14,5	4,4	664	60

Simulation parameters for all runs

All of the following parameters are the same for all problem instances defined above. This can be a limitation since for instance parking price should be different in a small city when compared to a big city. Nevertheless, maintaining these parameters equal allows for a better comparison between the scenarios.

The costs of vehicle ownership are estimated using the Interfile tool for car ownership costs estimation [665]. As a reference vehicle, use an average city vehicle with the initial cost of 20,000€ is used, under assumptions that the company financed the entire initial cost using a loan with an interest rate of 12% and the vehicle's residual value after three years equal to 5000€. For such a vehicle, the cost of depreciation (C_v) is 17€ per day, with expected use duration of 3 years. The cost of maintaining the vehicle (C_{mv}) is estimated to be 0.007 € per minute, taking into account insurance, fees, taxes, fuel, maintenance and wear of the vehicle. The relocation cost C_r is estimated to 0.20€ per minute, and it includes fuel, vehicle maintenance and staff costs, as well as the compensation for the cost of reaching the relocation trip origin. The parking cost C_p is estimated to be 1.2€ per hour [666]. The service fee per minute is set to 0.30€ per minute, based on the rates of the global operator Car2Go [667]. Taxi start price is set to 3.5€ and price of driving 1 km is 0.47€ [668]. User walking speed is set to 5 km/h and the vehicle walk-in distance c_{wd} is set to 250m [669].

Running the experiments

The simulator and the optimization algorithm are implemented in Java 1.8 programming language. The experiments were performed on a computer equipped with a 2.4 GHz Intel Core i7-4700HQ processor and 16 GB of RAM, using Java 1.8 runtime environment under Windows 10 operating system. The running time to simulate one full day is less than a second in the

Table 5.5: Iterated local search metaheuristic parameters [182]

Parameter	**Value**
Comfortable walk-in distance $\boldsymbol{c}_{wd}$	250 m
Local search distance step $\boldsymbol{r_{step}}$	200 m
Local search time step $\boldsymbol{h_{step}}$	480 min
Local search percentage to explore **partToSearch**	100 %
Perturbation distance change $\boldsymbol{\Delta r}$	100 m
Perturbation time change $\boldsymbol{\Delta h}$	300 min
Perturbation number of elements to consider $\boldsymbol{c}$	50 (50%)
Minimum allowed distance $\boldsymbol{r_{min}}$	50 m
Maximum allowed distance $\boldsymbol{r_{max}}$	500 m
Minimum allowed time $\boldsymbol{h_{min}}$	60 min
Maximum allowed time $\boldsymbol{h_{max}}$	1080 min
Radius resolution $\boldsymbol{r_{resol}}$	1 m
Time resolution $\boldsymbol{h_{resol}}$	1 min

smallest instances and around 30 s for the ones with the largest number of cars and trips.

The experiments were performed with three different sets of values for the objective function parameters. A balanced parameter set is used, that gives half of the weight to the profit: $w_p = 0.5$ and the rest is equally distributed to other three service quality components: $w_r = 0.167$, $w_h = 0.167$ and $w_d = 0.167$. The sum of all factors is equal to one, to ensure that the resulting values will be normalized to the [0, 1] interval. To optimize metaheuristic performance, tuning experiments were performed on the Lisbon (6,000) dataset with the balanced objective function parameters. The best performing configuration found during tuning is shown in Table 5.5, and this set of algorithm parameters was used in all subsequent experiments in this work. Note that all of these parameters are algorithm input parameters, and that the distance and time limits $[r_{min}, r_{max}]$, $[h_{min}, h_{max}]$ refer to allowed intervals, not the realized values in the solutions, although they might coincide.

Table 5.6: Vehicle locking method performance in Town (500 trips) problem instance [182]

Reservations percentage (ρ)	**Profit (P) (€/day)**	**Costs (€/day)**			**Revenue (R) (€/day)**	**Demand satisfied**
		C	**C$_f$**	**C$_v$**		
0 %	-351.27	460.55	458.00	2.55	109.28	9.80%
20 %	-370.68	460.09	458.00	2.09	89.40	8.03%
40 %	-383.93	459.77	458.00	1.77	75.83	6.48%
60 %	-384.66	459.75	458.00	1.75	75.09	5.89%
80 %	-391.00	459.60	458.00	1.60	68.60	5.28%
100 %	-419.37	458.92	458.00	0.92	39.56	4.10%

5.3.5 Results

R-BR method under constant *QoS*

In the first round of the experiments, the differences in the described reservation enforcement strategies are assessed under constant service quality in the entire city area. The reservation service quality for all experiments in this round was set to r = 200 m, h = 600 min (10 h). In the input trip volume, which consists of spontaneous walk-ins and trips reserved ahead, the reservations percentage (denoted ρ) was varied, while keeping all other conditions constant. Note that = 10% does not mean 10% more trips in total, it means that 10% of trips that were walk-ins in the original dataset are now long-term reservations.

The results are shown in Fig. 5.4. The x-axis of each graph shows the reservation percentage, and the y-axis shows the daily profit in euros for that day. A detailed breakdown of profit into its components: revenue (R), fixed, variable and total cost (denoted C_f , C_v and C respectively) as well as the percentage of demand satisfied and outsourced demand (taxis) are specifically presented for the Town (500) and the Lisbon (25,000) problem instances in Tables 5.6 - 5.9.

When the reservation service is not offered ($\rho = 0$), Town (500), Lisbon (3000) and Lisbon (6000) instances are generating losses and all others are profitable. As seen in Table 4, in the Town (500), only 9.80% of the demand is satisfied for $\rho = 0$. Despite the fact that there are 500 potential trips, most of them are not done because the closest vehicle is too far for a comfortable walk-in at the moment of the request ($c_wd = 250$ m). A small fleet with only 10 vehicles is not enough for sufficient coverage and to ensure that, on average, vehicles are close enough to interested users. The revenues generated by such a low number of trips are not sufficient even to cover the fixed costs of fleet ownership and parking. Similar results are observed in the smallest Lisbon instances. These results indicate that carsharing can hardly be

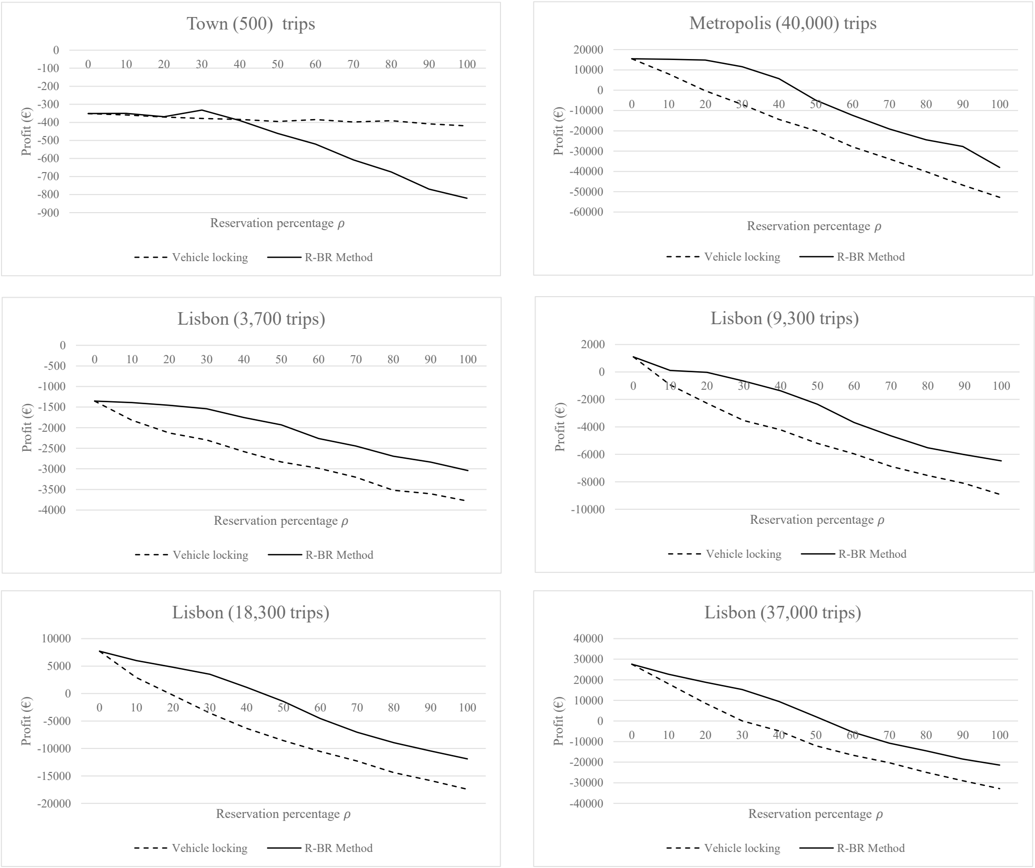


Figure 5.4: Comparison of vehicle locking and reservation based relocations under constant *QoS* [182]

Table 5.7: R-BR method performance in Town (500 trips) problem instance [182]

Reservations percentage (ρ)	**Profit (P) (€/day)**	**Costs (€/day)**			**Revenue (R) (€/day)**	**Outsourced (taxi) demand**	**Demand satisfied**
		C	**C$_f$**	**C$_v$**			
0 %	-351.27	460.55	458.00	2.55	109.28	0.00%	9.80%
20 %	-368.91	605.95	458.00	147.95	237.04	2.21%	21.29%
40 %	-391.47	813.42	458.00	355.42	421.95	7.44%	35.81%
60 %	-535.02	1,102.37	458.00	644.37	567.35	16.53%	51.61%
80 %	-686.17	1,405.30	458.00	947.30	719.12	28.43%	64.72%
100 %	-829.65	1,744.10	458.00	1,286.10	914.46	40.69%	80.57%

Table 5.8: Vehicle locking method performance in Lisbon (25,000 trips) problem instance [182]

Reservations percentage (ρ)	**Profit (P) (€/day)**	**Costs (€/day)**			**Revenue (R) (€/day)**	**Demand satisfied**
		C	**C**$_f$	**C**$_v$		
0 %	15,249.54	31,502.07	30,411.20	1,090.87	46,751.61	42.92%
20 %	5,027.00	31,257.85	30,411.20	846.65	36,284.85	34.41%
40 %	-4,116.62	31,039.40	30,411.20	628.20	26,922.78	26.61%
60 %	-10,641.26	30,883.52	30,411.20	472.32	20,242.26	20.61%
80 %	-16,898.83	30,734.02	30,411.20	322.82	13,835.19	14.75%
100 %	-21,783.32	30,617.33	30,411.20	206.13	8,834.01	9.86%

Table 5.9: R-BR method performance in Lisbon (25,000 trips) problem instance [182]

Reservations percentage (ρ)	**Profit (P) (€/day)**	**Costs (€/day)**			**Revenue (R) (€/day)**	**Outsourced (taxi) demand**	**Demand satisfied**
		C	**C**$_f$	**C**$_v$			
0 %	15,249.54	31,502.07	30,411.20	1,090.87	46,751.61	0.00%	42.92%
20 %	12,870.18	36,080.55	30,411.20	5,669.35	48,950.73	0.00%	45.62%
40 %	9,497.43	42,789.81	30,411.20	12,378.61	52,287.24	1.15%	49.91%
60 %	1,558.51	54,054.44	30,411.20	23,643.24	55,612.95	4.35%	54.96%
80 %	-4,965.13	68,583.61	30,411.20	38,172.41	63,618.48	12.33%	64.68%
100 %	-10,402.24	84,073.84	30,411.20	53,662.64	73,671.60	23.97%	77.29%

profitable below a certain threshold of a minimum trip volume and confirm similar findings in other studies [670, 671, 672].

When reservations are allowed ($\rho > 0$) and vehicle locking is applied as the enforcement method, the profit steeply drops. At $\rho = 20\%$, only Lisbon (25,000) is profitable and when reaches 30%, vehicle locking is not profitable in any of the investigated problem instances. This method causes long waiting periods during which vehicles are idle and therefore it reduces the overall service availability for potential trips. These effects are clearly visible in Tables 5.6 and 5.8: more reservations bring less satisfied demand and less revenue. Very large losses result from high reservation percentages and high trip volume, e.g., more than 20,000 € per day in Lisbon (25,000). The proposed RB-R method also brings profit drops when reservations are present. However, in all problem instances except in Town (500), R-BR considerably outperforms vehicle locking. In Lisbon (3000) and Lisbon (6000), R-BR brings more revenue and allows more demand to be satisfied. Nevertheless, the improvement is not sufficient to turn around these losses. In the remaining four instances, R-BR can maintain the profitability of the operations with two to three times the reservation volume than the vehicle locking. While R-BR imposes additional relocation and taxi costs, the method leads to less rejected trips which brings higher revenues (Tables 5.7 and 5.9). Whether the benefits of the added revenue will outweigh the costs depends on the number of performed trips. Note that even with $\rho = 100\%$ all trips are not accepted since the *QoS* settings allow the system to reject reservations more than 10 h ahead ($h = 600$ min).

In the Town (500) instance, the profit of vehicle locking is similar to the RB-R for low , however, with more than 50% of reservations, the RB-R quickly starts to be worse than locking (Fig. 5.4). Part of this is due to the increased costs brought by the relocation movements, however, the key reason for poor performance with high is the increased level of trip outsourcing to taxi. With $\rho > 50\%$, the outsourcing rate quickly starts rising, adding large extra costs (Table 5.7).

The performance of the newly proposed R-BR method outperforms the simple vehicle locking, in all cases with sufficiently high demand. While R-BR method has a high potential to improve the profit, it should only be considered for the systems that are profitable with no reservations as the added operational cost of enforcing reservations can lead to even more losses when the number of trips is small.

Variable *QoS*

To further improve the performance of the reservations, the devised ILS metaheuristic is applied on the available problem instances in a setting with high reservation load: $\rho = 50\%$. Profit bounds for all problem instances P_{min} and P_{max} were estimated by running a simple *QoS sweep* algorithm that examines all (r, h) combinations with the (50 m, 60 min) steps under various

Table 5.10: ILS performance in 5 runs [182]

Problem instance	**Constant QoS**	**Variable QoS**					
	Best known Z	**ILS time limit (h)**	**Z in 5 ILS runs**				
			Average	**Median**	**Best**	**Worst**	**Std. dev**
Town (500)	53.81%	2	62.74%	62.00%	71.54%	57.72%	5.66%
Metropolis (40,000)	76.49%	10	80.79%	81.02%	82.75%	79.07%	1.38%
Lisbon (3,000)	68.24%	2	74.14%	73.93%	74.97%	73.21%	0.75%
Lisbon (6,000)	66.59%	2	77.14%	77.05%	78.35%	75.92%	1.05%
Lisbon (12,000)	68.08%	2	76.58%	76.03%	79.87%	73.09%	2.53%
Lisbon (25,000)	70.42%	5	76.92%	78.73%	78.96%	72.18%	2.95%

reservation percentages. Further, an additional run of the ILS was done for highly profitable problem instances. After the profit bounds have been established, the best constant *QoS* was found by calculating the entire objective function in the *QoS* sweep algorithm.

Experiments with five runs of the ILS metaheuristic on each problem instance were performed, with the initial solution being a random solution with $r \in [50, 500]$ and $h \in [60, 1080]$. The other algorithm parameters were used as in Table 5.5. For smaller problem instances, the algorithm is able to produce good solutions more quickly than for those with more trips and cars. Therefore, the time limit of 2 h was used for problem instances with less trips, while up to 10 h was used for those with more trips and longer time needed for solution evaluation. The results are given in detail in Table 5.10, where average, median, best and worst Z in the performed runs, as well as the standard deviation of the objective function values are provided. The results show that ILS was always able to find a better solution than the best known with constant *QoS*.

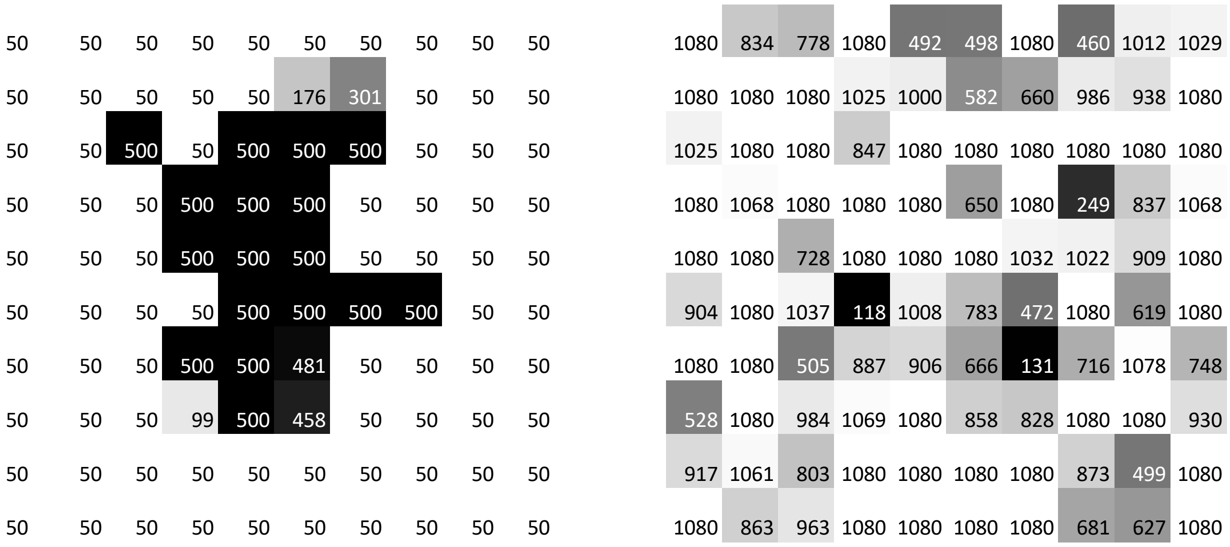
A detailed comparison of best known constant and variable *QoS* solutions is provided in Table 5.11, where the elements of the objective function are given: average time $\overline{h}$, and radius $\overline{r}$, over zones, as determined by the heuristic as well as the satisfied demand, profit and the overall objective function value Z. These results show that th ILS metaheuristic was able to improve the profit in all problem instances, in some of them substantially. In most examples, the satisfied demand is slightly lower than with the constant *QoS*. The average *r* and *h* are very comfortable for all variable *QoS* solutions: less than 200m and more than 12 h in all problem instances. Further, Lisbon (6000) is not profitable even with the best constant *QoS*. However, ILS was able to achieve high increases of the profit, turning the service generating losses into a profitable one. Attaining profitability with the large reservations pressure of 50% used in these experiments is very difficult, as shown in the constant *QoS* experiments. Nevertheless, using ILS, it was possible to optimize the profit to the levels which are better than the profits with no reservations. The especially good *QoS* with high profits for Metropolis (40,000) and Lisbon (25,000) problem instances show the potential of this method with large trip volume. For example, the Metropolis (40,000) with no reservations has a daily profit of 16,573.85 € (Fig. 5.4), while the profit of the best variable *QoS* solution with 50% reservations is a slightly higher value of 17,905.17€. At the same reservation level, the constant *QoS* solution with the best Z is barely profitable (less than 300€ per day). The highest known constant *QoS* profit is 11,122.17€, achieved with much higher *r* and similar *h* *QoS* (*r*=500 m, *h*=840 min). For an additional comparison, vehicle locking with 50% of reservations causes losses of more than 35,000 € per day.

Additionally, the performance of both ILS and best constant solutions is compared with the basic random restart search algorithm in which the best out of randomly generated solutions is selected. With run-times equal to those given to the ILS, the random restart algorithm was not able to outperform the best-found constant solution, and the solutions found by ILS outperform it. A very simple algorithm such as random restart is not able to refine the solutions well enough to give good results and produces a very irregular distribution of radiuses and times that does not reflect the shape of the central business district as ILS does.

Detailed values of *QoS* parameters across the zones for the best-known solution for Metropolis (40,000) are shown in Fig. 5.5, where the radiuses (left) and reservation times ahead (right) are displayed in a heat map. Comparing these values with O-D probabilities indicate that the ILS metaheuristic was able to capture the general behavior of the system. In general, the algorithm lowered the service quality in zones with many trips and kept it very high in areas with fewer trips. Increasing the radius in the central business district has the benefit of lowering the relocation costs. Likewise, it is in general kept low in zones with fewer trips where due to the low concentration of vehicles, relocations will most likely be needed regardless of *r*. Average *r* is 139 m, however, median *r* is 50 m, equal to the lowest allowed value rmin=50 m. Distribution of times ahead is similar: the *h* heat-maps (Fig. 5.5, right) give an approximate outline of the city center contours. In areas with a lot of trips, the algorithm had a tendency to lower the allowed time-ahead *t*, most likely due to the fact that many reservations in these areas have the potential to overload the fleet and cause too many relocations and outsourcing costs. The

Table 5.11: Comparison of best known constant and variable *QoS* solutions [182]

Problem instance	**Best found constant** *QoS*					**Best found variable** *QoS*				
	h (min)	***r*** **(m)**	$\frac{d_{sat}}{d_{tot}}$	**Profit (€)**	***Z***	$\bar{h}$ **(min)**	$\bar{r}$ **(m)**	$\frac{d_{sat}}{d_{tot}}$	**Profit (€)**	***Z***
Town (500 trips)	180	50	19%	-368.59	53.81%	751.39	129.0	38%	-335.44	71.54%
Metropolis (40,000 trips)	1080	50	53%	293.15	76.49%	924.71	139.15	47%	17,905.17	82.75%
Lisbon (3,000 trips)	1080	50	49%	-1,368.32	68.24%	764.65	135.56	40%	-725.82	74.97%
Lisbon (6,000 trips)	900	50	50%	-1,255.04	65.39%	786.83	153.35	44%	275.90	78.35%
Lisbon (12,000 trips)	900	500	52%	2,267.86	68.08%	821.83	196.36	48%	2,832.93	79.87%
Lisbon (25,000 trips)	960	500	57%	10,668.61	70.42%	785.12	186.32	51%	11,552.52	79.88%



50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	176	301	50	50	50
50	50	500	50	500	500	500	50	50	50
50	50	50	500	500	500	50	50	50	50
50	50	50	500	500	500	50	50	50	50
50	50	50	50	500	500	500	500	50	50
50	50	50	500	500	481	50	50	50	50
50	50	50	99	500	458	50	50	50	50
50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50

1080	834	778	1080	492	498	1080	460	1012	1029
1080	1080	1080	1025	1000	582	660	986	938	1080
1025	1080	1080	847	1080	1080	1080	1080	1080	1080
1080	1068	1080	1080	1080	650	1080	249	837	1068
1080	1080	728	1080	1080	1080	1032	1022	909	1080
904	1080	1037	118	1008	783	472	1080	619	1080
1080	1080	505	887	906	666	131	716	1078	748
528	1080	984	1069	1080	858	828	1080	1080	930
917	1061	803	1080	1080	1080	1080	873	499	1080
1080	863	963	1080	1080	1080	1080	681	627	1080

Figure 5.5: Radiuses *r* (left) and times ahead *h* (right) in the best known variable *QoS* solution for Metropolis (40,000) [182]

average and median values of *r* and *h* are only slightly worse than in the best known constant *QoS* and even the zones with the lowest *QoS* by far outperform the reservation time-ahead of 30 min that is nowadays allowed by the carsharing operators.

5.3.6 Speeding up the algorithm: initial solution tuning

While the results in the previous Section show that the ILS was flexible enough to find solutions to this transportation problem in reasonable time, there is still room for improvement. An especially interesting direction, since this is a problem with a slow evaluation function was to investigate the potential of using a fixed initial point. This way, if a particularly good initial solution is known, the search could start from it, this way potentially saving a lot of time to reach to a good area of the search space from a random initial point.

To asses this, first the constant *QoS* sweep algorithm is run to find the best constant *QoS* solutions for two test problem instances: Lisbon (3,000) and Lisbon (12,000). The best constant solutions were ($r = 50$m, $h = 1080$ min) for Lisbon (3,000), and ($r = 100$m, $h = 1080$min) for Lisbon (12,000). The best constant solutions were then used as the initial point for the ILS, and the results can be found in Table 5.12. It can be seen that the algorithm start from a good solution considerably improved the solution quality. The Lisbon (3,000) instance is difficult to optimise due to low number of trips, nevertheless, using the best known constant solution as the initial point added extra 5% to the total evaluation. The difference is striking for the Lisbon (12,000) problem instance, where the evaluation increased to more than 113%. The evaluation

Table 5.12: Speeding up the ILS by using a good initial solution [182]

Problem instance	Random initial solution	Best known constant initial solution
Lisbon (3,000)	74.14%	**79.44%**
Lisbon (12,000)	76.58	**113.20%**

higher than 100% indicates it is higher than the estimated top profit, that is set before running the algorithms. This both demonstrates the potential of efficient algorithms on problems where good solutions do exist, as well as the potential of the algorithm improvements by selecting a good initial point. A single evaluation for Lisbon (12,000) requires around 2 seconds, therefore for this problem, up to 3600 evaluations were possible.

Practical considerations

The author recommends gradual implementation in the existing carsharing systems. Before implementation, a rigorous viability assessment of reservations is required, since long-term reservations are not well suited to, e.g., carsharing systems with low demand or small fleets. Reservations have a clearly defined market: airport trips, intermodal trips connecting users to trains, buses, and other modes with a fixed schedule, users who need a temporary replacement car for everyday commute, people who do not own a car and might want to use carsharing to go to work during a week of bad weather etc. Therefore, a survey of user preferences to assess the market size for the long-term reservation service should be performed prior to implementing R-BR to help decide if the revenue increase from such a service would justify the change in operations.

The gradual implementation of the R-BR methodology can be performed in the following four steps:

1. Setting up the constant service quality reservation system,
2. Experimenting with improving the key performance indicators (profit, satisfied demand, user satisfaction) using the ILS metaheuristic,
3. Evaluating the real-world performance,
4. Repeating steps 2 and 3 if needed.

The optimization in step 2 requires a demand database or a forecast. Further, the ILS algorithm in a simulation-optimization approach is not fast enough for real-time decisions so it would typically be used for long-term iterative planning, on, e.g., weekly or monthly basis. Depending on the yearly and daily variations in demand, it is advisable to create several separate setups for typical working day vs. a weekend, summer months, and other specific circumstances. Experimenting with several different solutions before choosing the final one is recommended, while carefully monitoring the service quality KPIs in the real carsharing system.

5.3.7 Carsharing reservations – concluding remarks and future work

Carsharing systems are being classified as a sustainable green mode, especially if provided with electric vehicle fleets. Nevertheless, one-way systems, in which the user may drop-off the vehicle anywhere inside a service area, are still being used by only a small segment of the urban transport demand. As the number of users grows, so do the logistic management problems to be solved if the quality of service and profitability are to be maintained.

Vehicle reservations provided through simple vehicle locking, by which the vehicle must stay idle until the client comes, have a too high impact on the profit to be viable as showed in this Chapter. While the author is not aware of any research that estimates the potential of reservations to attract more customers, it is reasonable to assume that customers do not favor high restrictions in the reservations service in commercial carsharing providers. Services such as restaurants, theatres, and hotels allow flexible reservation options, however, one-way carsharing reservations are currently mostly limited to very short time or are at best, very expensive. As the carsharing services do not control the trips their clients will be doing, additional information about the demand gathered from the reservations is not very valuable without a reliable and sustainable way to ensure that a reserved vehicle will indeed be in the correct place at the proper time.

In this work, an innovative relocations-based reservations (R-BR) method is proposed, in which vehicles are only locked sometime before the beginning of the trip, and if a vehicle is not naturally available, one will be relocated. The author hypothesized that this method could perform well even with much higher reservation times ahead than the commercial carsharing providers offer nowadays (typically 30 min). Furthermore, optimizing the allowed reservation time ahead and the proximity of reserved vehicles to users in different areas of the city is an additional step to help tailor the system to the demand profile.

To test the performance of both types of reservations (with and without the variable quality of service), the author developed a simulator that enables evaluating various service configurations related to reservations. Several test instances have been created with daily carsharing trips in typical cities of various sizes. Problem instances include the trips in two artificial cities and a set of experiments with the case-study city of Lisbon (Portugal) where long reservation times are allowed. Results show that the vehicle locking strategy with long reservation times gives bad results for anything but a very low number of reservations (up to 20% of the total trip number). Furthermore, it is demonstrated that the proposed relocations method is able to keep the system profitable with up to 60% reservations in the Lisbon (25,000 trips) example. The R-BR method achieved better results than locking in all problem instances except the Town (500) with a very low number of trips.

Unprofitable results for low trip volumes are in line with similar conclusions by other researchers: small towns are not well suited for commercial carsharing services. In such places, the revenues are too low to allow the successful operation of commercial providers, and the successful examples are typically restricted to the volunteer-based community services [673]. Conversely, big cities have shown to be very suitable for carsharing as it is much easier to sustain a company with a high concentration of the demand in areas with high population density and high revenue from a lot of daily trips.

As a guideline to operators interested in implementing reservations, if the system is not a profitable one without reservations, offering them is risky. While this method is able to increase the customer satisfaction and the profit of already successful carsharing enterprises, in systems that are not profitable, the gains from adding reservations will most likely be very low to none, and with low trip volume, it might even cause profit decreases.

The implemented ILS metaheuristic was able to perform complex trade-offs of increasing the profit without lowering the service quality and the demand too much. It is able to learn from the performed daily trips and successfully identify areas with the highest demand, where *QoS* adjustments bring the most benefits. In general, increasing the r in the areas with the highest demand has shown to be a very effective tool that ILS used to lower relocation costs and increase profit. Further, lowering h in the areas with the highest demand can help prevent system overload and high costs of reliance on outsourcing. The flexibility of adjusting the relative importance of multiple optimization criteria allows further adjustments to the current goals of the operator.

While the objectives of this work were achieved using the above experiments, several refinements to the methodology are possible. An interesting research direction would be giving more attention to adding more realism to reservation time distribution and investing more computational resources to simulate longer periods of time, as well as supporting more realistic relocations, e.g., for companies with permanent staff in charge of relocations. Another interesting direction is investigating the performance of reservations when used with various different balancing mechanisms.

In this work, the added demand as a result of introducing the reservation system is not considered. A more detailed demand model that adjusts the demand with the *QoS* changes would further improve the accuracy of the obtained results and extend the applicability of the method to allow larger variance in the *QoS*. This approach would also allow users of the model to view more precise profit effects of each optimization variable that has been changed. Better integration of the effects of the *QoS* variation into the profit calculation is especially suitable for businesses, which are typically concerned with profit maximization as their top priority. Such demand models are usually non-linear mode choice models, nevertheless, they are suitable for integration into the simulation model, which is another advantage of the simulation-based optimization approach.

5.4 Variable pricing in one-way station based carsharing services [3]

As described in Section 5.2.2, one-way carsharing is attractive to the users, due to its flexibility and possibility to perform e.g. daily commute using carsharing. For operators, they are challenging since they promote the *vehicle stock imbalance problem*. Several approaches have been proposed in the literature to mitigate the effects of vehicle stock imbalance where the most extensively studied method is vehicle relocation [643, 646, 652, 674, 675, 676, 677]. Alternative methods include accepting or refusing a trip based on its impact on vehicle stock balance [596, 678], station location selection to achieve a more favorable distribution of vehicles [596], and price incentives for grouping people if they are traveling from a station with a shortage of vehicles, and ungrouping them otherwise [679, 680]. Regarding the use of pricing, two methods exist in the literature: price incentive policies for users to choose another drop-off station, where total demand stays unaltered [644, 650, 681], and trip pricing, that is, changing the trip prices to control the demand, taking its contribution to stock balancing into account, which is seen as a proxy for profit maximization [682, 683]. Mitchell et al. [682] and Weikl and Bogenberger [644] only suggested this as a theoretical balancing strategy and did not define ways of choosing prices and applying them in reality. In the case of the other authors, the methodological approaches mainly fall into two categories, one in which the state of stations is analyzed to determine if they are oversupplied, undersupplied or balanced, and the other where the the users' response to it is studied, by means of a simulation approach.

In this work, the goal was to help bridge the apparent research gap in the literature by proposing a method for optimising the trip prices in one-way station-based carsharing systems and show how it can be useful for profit maximization by reducing vehicle fleet imbalance. Contrary to previous studies, pricing is not used as a reactive measure but as a stable reference of the company that produces a table of prices that should be tailored to the existing consumer preferences and the operational constraints of the company. It is relevant to say that even though the trip pricing has not yet been implemented to solve carsharing imbalance problems, it has been used to solve other transportation problems, in particular with respect to: traffic congestion [684, 685, 686, 687]; high occupancy/toll lane management [688], and airline seat management [689, 690]. As in this approach, these methods also consider elastic travel demand towards price either with simple elasticity models or expressed by logit models [688, 689, 690]. Logit models are more precise but they also make it more difficult to reach an optimized solution, since they introduce a non-linear and non-convex behavior of the objective function. To address this limitation, researchers used three main methods: transforming the non-convex formulation into

[3] This section is based on the paper: "Trip pricing of one-way station-based carsharing networks with zone and time of day price variations", published in the journal "Transportation Research Part B: Methodological", copyright 2015 Elsevier Ltd.

a convex one [684, 689], for example by using the inverse of the logit model function [689], using heuristics [685], metaheuristics [687, 690], or using simulation instead of optimization [688].

The method for setting variable carsharing prices has two main components: (*i*) a mixed-integer non-linear programming (MINLP) model which, with trips made throughout an entire day and the price elasticity of demand, determines which prices to charge in a given period of time for profit maximization; (*ii*) an *iterated local search* (ILS) metaheuristic to find good solutions as fast as possible, given the non-linearity of the MINLP.

In this work, it was decided to maximize profit since carsharing is mostly provided by private companies. Nevertheless, it should be noted that maximizing profit, although not leading to the highest level of service provided to the clients, it should allow to operate a bigger network since the company is able to profit even when its market is highly imbalanced as it happens often with encompassing city center and suburban areas [691, 692].

The method is applied and tested for the case study of Lisbon, in Portugal, providing the following main scientific contributions:

1. it is the first method to set a variable trip pricing table for one-way carsharing;
2. it is a method that leads to better results than having no balancing strategy and it avoids high logistic adaptations entailed in using relocation operations.

The method assumes that the estimated value of the price elasticity of demand is known and that the mobility patterns in the network can be forecast. Elasticity is typically estimated from historical sales information and can be extracted using different statistical methods [693, 694, 695]. Transportation forecasting can also be done by simulation or modeling based on the data acquired from past network use or from surveying users. Since there is ample research on this separate subject [696, 697], a detailed description of the techniques for demand modeling is beyond the scope of this work.

5.4.1 The trip pricing problem for one-way carsharing systems (TPPOCS)

The problem of trip pricing for one-way carsharing system is defined as a problem of setting prices for carsharing trips based on their origin, destination and the time of day. Given a set of carsharing stations operating in one-way mode for which an origin-destination matrix is known for a given reference price, the TPPOCS aims at finding new prices between groups of stations during a working day such that the profit of running the system is maximized while satisfying all demand.

Let us define the following notation.

Sets: $\mathrm{I}'=\{1,\ldots,\mathrm{i}\ldots\mathrm{I}\}$: The set of time intervals in the operation period.

$\mathrm{Z}'=\{1,\ldots,\mathrm{z}\ldots\mathrm{Z}\}$: The set of zones.

Parameters:

$\alpha^0_{k_t}$: Number of vehicles relocated to station k at time instant t when relocation costs are $0, \forall k_t \in \boldsymbol{X}$.

$\beta^0_{k_t}$: Number of vehicles relocated from station k at time instant t when relocation costs are $0, \forall k_t \in \boldsymbol{X}$.

$\varepsilon^0_{k_t}$: Difference in the number of vehicles relocated to/from station k at time instant t when relocation costs are 0, $\forall k_t \in \boldsymbol{X}$.

tb_i: The beginning instant of time interval i, $\forall\, i \in \boldsymbol{I'}$.

te_i: The end instant of time interval i, $\forall\, i \in \boldsymbol{I'}$.

$\omega^0_{k_i}$: Difference in the number of vehicles relocated to/from station k during time interval i when relocation costs are 0, $\forall k \in \boldsymbol{K'}$, $\forall i \in \boldsymbol{I'}$.

o: Number of observations in the cluster analysis.

u: Number of clusters desired.

E: Price elasticity of demand.

$\mathrm{P0^i_{zw}}$: The current carsharing price per time step driven between zones z and w when departure time interval is $i, \forall z, w \in \boldsymbol{Z'}, i \in \boldsymbol{I'}$ (all prices set to $P0$).

Decision variables:

$\mathrm{D_{k_t j_{t+\delta^t_{kj}}}}$: Number of customer trips from station k to station j from time instant t to $t + \delta^t_{kj}, \forall \left(k_t, j_{t+\delta^t_{kj}}\right) \in \boldsymbol{A_1}$ after the price is varied.

P^i_{zw}: Carsharing price per time step driven between zones z and w when departure time period is $i, \forall z, w \in \boldsymbol{Z'}, i \in \boldsymbol{I'}$.

Demand, in this model, varies according to a simple elastic behavior. The new demand $\left(D_{k_t j_{t+\delta^t_{kj}}}\right)$ results from applying the price elasticity E to a reference demand $\left(D0_{k_t j_{t+\delta^t_{kj}}}\right)$ that exists for price $P0$. The expression is:

$$E = \frac{\frac{\mathrm{D_{k_t j_{t+\delta^t_{kj}}}} - \mathrm{D0_{k_t j_{t+\delta^t_{kj}}}}}{\mathrm{D0_{k_t j_{t+\delta^t_{kj}}}}}}{\frac{\mathrm{P^i_{zw}} - \mathrm{P0^i_{zw}}}{\mathrm{P0^i_{zw}}}} \tag{5.11}$$

The model assumes the elasticity to be the same for any interval of price variation. This may be unrealistic for large price variations, however, one does not expect to change prices beyond a realistic interval around the current reference price P0.

Using the notation presented in the previous sub-sections and the elasticity defined in Equation (5.11), the MINLP model is formulated as follows:

$$Max\ \theta = \sum_{\substack{k_t j_{t+\delta^t_{kj}} \boldsymbol{\varepsilon A_1} \\ z,w \in \boldsymbol{Z'} \\ i \in \boldsymbol{I'}}} \left(P^i_{zw} - C_{mv}\right) \times D_{k_t j_{t+\delta^t_{kj}}} \times \delta^t_{kj} - C_{mp} \sum_{k \in \boldsymbol{K'}} Z_k - C_v \sum_{k \in \boldsymbol{K'}} a_{k_1} \tag{5.12}$$

subject to,

$$\mathrm{D}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} \geq \mathrm{D0}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} + \frac{\mathrm{E} \times \mathrm{D0}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} \times \left(\mathrm{P^i_{zw}} - \mathrm{P0^i_{zw}}\right)}{\mathrm{P0^i_{zw}}} - 0.5\ ,\ \forall \left(\mathrm{k_t, j_{t+\delta^t_{kj}}}\right) \varepsilon \mathrm{A_1},\ \mathrm{z,w} \in \mathrm{Z'},\ \mathrm{i} \in \mathrm{I'} \tag{5.13}$$

$$\mathrm{D}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} \leq \mathrm{D0}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} + \frac{\mathrm{E} \times \mathrm{D0}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} \times \left(\mathrm{P^i_{zw}} - \mathrm{P0^i_{zw}}\right)}{\mathrm{P0^i_{zw}}} + 0.5\ ,\ \forall \left(\mathrm{k_t, j_{t+\delta^t_{kj}}}\right) \varepsilon \mathrm{A_1},\ \mathrm{z,w} \in \mathrm{Z'},\ \mathrm{i} \in \mathrm{I'} \tag{5.14}$$

$$\mathrm{D0}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} + \frac{\mathrm{E} \times \mathrm{D0}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} \times \left(\mathrm{P^i_{zw}} - \mathrm{P0^i_{zw}}\right)}{\mathrm{P0^i_{zw}}} \geq 0 \tag{5.15}$$

$$\mathrm{V}_{\mathrm{k_t k_{t+1}}} + \sum_{\mathrm{j} \in \boldsymbol{K'}} \mathrm{D}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} - \sum_{\mathrm{j} \varepsilon \boldsymbol{K'} : \mathrm{t'} = \mathrm{t} - \delta^{\mathrm{t}}_{\mathrm{jk}}} \mathrm{D}_{\mathrm{j_{t'} k_t}} - \mathrm{V}_{\mathrm{k_{t-1} k_t}} = 0\ \ ,\forall \mathrm{k_t} \in \mathrm{X} \tag{5.16}$$

$$\mathrm{a}_{\mathrm{k_t}} - \sum_{\mathrm{j_t} \in \mathrm{X}} \mathrm{D}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} - \mathrm{V}_{\mathrm{k_t k_{t+1}}} = 0\ \ \ ,\forall \mathrm{k_t} \in \mathrm{X} \tag{5.17}$$

$$\mathrm{Z_k} \geq \mathrm{a}_{\mathrm{k_t}}\ \ \ \ ,\forall \mathrm{k_t} \in \mathrm{X} \tag{5.18}$$

$$\mathrm{D}_{\mathrm{k_t j_{t+\delta^t_{kj}}}} \in \mathbb{N}^0\ ,\forall \left(\mathrm{k_t, j_{t+\delta^t_{kj}}}\right) \varepsilon \mathrm{A_1} \tag{5.19}$$

$$P^i_{zw} \in \mathbb{R}^0\ ,\forall \mathrm{z,w} \in \mathrm{Z'},\ \mathrm{i} \in \mathrm{I'} \tag{5.20}$$

$$\mathrm{V}_{\mathrm{k_t k_{t+1}}} \in \mathbb{N}^0\ ,\ \forall \left(\mathrm{k_t, k_{t+1}}\right) \in \mathrm{A_2} \tag{5.21}$$

$$\mathrm{a_{k_t}} \in \mathbb{N}^0 \ , \forall \mathrm{k_t} \in \mathrm{X} \tag{5.22}$$

$$\mathrm{Z_k} \in \mathbb{N}^0 \ , \forall \mathrm{k} \in \mathrm{K}^{'} \tag{5.23}$$

The objective function (5.12) maximizes the total daily profit (θ) of the one-way carsharing service, taking into consideration the revenue from trips paid by the clients, vehicle maintenance costs, vehicle depreciation costs, and station maintenance costs. Note that in this model no relocations are considered. Constraints (5.13) and (5.14) compute the demand resulting from considering the price change. Given that this demand is a continuous function of price, two inequalities are used to ensure that D will be integer. Constraints (5.15) ensure that the demand resulting from the application of price elasticity to the reference demand is positive. Constraints (5.16) and (5.17) ensure the flow conservation and calculate the number of vehicles at each station at each time instant. Constraint (5.18) guarantees the station capacity constraint is met. Expressions (5.19)-(5.23) set the variables domain.

The decision variables of the model are: the number of vehicles in each station at the beginning of the day, the demand for each OD pair of stations at each time step, and the prices charged for each OD pair of zones per time interval. It is evident that the objective function (5.12) is non-linear because demand is multiplied by the price and is non-concave, which makes this a MINLP problem not easily solvable by traditional branch and cut algorithms. Some MINLP solver software solutions are available to solve this type of problem for both concave and non-concave formulations, but these typically have difficulties managing real size problem instances [698]. The size of the search space of the trip pricing problem is much greater than these solvers are able to tackle. For only 5 zones and 6 time periods, if prices vary from 0 to 0.70 €/min, with 0.01 increments, the number of possible solutions for this problem would be $|\mathcal{P}|=71^{5\cdot5\cdot6}= 4.88\cdot10^{277}$.

5.4.2 Solution algorithm

The goal of the solution algorithm presented in this section is to find the prices P^i_{zw} for which the daily profit θ of the TPPOCS will be as high as possible. A solution of this problem is a set of trip pricing tables denoted $P\ [|I|]\,[|Z|]\,[|Z|]\ (p_{min},\ p_{max})$, or in short $P\ (p_{min},\ p_{max})$, where $|I|$ is the number of time intervals, $|Z|$ is the number of zones and p_{min} and p_{max} are the minimum and maximum allowed prices, respectively. Pricing table $P\ (p_{min},\ p_{max})$ contains $(|I|\cdot|Z|\cdot|Z|)$ individual elements and each element P^i_{zw} corresponds to the price charged per minute for trips from any station in zone z to any station in zone w, starting in time interval i. The set of feasible solutions $\mathcal{P}(p_{\min},\ p_{\max})$ is defined as the set of all possible trip pricing tables of appropriate dimensions $(|I| \times |Z| \times |Z|)$ whose elements are in a given price interval.

The optimal pricing table $\overline{P}$ is a trip pricing table for which the daily profit (θ) of a carsharing company is maximised. More formally, the goal of the optimization algorithm is to find $\overline{P}$ for which the following equation is satisfied:

$Max\ \theta\left(\overline{P}\right) \geq Max\ \theta\ (P),\ \forall P\ (p_{min},\ p_{max}) \in \mathcal{P}\,(p_{min},\ p_{max})$.

For each trip pricing table $P(p_{min},\ p_{max})$ generated by the solution algorithm, the TPPOCS mathematical model is executed as a classical mixed integer programming (MIP) problem where prices are given. In that way, the TPPOCS model finds the best possible profit that can be achieved using a fixed trip pricing table suggested by the solution algorithm. The best possible profit value is then introduced back to the algorithm, in essence rendering the model an evaluator for the solutions suggested by the algorithm.

For this problem, again, the *iterated local search* (ILS) metaheuristic is selected. The algorithm outline is available in Section 3.6.3. Various options are available when deciding on the end condition for the algorithm and the local search operator. In the numerical experiments, the time limit, which can be set as a parameter is used as the end condition. Furthermore, different solution acceptance criteria can be chosen for the perturbation operator: starting each perturbation from the best so far, from the current local search result or some other solution found during the algorithm run history. The algorithm runtimes for the numerical experiments, as described in Section 5.4.4, are very long, therefore, it was decided to focus the search as much as possible, always using the best known solution as the perturbation starting point. In the ILS literature, such acceptance criterion is usually called the *best* acceptance criterion.

Initial solutions

Initial solutions $P_{initial}$ are randomly generated trip pricing tables with each element $P^{i}_{zw} \in P_{initial}$ in the interval p_{min}, p_{max}. They are obtained using a random number generator that generates numbers with approximately uniform distribution in the specified interval. Preliminary tests have shown that the quality of initial solutions varies significantly, depending on the choice of the price interval, therefore this interval is subject to tuning. Such tuning considerably increased the algorithm performance, similarly to the initial point selection in for the reservation problem.

Local search operator

The *local search operator* (LSO) used in this approach is explained in the pseudo-code in Algorithm 2. It is a simple method that iteratively increases and then decreases trip pricing table elements, as long as these changes improve profit. The procedure has two parameters: *step* and *time*. The *step* parameter defines the smallest change in price that can be made during the search and the *time* parameter defines the longest allowed search duration. The interval in which the local search can modify the solutions isp_{min} and p_{max}. The price interval should be selected as a reasonable interval for the problem at hand.

The order in which table elements are modified is randomized to encourage the discovery of features for which the order of price changes matters, thus the operator is non-deterministic. For each considered element of the table, the operator first tries to increase the price by adding $step$ to the initial price. If the modification causes a better profit, further increases will be performed until p_{max} is reached or price increases are no longer improving the profit (or the time expires). The analogous procedure is followed for price decreases. After the benefits of both increasing and decreasing the price have been examined, the algorithm updates the trip pricing table accordingly so that the new value gives the highest profit gain or retains the old value if price changes caused a profit drop. The subset of the prices to consider changing can vary, but in this work, the author decided to search through the entire table, i.e. the changing candidate set of prices for the LSO, denoted C_{ls} is a set of all table elements. After all of the elements of C_{ls} have been considered for modification, the operator will start again, but it will visit the elements in a new randomly generated sequence.

The above procedure continues until on entire pass through the table has been done without any improvements being made or until the allowed time has elapsed. By systematically exploring the effect of the price variations and combining the contributions of many small price changes, the LSO can yield significant solution improvements, as shown in the numerical application.

Some additional notation used in Algorithm 16:

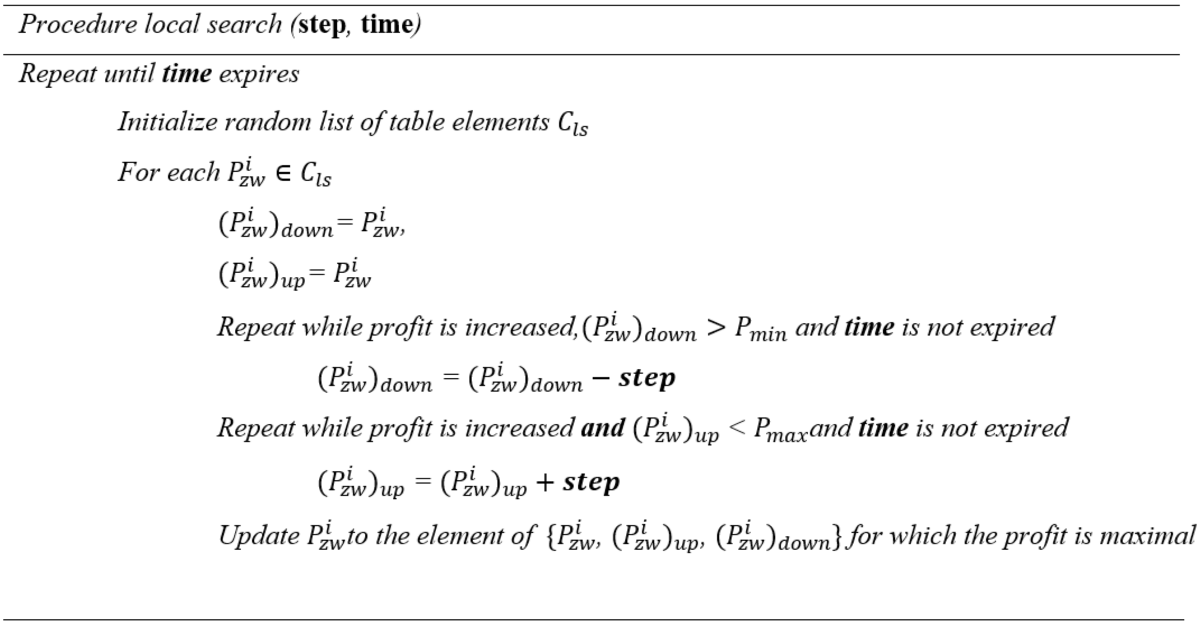
$(P^i_{zw})_{down}$: Potential new lower price considered by the LSO and calculated by lowering the initial price P^i_{zw}.

$(P^i_{zw})_{up}$: Potential new higher price considered by the LSO and calculated by increasing the initial price P^i_{zw}.

Perturbation operator

The perturbation operator (PO), presented in Algorithm 17, introduces random price changes in a small subset of the price table elements. The operator has two parameters: maximum number of elements to change (n) and the maximum allowed change (d). First, n elements from a price table $P(p_{min}, p_{max})$ are randomly selected into the perturbation modification candidate set $C_p \subseteq P(p_{min}, p_{max})$. Then, for each element $P_(z,w)^i \in C_p$, the new price is calculated by adding a random value $\Delta p \in [-d, d]$ to the previous value. The interval, in which the perturbation can change the prices, varies between p_{min}, and p_{max}. If the price after adding Δp is lower than p_{min}, it is updated to p_{min}, and likewise, if it is greater than p_{max}, it is updated to p_{max}. The set C_p is called the modification candidate set, since there is no guarantee that all of its members will be changed. Due to the definition of the interval from which Δp values are selected, it is

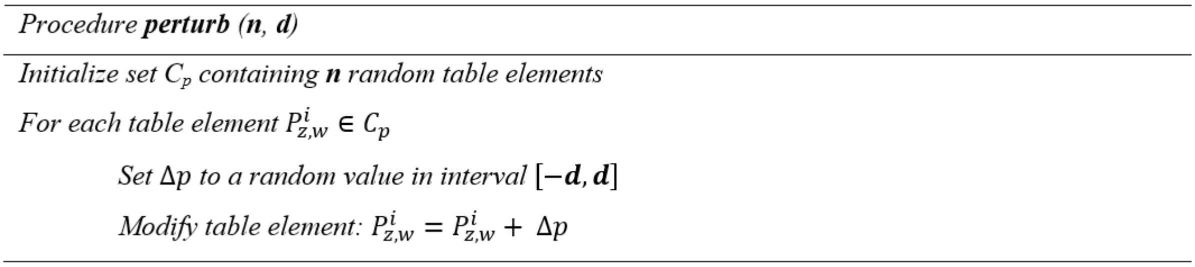
Algorithm 16 The local search operator [181]



```
Procedure local search (step, time)
Repeat until time expires
    Initialize random list of table elements $C_{ls}$
    For each $P^i_{zw} \in C_{ls}$
        $(P^i_{zw})_{down} = P^i_{zw}$,
        $(P^i_{zw})_{up} = P^i_{zw}$
        Repeat while profit is increased, $(P^i_{zw})_{down} > P_{min}$ and time is not expired
            $(P^i_{zw})_{down} = (P^i_{zw})_{down} - \mathbf{step}$
        Repeat while profit is increased and $(P^i_{zw})_{up} < P_{max}$ and time is not expired
            $(P^i_{zw})_{up} = (P^i_{zw})_{up} + \mathbf{step}$
        Update $P^i_{zw}$ to the element of $\{P^i_{zw}, (P^i_{zw})_{up}, (P^i_{zw})_{down}\}$ for which the profit is maximal
```

possible that for some elements Δp will be zero, leaving the table elements unchanged. This behavior is intentional to ensure greater variability of the perturbation effects on a candidate solution. It should be noted that LSO and PO are structurally related in such a way that the local search is unlikely to cancel the effects of perturbation. If step is greater than 0.01 €/min, the local search can return to the previous local optimum only if all of the changes caused by the perturbation are multiples of the search step value. The probability for such an event to occur drops very quickly as d grows in comparison to step and $n > 1$. Nevertheless, finding a balanced perturbation intensity is still very important, to ensure that it is not too strong, as shown in the numerical application in Section 5.

Algorithm 17 The perturbation operator [181]



```
Procedure perturb (n, d)
Initialize set $C_p$ containing n random table elements
For each table element $P^i_{z,w} \in C_p$
    Set $\Delta p$ to a random value in interval $[-\mathbf{d}, \mathbf{d}]$
    Modify table element: $P^i_{z,w} = P^i_{z,w} + \Delta p$
```

5.4.3 Lisbon case study

The case study used in this work is the municipality of Lisbon, in Portugal. This municipality has been dealing with several mobility problems, including traffic congestion and shortage of parking space associated with the increase in car ownership and the consequent high use of private transport. Public transport has been upgraded; however, it has not been able to reduce the use of private transportation for commuter trips. There is a need to manage mobility in a smart way by, for instance, encouraging transport alternatives such as carsharing.

The base carsharing price per minute, P0, was considered to be 0.3 euros per minute, which is based on the rates of Car2go from 2014 [639]. Note that there is no linkage between this price and the demand that is going to be used for the computational experiments, since carsharing was not offered in Lisbon at the time this research was conducted (2014). Time was divided into 6 intervals: 6:00 a.m. to 8:59 a.m., 9:00 a.m. to 11:59 a.m., noon to 2:59 p.m., 3:00 p.m to 5:59 p.m., 6:00 p.m. to 8:59 p.m, and finally from 9:00 p.m. to midnight, and the stations were grouped into 5 zones. Note that zones are not necessarily geographically distributed, since clustering based on demand patterns similarity was used to group the stations into these categories [181].

5.4.4 Running the experiments

The TPPOCS mathematical model was also implemented using Xpress 7.7 with the same data that was used in the VRPOCS model. The ILS metaheuristic was implemented in Java 1.8 programming language and Xpress Java Application Programming Interface to gain access to the model. All experiments were performed on two identical computers equipped with a 2.4 GHz Intel Core i7-4700HQ processor and 16 GB of RAM and using Java 1.8 runtime environment under Windows 8.1 operating system.

A single run of the TPPOCS model takes around 30 seconds and approximately one minute when 8 instances of the model are running simultaneously. Most of the algorithm runtime is therefore spent evaluating the candidate solutions; the benchmarks have shown that, in the best case, only around 430 evaluations could be performed in one hour, using all of the eight logical processor cores in parallel. This fact strongly influenced the tuning process of the algorithm as well as the algorithm design itself. To obtain good solutions as quickly as possible, strong intensification is performed through detailed local search and the use of the best perturbation acceptance policy [127]. The strong intensification is introduced to facilitate the discovery of profit-increasing features as soon as possible. The rest of this section gives an overview of the tuning process and the best results found by the algorithm are presented in the next section.

Parameter tuning

The parameter tuning for this problem was done in three stages:

- Initial solution generator tuning,
- Local search tuning,

- Perturbation tuning.

Initial solution generator tuning consisted of determining price bounds p_{min} and p_{max} for the initial solutions. Initial solution tuning rationale is based on the assumption that good initial solutions will enable the local search to find better results more quickly. In total, 105 different intervals were explored: each interval with p_{min} and p_{max} being a multiple of 0.05€/min in the range $[0.00; 0.70]$ €/min, with 50 solutions generated in each of these intervals. The results proved that the average daily profit for the initial solutions varied greatly depending on the price interval. The worst profit was measured for the interval $[0; 0.05]$ €/min, (average deficit of 16,714.2 €/day) and the best profit, zero, was achieved for any interval with p_{min} and p_{max} above 0.5 €/min. While at first this might seem like a good result, it should be pointed out that the simulated carsharing demand adapts to price. When unreasonably high prices are applied, the demand drops to zero by force of the elasticity. Zero demand causes the model to shut down the service, interpreted as a zero profit result.

To take this into account, both profit and demand were considered for the initial trip pricing tables. The scatter plot of a subset of explored initial intervals can be seen in Fig. 5.6, with average profit on the x-axis and average demand on the y-axis. The Pareto non-dominated set of (profit, demand) pairs, indicated by grey dots on the chart, was selected as a set of candidate intervals [699, 700]. The highlighted interval $[0.35; 0.40]$ €/min has the lowest average deficit (32.48 €/day) while also retaining high average demand (1579 trips, which corresponds to 89 % of the demand with the reference price). Configurations with higher profit do exist, but for them, the demand drops to nearly zero, as can be seen in the example of the interval $[0.45; 0.65]$ €/min, indistinguishable from the zero demand interval $[0.5; 0.7]$ €/min. Having near-zero demand in the system is clearly not the desired result. Therefore, the values of p_{min}=0.35 €/min and p_{max}=0.40 €/min are selected as the initial randomly generated solutions.

The local search tuning consisted of determining the best search step parameter. Five initial solutions with price intervals set up according to the values given above were randomly selected and for each of them, local search with step equal to €0.01, €0.02, €0.05 and €0.10 was applied. The experiment was repeated five times, resulting in 100 test runs with each local search being limited to run for 4 hours. It is assumed that a better functioning local search will provide good results faster in the environment of the ILS. The best results were in general achieved using a step of €0.02.

Perturbation tuning consisted of trying to identify which pair of changed set size and intensity (n, d) works best with the LSO configured as in the previous stage. A total of 8 different configurations were run 5 times for 12 hours, to determine which perturbation parameters best fit the balance of diversification and intensification. As the preliminary tests have shown, the model is very sensitive to price changes. Best average profits are achieved with low perturbation intensities (n=2,d=0.02) and $(n = 5, d = 0.02)$, which have almost equal average profit. If

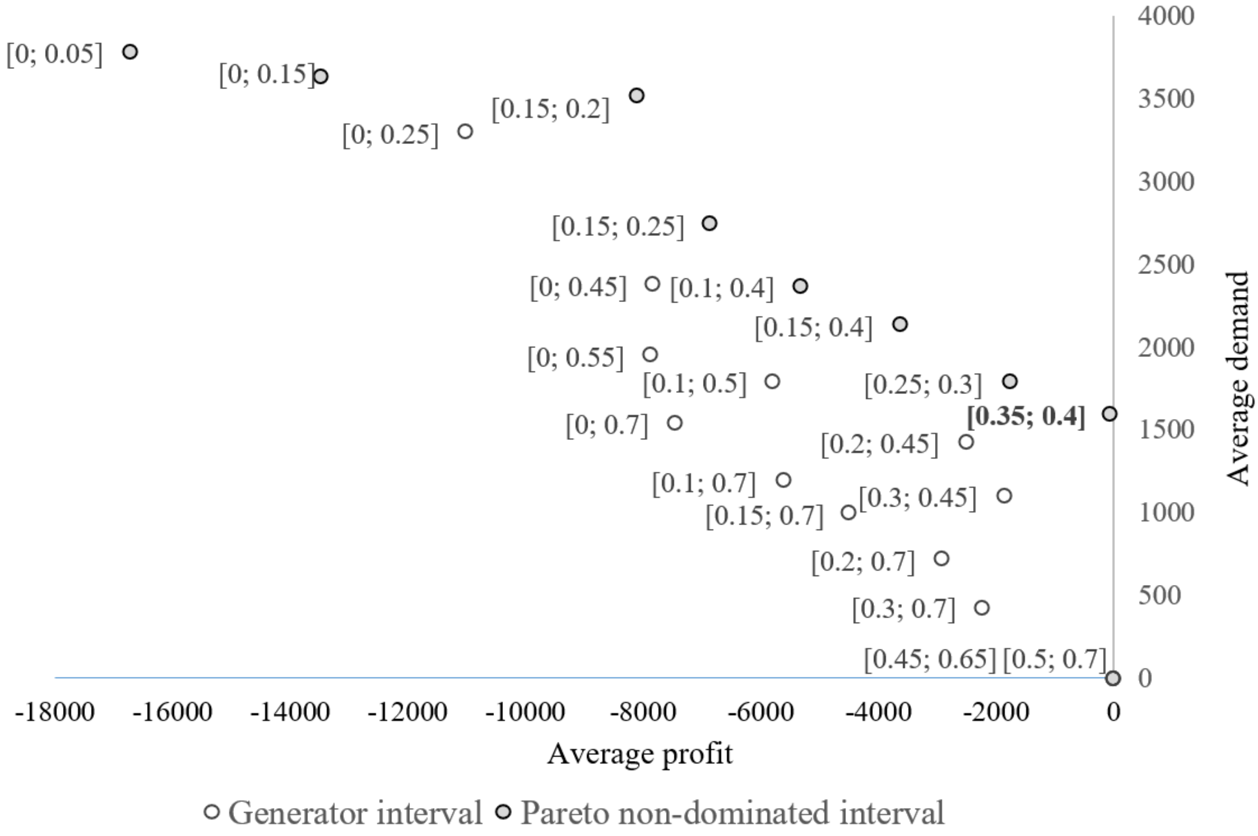


Figure 5.6: Initial solution generator tuning [181]

Table 5.13: Recommended ILS parameters [181]

	Initial solution		*step* (€)	*n*	*d*
	P_{min} (€)	P_{max} (€)			
Short runs (≤ 12h)	0.35	0.40	0.02	2	0.02
Long runs (>12h)	0.35	0.40	0.02	10	0.02

perturbation is more pronounced, average profits fall. This can be explained by the fact that the local search is unable to find good solutions before another intensive round of perturbation reduces the profit of the current local optimum.

The recommended metaheuristic parameters for solving the TPPOCS, using Lisbon as a case study are given in Table 5.13. The time limits in the table are valid for equipment with similar performance to the experimental setup from this work. When using the hardware as specified in this Section, around 1440 model evaluations is available in 12 hours.

Best found solution

The best trip pricing table found in the experiments, denoted P_{best}, is presented in Table 5.14. Using this pricing table, the system is able to achieve a profit of 2068.1 €/day. Compared to

Table 5.14: Best found trip prices for each origin-destination pair of zones and time interval [181]

TI 1 (6:00 a.m.-8:59 a.m.)						**TI 2** (9:00 a.m.-11:59 a.m.)						**TI 3** (midday-2:59p.m.)					
Zones	1	2	3	4	5	Zones	1	2	3	4	5	Zones	1	2	3	4	5
1	0.36	0.44	0.38	0.40	0.38	1	0.35	0.43	0.38	0.41	0.40	1	0.40	0.39	0.39	0.40	0.41
2	0.38	0.39	0.39	0.38	0.39	2	0.39	0.39	0.39	0.39	0.39	2	0.38	0.39	0.38	0.39	0.39
3	0.46	0.37	0.38	0.44	0.38	3	0.38	0.38	0.39	0.38	0.39	3	0.39	0.39	0.39	0.39	0.39
4	0.35	0.41	0.36	0.38	0.39	4	0.39	0.40	0.39	0.41	0.40	4	0.38	0.40	0.38	0.39	0.39
5	0.39	0.45	0.35	0.41	0.39	5	0.38	0.39	0.38	0.36	0.38	5	0.39	0.39	0.39	0.39	0.39
TI 4 (3:00p.m.-5:59p.m.)						**TI 5** (6:00p.m.-8:59p.m.)						**TI 6** (9:00p.m.-midnight)					
Zones	1	2	3	4	5	Zones	1	2	3	4	5	Zones	1	2	3	4	5
1	0.39	0.43	0.39	0.39	0.38	1	0.39	0.38	0.39	0.38	0.45	1	0.39	0.36	0.38	0.38	0.36
2	0.38	0.38	0.38	0.38	0.39	2	0.39	0.38	0.38	0.38	0.39	2	0.40	0.39	0.40	0.35	0.40
3	0.39	0.39	0.38	0.38	0.39	3	0.36	0.39	0.40	0.38	0.38	3	0.38	0.36	0.39	0.38	0.39
4	0.39	0.39	0.39	0.37	0.46	4	0.38	0.40	0.39	0.39	0.38	4	0.35	0.35	0.38	0.36	0.36
5	0.39	0.39	0.38	0.38	0.35	5	0.39	0.38	0.41	0.38	0.39	5	0.39	0.38	0.40	0.38	0.38

the loss of 1160.7 €/day resulting from not implementing any balancing strategy and having to satisfy all the reference demand (1777 trips) for the reference price, it is clear that variable pricing can lead to significant profit increases. For the best trip pricing table found, satisfied demand is 1471 trips per day, which is a loss of 306 trips in relation to the reference demand (17.7 % demand reduction). Note, however, that this demand is not rejected per se; it indicates that some travelers will find the price too high to use the carsharing service. The average price charged is 0.39€/min, with all the prices being in the interval [0.35 ; 0.46] €/min, that is, all prices charged are higher than the reference carsharing price ($P0$), which is 0.30€/min. Most of the OD pairs of zones that have a significant fall in demand correspond to a price of 0.40 €/min or higher. It should be mentioned that the elasticity is being applied to the unit price per trip and not to the total price of a trip; thus, it is not considered that longer trips may give different results from shorter ones.

The improvement in the profitability of the company is not only due to the decreased demand for some OD pairs of zones and time intervals, it is also due to the price increase itself in many OD pairs where, while it is not enough to produce an expected demand reduction, it is sufficient to have an impact on increasing the profits. This occurs even though it is considered in the case study that demand is elastic to price variations, and elasticity is greater than 1 (absolute value), which should point to a reduction in profit from price increase in a linear model. The special and complex interdependence of supply and demand in carsharing systems leads to a system that is beneficial when run for a lower number of trips, yet one that is more balanced.

Table 5.15: Global results with and without trip pricing [181]

	Profit (€/day)	Revenue related to the trips (€/day)	Costs of vehicle maintenance (€/day)	Costs of vehicle depreciation (€/day)	Costs of parking spaces maintenance (€/day)	Satisfied demand	Fleet of vehicles	Number of parking spaces
No balancing strategy	-1160.7	7113.3	166.0	6630	1478	1777	390	739
Trip pricing	2068.1	7576.4	138.3	4352	1018	1471	256	509

Zoning that was determined by computing a theoretical desired relocation vector is able to divide the stations into sets for which the price variations yield a higher profit. Even though using the metaheuristic does not guarantee the optimal solution will be found, it can still be demonstrated through its application to the case study that an increase in prices can actually lead to a higher profit; a solution that not only avoids losses (system closure will generate 0 profit) but that is able to generate positive and significant profits. From the global results presented in Table 5.15 it can be concluded that the balance of vehicles and profit are directly related, because a more balanced system, despite resulting in lower revenue, requires fewer vehicles and fewer parking spaces, which means lower operating costs.

5.4.5 Notes for potential practitioners – precision vs. simplification

When deciding on the number of clusters and time intervals for modeling a new city, they should be sufficiently large to faithfully model the mobility patterns and classify the stations in enough detail, but not so big as to diminish metaheuristic performance. For example, if a city has rush hours in the morning (around 8:00 a.m.) and in the afternoon (around 4:00 p.m.), dividing the day in two intervals (7:00 a.m. - 7:00 p.m.) and (7:00 p.m. - 7:00 a.m.) is clearly a bad choice. Because both rush hours happen in a single time interval (the first one), this oversimplification will cause the system to treat them both the same, despite the fact that they typically move in opposite directions. The insufficient detail level of only two time intervals and a bad choice of the interval start will therefore be likely to prevent the system from efficiently taking advantage of differences in mobility patterns in the morning and afternoon rush hours. Similar reasoning can be given for the cluster detail level.

The number of clusters and time intervals only has to be enough to allow the algorithm to do detailed decisions that take advantage of the user behavior in the best possible way. Despite the greater precision it brings, making the number of clusters too high may not be beneficial. The number of possible candidate solutions increases exponentially the higher the number of clusters and time intervals. Too many clusters and intervals cause the metaheuristic to converge rather slowly due to the increased computational complexity arising from the combinatorial explosion of the search space size. The number of clusters and time divisions can be a part of the metaheuristic tuning as well, if the initial results with the predefined values are not satisfactory.

In [181], a bound analysis formulas and models are provided for the TPPOCS problem. They can be highly useful guidelines to arriving at educated decisions while implementing this methodology. If the solutions found by the metaheuristic algorithm are far from the optimum, or even lower than the lower optimum bound, this can be a good indicator that a tuning procedure, although lengthy, might be beneficial to the system. Likewise, if the profit is closer to the upper optimum bound (which is clustering independent), it is a sign that the metaheuristic is approaching the upper limit of its capabilities. In these cases, further tuning and increases in the cluster number will almost certainly produce little or no improvement.

5.4.6 Variable pricing - concluding remarks

The case study application shows that using price variation to balance vehicle stock across one-way carsharing stations works satisfactorily. When no vehicle balancing mechanism is applied, the carsharing company has a deficit of 1160.7 €/day. In a perfectly balanced solution, a profit of 316.0 €/day is achieved. Using the trip pricing metaheuristic approach, the profit for the best price combination found through the use of the ILS is 2068.1 €/day. This is an increase of 3228.8 €/day over the case of no balancing mechanism in a system that has 75 stations and serves 1471 trips. It is demonstrated that system balancing has a very important role in reducing the costs and increasing profitability of carsharing systems. The perfectly balanced solution has a higher profit than the imbalanced one.

Still, as shown in [181], balancing alone is not enough to achieve optimal profits. Using a metaheuristic algorithm to optimize profits, it was possible to find solutions offering more than six times higher profit than a perfectly balanced solution. While the best solution found also has significantly decreased imbalance, it is not zero.

It is also relevant to note that the prices charged to the clients for every OD pair of zones increased in comparison to the reference price, which leads to lower demand. However, the increase in price happens through a generalized reduction in the imbalanced demand served by carsharing. The results show that in most cases the OD pairs of zones that have price increases, and therefore a decrease in demand, are the ones with a greater difference between trip origins and trip destinations which demonstrates that the solution algorithm (metaheuristic) is able to capture the essential behavior of the system related to the trip imbalance across zones and time intervals and improve its performance. Additionally it is quite interesting to observe that even though a notable elastic behavior of demand toward price (-1.5) was introduced in the model the general increase of prices produces a higher profit, which in a simpler case of one demand for one price is not intuitive, but in this case is the result of a complex system with multiple feedbacks that characterizes carsharing.

The main conclusion that is drawn from this study is that trip pricing can be considered an effective method to improve the profitability of one-way carsharing systems. Concerning the generalizability of the method, it was shown that it is possible to successfully apply it under variability such as changes in the way stations are divided into clusters, or changes in consumer habits that impact price elasticity over time [181]. Therefore, the methodology is robust enough to work reasonably, even if some of the real world parameters change. Additionally, the meta-heuristic is a generic optimization tool that could be used with any other traffic simulator that is able to estimate profit based on a variation in pricing, and is not restricted to the elasticity model used in this Chapter. This indicates that the developed methods could be applied in various other environments. However, it must be noted that the solution algorithm computation time is dependent on the problem dimension.

Chapter 6

Metaheuristics for problems with limited budget of evaluations – lessons learned

Several general conclusions can be taken from the described experience implementing metaheuristics for problems with a limited budget of evaluations. Caution is needed when attempting to generalise any experiments with metaheuristics, especially since the two studied problems are similar and come from a specific area of carsharing optimisation. While these restrictions hold, the fact that the devised algorithms are first implementations of the iterated local search metaheuristic for such type of problems can provide important insight for other researchers interested to applying this metaheuristic to problems with limited budget of evaluations.

Further, not using a surrogate, but instead focusing on improving the performance of the "naked" metaheuristic is a rare approach in the current literature. Existing research typically suggests that the best course of action is building appropriate surrogate methods and developing a sampling strategy that balances the quality of the surrogate and the focus around areas with good potential objective function value. Despite this in the two successful implementations of the ILS this step was not necessary. In this chapter, the benefits and drawbacks of both using and not using surrogate models are discussed.

The author believes that some of the seemingly simple techniques such as initial solution generator tuning, and more generally a high quality initial solution generation technique can bring great improvements in the algorithm performance. In fact, the tuning of the initial solution generator was the step that has brought the highest performance improvements during the development. While this analysis is restricted to ILS, there is no reason to suspect that improving the initial solutions would also improve performance of any other metaheuristic, when very low number of solutions can be sampled. In the remainder of this chapter, the most important findings are described, followed by guidelines based on the conducted experiments.

6.1 Initial solution generation

When applied on problems that have an objective function that can be quickly evaluated, metaheuristics can perform large number of solution evaluations, and evolve a solution to a highly optimised one. This is possible even with very small changes in each step because it is not uncommon to evaluate and compare hundreds of thousands of solutions before returning the best found. As described in the introductory chapters of this thesis, the progress of a metaheuristic technique can be visualised as a path of a point (when single-soution methods are used) or multiple paths of multiple points (when a population method is used) in the solution hyperspace. With quick objective functions, the large number of evaluated solutions corresponds to a long path the algorithm can perform in the solution space, before finishing. This also indicates that with such problems, the choice of the initial point does not matter much as any other point in the solution space will be reachable by the algorithm in the provided time. Because of that, for most problems with fast evaluation functions, even a trivial random solution initialisation works well enough.

This does not apply to the problems with a slow evaluation function, where only a few hundreds, up to a few thousands of points can be sampled and evaluated. Algorithms applied to such problems also traverse a path in the solution space, however it is much shorter than with a fast objective function. This also means that it is less likely that an algorithm will reach a region with high quality solutions if the initial solution (or solutions) are very distant from the initial point. This fact is mentioned e.g. in [594], where the authors mention that the choice of the initial point is not highly important for the quality of the final solution (in implied situation when it is possible to perform a lot of evaluations). However, it is specifically stated that the choice of the initial point is important if good solutions need to be found quickly. Unfortunately, the literature related to solving optimisation problems with slow objective function barely mentions this simple approach.

Experiments conducted in this work show that choosing good starting points significantly improves the solutions. Good solutions tend to be clustered, therefore being in an already not-too-bad solution increases the probability that we are indeed close to a good region of the solution space. For this purpose, several techniques were considered: tuning the initial solution random generator, implementing a greedy algorithm, and deducing construction heuristics based on experiments performed by hand. In the two applications from this work, the first approach performed well enough, therefore the more advanced ones were skipped.

6.1.1 Tuning the parameters of the random solution generator

The initial experiments for the variable trip pricing problem in 5.4 indicated that a very simple configuration of the interval in which the prices are generated can have a considerable effect on the inital solution quality. In that example, instead of generating each price in the general interval $p \in [0.00, 0.70]$, the initial solutions used a restricted version where all initial prices were set to a random value between 0.35€ and 0.40€ was used.

This interval was selected after running a simple experiment in which the average objective function value was calculated for 50 randomly initialised solutions with the variable versions in the provided interval. A total of 105 different intervals was used, with 0.05 € as the step value and both the upper and lower limit being divisible by 5 cents. This resulted in experimentally tested average solution quality for each of the intervals below

$$\begin{aligned}
&[0.00, 0.05], [0.00, 0.10], [0.00, 0.15], \ldots, [0.00, 0.70]\\
&[0.05, 0.10], [0.05, 0.15], [0.05, 0.20], \ldots, [0.05, 0.70]\\
&\ldots\\
&[0.60, 0.65], [0.60, 0.70],\\
&[0.65, 0.70]
\end{aligned} \tag{6.1}$$

Such tuning is simple to implement—requires two nested for loops, which is trivial. The resolution of this sampling and the number of repetitions can be easily configured so that the running time is not too long. The method can be easily generalised to problems in which each variable has a different default interval, or even a completely different domain. General version of the interval narrowing algorithm could take e.g. a percentage of the allowed domain for each variable category.

6.1.2 Greedy algorithms

Greedy algorithms have the potential to improve the initial solutions. An important drawback of such algorithms is the fact that they require that the objective function can evaluate incompletely initialised solutions. In our problems, this was not the case since both the reservations simulator and the variable trip pricing model did not support evaluating partial solutions. Still, in problems where this might be possible, greedy algorithms might be a very simple but efficient technique for choosing the initial point. Experimental validation is recommended, to validate the assumption that the greedy algorithm indeed does return better solutions than the random value generator (or the restricted-interval random value generator).

6.1.3 Investigating the objective function

In nearly all implementations, problem specific algorithms that have the potential to generate good solutions can be added. In the carsharing variable pricing example, increasing the prices for trips originating in areas with vehicle deficiency and in the direction of areas with a suficit could be increased by some predefined percentage. In the problem of carsharing reservations, experiments with constant QoS tables identified candidates with good objective function values. Such experiments by the developer can be done for any problem, even the black box problems.

The surrogate modelling approach assumes it is more productive to spend the developer time implementing models that will then learn the objective function landscape from the inputs gathered during the algorithm run. Conversely to this approach, in some cases it might be more useful to allow the developer to deduce some simple rules for initial solution generation based on what is known about the problem and what can be deduced by human intelligence.

6.2 Intensification instead of diversification, but not too much

Since there is not enough chance to spend much time intensifying the solution, successful algorithms for both LBE problems had much more emphasis on the intensification (local search) phase, than diversification (perturbation), which was left to be mild. Too much diversification can divert the search from good solutions even in problems where fast evaluation is possible. With slow evaluation, the reach of local search is very limited, therefore perturbation must also not be too intense, otherwise the ILS metaheuristic woud degrade to the random restart local search, which is a suboptimal configuration.

Similar results were reported in [174], where authors report best configurations of the ACO metaheuristic, applied without surrogates to a limited budget of evaluations variant of the TSP problem. The best configurations had parameter values that indicated more intensification than in the TSP version with the usual quick evaluation. When there is available information about good parameter values for the regular version of the problem, it is indeed reasonable to use the metaheuristic configuration with more intensification. This of course is possible only with problems with "artificially slowed down" evaluation, such as TSP which is in [174] limited to 100 or 1000 evaluations, despite the fact that much more are possible in short time. With "true slow evaluation" problems, it is generally not known what would be the best configuration in the "fast evaluation" case. Even then, it seems like a good idea to start with low diversification intensity and then tune the algorithm by gradually increasing the diverrsification component, especially if the initial solution generator is also tuned to produce good solutions.

6.3 Population or single-point methods?

Despite the fact that a large body of research in optimisation of slow objective functions uses variants of the genetic algorithm, the results from this work indicate that it might not be the best metaheuristic for the LBE problems. Genetic algorithm, and all population methods require a population of solutions, and evaluating all of them takes a long time. In fact, population methods are metaheuristics that use the highest number of evaluations per iteration of all metaheuristics and without the help of local search converge slowly.

Why is then genetic algorithm the most commonly used algorithm for solving LBE problems? Part of the answer is probably in the fact that it is the oldest metaheuristic technique with ample research available for the conventional problems with quick evaluation. Another reason might be that a large part of the LBE literature comes from publications in areas of application such as chemistry and mechanics, where authors might not be aware of all available optimisation techniques and simply decided to use the most widely cited or the oldest. To help speed up GA when applied to LBE problems, techniques that evaluate only a subset of the solutions during time have been developed. However, as with surrogate functions, this risks missing good solutions whose fitness was falsely estimated as low.

Single point methods such as ILS or Tabu Search require only one evaluation per iteration. There is no evidence that population based methods are faster, in fact, available literature indicates that the opposite is true for several problems described in 4.3.2. Additionally, single-point methods have simpler operators. This not only simplifies the development, it also allows the developer more direct control and higher predictability over where the search process will go. This can be important when a highly limited budget of evaluations is available. Due to these reasons and given the successful implementations of ILS for two LBE problems, the author argues that simple methods such as ILS might be a better way to solve practical problems even when they have a slow evaluation function.

6.4 Are surrogates needed?

The literature in general reports success with surrogate modelling based algorithms, with a few exceptions reporting negative results. It is unclear if this is due to high level of success of surrogates or due to the tendency of researchers in the optimisation community not to report negative results. Positive results generally include significant improvement in the solution quality given the same number of evaluations or reaching the best known solution quicker than without them. It should also be noted that for highly dimensional problems, the ability of surrogates to give good approximations with limited budget of evaluations decreases.

A notable drawback of the existing literature is that it usually does not report much detail about the effort invested into tuning and adaptation of the bare, “surrogate-free” techniques. As with metaheuristic comparisons, it is only fair to compare metaheuristics that are equally carefully tuned to the given problem. Further, a single study that uses a systematic comparison methodology reports that simply tuning the ACO algortihms performs better than the surrogate model based on EGO sampling [174].

Another issue with surrogates is that they add a new layer of complexity into already complex art of metaheuristic development. All the problems related to metaheuristics also apply to implementing surrogates. Which surrogate method is best suited for our problem? While there exist some guideliens, there is no formal process to help decide that. Integrating surrogates takes development time, and before testing them experimentally, it is difficult to predict how successful will the surrogates be. With the surrogate, there also comes the question of the sampling strategy and balancing the the surrogate fidelity vs. exploring known good regions. With all this comes also the problem of parameter tuning, which is complex for such systems with a lot of parameters and unclear connections between them. How does an increase of the tendency to improve surrogate fidelity and explore unseen regions correspond to the population size in GA? Should it be kept the same, increase or decrease? Detailed experiments are the only way to find such answers, however, with LBE problems, detailed experiments might not be practical given the their high resource requirements.

Finally, none of the surrogates provides any guarantees that they will help the algorithm work better. Given the fact that implementing and integrating the surrogate takes time, requires high levels of expertise and add even more complexity to the process, the author argues that at the very minimum, the performance of the direct application of the metaheuristic should be tested before starting using surrogates, especially if state-of-the-art performance is not a requirement. Starting off with a surrogate implementation without testing if we can have sufficient performance with a simpler method risks using precious development time to develop complex methods that might not bring the performance improvement at all, or th improvement might not be sufficient to justify the extra complexity and development time.

6.5 Bottom-up development of metaheuristics for the problems with limited budget of evaluations

Applying the bottom-up development methodology with gradual addition of complexity, and only if necessary, the process of solving problems with limited budget of evaluations would not introduce surrogates until reasonable effort to implement a direct metaheuristic is finished. Single-point methods are advisable instead of population methods. Then, and only if the performance is not good enough, it would be sensible to experiment with surrogates. If possible, it is advisable to run a preliminary test of various possible surrogates to identify the one performing the best on the given objective function. While this strategy cannot be proven to produce the best results, it is helpful since it has a tendency to produce simple and efficient solutions while saving the development time.

Chapter 7

Conclusion

This thesis is an effort to increase the applicability and manageability of metaheuristic techniques. While these methods have been praised as efficient and general problem-solvers, their adoption in practical problems, especially outside academia is still widely considered as complicated, expensive and requires a high level of expertise. A large number of available metaheuristics further confuses practitioners who want to choose the best method for their problem. To mitigate all these problems and reduce the complexity, the *bottom-up* development methodology for metaheuristic development is proposed. It is based in component-based view of metaheuristics, and the results that show that algorithms with more operators or parameters do not necessarily have better performance, and that sometimes the opposite holds. Following this proposed development methodology, the simplest component capable of producing a solution is implemented first. Then, more sophisticated components are gradually added until satisfactory performance is achieved.

The proposed methodology is tested on three problems that appear in practice. The first problem, called call centre workforce scheduling problem consists of scheduling staff in a call centre so that the number of available staff always meets the forecasted demand (volume of incoming calls), while also taking care of all the legal and organisational requirements. The problem was successfully solved with satisfactory solution quality, and to the author's surprise, this was achieved even with very simple metaheuristics: GRASP and iterated local search. The work is published as a conference paper in [395].

The second and third problem are the interdisciplinary part of the thesis. They are transportation problems related to carsharing. In the first problem, an efficient method for handling reservations in one-way carsharing systems is proposed and tested on a case study of the Lisbon municipality in Portugal. The method is further improved by introducing a geographically varying service quality, whose goal is to improve the overall service, as defined by four parameters: profit, proximity of cars to the user, longest allowed reservation time and accepted demand. An iterated local search metaheuristic was implemented to solve this problem. Using this method, the reservation times systems increased from 15-30 minutes available in current commercial providers up to 18 hours available in a simulated Lisbon provider, without a significant loss of profitability. The work is published in [181].

The third problem is the first application of variable pricing in one-way carsharing. The goal of the project was to create a procedure for determining trip prices that vary depending on the time of day and geographical zone of the service area that increase the provider profit the most. By setting the correct prices in each zone, the users can be stimulated to behave in the way that helps the carsharing provider to improve the vehicle stock balance and at the same time earn more money by having the prices adjusted to the current demand. The method was successful in improving the profit of a simulated carsharing provider in Lisbon under several possible variations. In the most likely case, the service using constant pricing was reporting losses of around 1800 € per day. The iterated local search for trip pricing was able to set the price values that turned these losses around and achieved profits of up to 2000€ per day. The work is published in [182].

While all three problems were useful tests and demonstrations that the development methodology proposed in this work allows rapid development of efficient problems solvers, the studied transportation problems had another element that increased their difficulty. Both problems, and especially the problem of carsharing variable pricing, are problems with a limited budget of evaluations. Unlike the most common approach in the literature—genetic algorithm and a surrogate method to estimate the values of the objective function, the solutions proposed in this work use iterated local search, without the use of surrogates. To the best of the author's knowledge, these are the first applications of the iterated local search metaheuristic to the problems with a limited budget of evaluations. These algorithms were implemented using a bottom-up development methodology, as a further indicator that the methodology can be extended to problems typically used complex surrogate modelling. The experience with successful solving of these two problems is further synthesised into a set of guidelines and recommendations for solving problems with limited budget of evaluations using metaheuristics.

Bibliography

[1] ““optimization.”, merriam-webster.com dictionary, merriam-webster”, available at: https://www.merriam-webster.com/dictionary/optimization , accessed Feb. 2020.

[2] Römer, L., Scheibel, T., “The elaborate structure of spider silk: structure and function of a natural high performance fiber”, Prion, Vol. 2, No. 4, 2008, str. 154–161.

[3] Termonia, Y., “Molecular modeling of spider silk elasticity”, Macromolecules, Vol. 27, No. 25, 1994, str. 7378–7381.

[4] Hearle, J. W., “Protein fibers: structural mechanics and future opportunities”, Journal of materials science, Vol. 42, No. 19, 2007, str. 8010–8019.

[5] Cummins, C., Seale, M., Macente, A., Certini, D., Mastropaolo, E., Viola, I. M., Nakayama, N., “A separated vortex ring underlies the flight of the dandelion”, Nature, Vol. 562, No. 7727, 2018, str. 414–418.

[6] Knorr, W. R., Textual studies in ancient and medieval geometry. Springer Science & Business Media, 2012.

[7] Alekseev, V. M., Optimal control. Springer Science & Business Media, 2013.

[8] Birattari, M., “The problem of tuning metaheuristics as seen from a machine learning perspective”, 2004.

[9] Rashed, R., Classical mathematics from al-Khwarizmi to Descartes. Routledge, 2014.

[10] Dunham, W., The mathematical universe: An alphabetical journey through the great proofs, problems and personalities. Wiley, 1994.

[11] Horsfall, N., Virgil, Aeneid 2: a commentary. Brill, 2008.

[12] Floudas, C. A., Pardalos, P. M., Encyclopedia of optimization. Springer Science & Business Media, 2008.

[13] Du, D.-Z., Pardalos, P. M., Wu, W., “History of optimization.”, 2009.

[14] Barnes, J. *et al.*, Complete works of Aristotle, volume 1: The revised Oxford translation. Princeton University Press, 1984, Vol. 192.

[15] Euclid, Heath, S. T. L., Euclid's elements. Dent, 1933.

[16] Heath, T. L. *et al.*, The thirteen books of Euclid's Elements. Courier Corporation, 1956.

[17] Smith, A. M. *et al.*, Ptolemy's theory of visual perception: an English translation of the Optics. American Philosophical Society, 1996, Vol. 82.

[18] Smith, A. M., "Ptolemy and the foundations of ancient mathematical optics: A source based guided study", Transactions of the American Philosophical Society, Vol. 89, No. 3, 1999, str. 1–172.

[19] Cohen, M. R., Drabkin, I. E., A source book in Greek science, 1948.

[20] Cuomo, S., Pappus of Alexandria and the mathematics of late antiquity. Cambridge University Press, 2000.

[21] Heath, T., A History of Greek Mathematics Vol 2 from Aristarchus to Diophantus. Clarendon Press, 1921.

[22] Cajori, F., A history of mathematics. American Mathematical Soc., 1999, Vol. 303.

[23] Garber, D., Ayers, M., The Cambridge history of seventeenth-century philosophy. Cambridge University Press, 2003, Vol. 2.

[24] Newton, I., "Philosophiæ naturalis principia mathematica (mathematical principles of natural philosophy)", London (1687), Vol. 1687, 1687.

[25] Newton, I., The Principia: mathematical principles of natural philosophy. Univ of California Press, 1999.

[26] Bernoulli, J., "Problema novum ad cujus solutionem mathematici invitantur", Acta Eruditorum, Vol. 15, 1696, str. 264–269.

[27] "De ratione temporis quo grave labitur per rectam data duo puncta conjungentem, ad tempus brevissimum quo, vi gravitatis, transit ab horum uno ad alterum per arcum cycloidis", Philosophical Transactions (1683-1775), Vol. 19, 1697, str. 424–425, available at: http://www.jstor.org/stable/102342

[28] Leibniz, G., "Communicatio suae pariter, duarumque alienarum ad adendum sibi primum a dn. jo. bernoullio, deinde a dn. marchione hospitalio communicatarum solutionum problematis curvae celerrimi descensus a dn. jo. bernoullio geometris publice propositi,

una cum solutione sua problematis alterius ab eodem postea propositi", Acta Eruditorum, Vol. 16, 1697, str. 201–205.

[29] Bernoulli, J., "Curvatura radii in diaphanus non uniformibus, solutioque problematis a se in "acta", 1696", propositi de invenienda "Linea Brachistochrona etc.", Acta Eruditorum Leipzig, May issue, 1996, str. 206.

[30] Cooke, R. L., The history of mathematics: A brief course. John Wiley & Sons, 2011.

[31] Du, D.-Z., Lu, B., Ngo, H. Q., Pardalos, P. M., "Steiner tree problems.", Encyclopedia of optimization, Vol. 5, 2009, str. 227–290.

[32] Hwang, F. K., Richards, D. S., Winter, P., Widmayer, P., "The steiner tree problem, annals of discrete mathematics, volume 53", ZOR-Methods and Models of Operations Research, Vol. 41, No. 3, 1995, str. 382.

[33] Gross, J. L., Yellen, J., Handbook of graph theory. CRC press, 2003.

[34] Euler, L., "Solutio problematis ad geometriam situs pertinentis", Commentarii academiae scientiarum Petropolitanae, 1741, str. 128–140.

[35] Schrijver, A., "On the history of the shortest path problem", Documenta Mathematica, Vol. 17, 2012, str. 155.

[36] Karp, R. M., "Reducibility among combinatorial problems", in Complexity of computer computations. Springer, 1972, str. 85–103.

[37] Graham, R. L., Hell, P., "On the history of the minimum spanning tree problem", Annals of the History of Computing, Vol. 7, No. 1, 1985, str. 43–57.

[38] Tarjan, R. E., "Algorithms for maximum network flow", in Netflow at Pisa. Springer, 1986, str. 1–11.

[39] Villani, C., Topics in optimal transportation. American Mathematical Soc., 2003, No. 58.

[40] Monge, G., "Mémoire sur la théorie des déblais et des remblais", Histoire de l'Académie Royale des Sciences de Paris, 1781.

[41] Kantorovich, L., "On the translocation of masses. c. r", Doklady) Acad. Sci. URSS (NS), Vol. 37, 1942, str. 199–201.

[42] Fourier, J., "Histoire de l'académie, partie mathématique", Mémoire de l'Académie des sciences de l'Institut de France, 1824.

[43] Fourier, J. B. J., “Solution d’une question particuliere du calcul des inégalités”, Nouveau Bulletin des Sciences par la Société philomatique de Paris, Vol. 99, 1826, str. 100.

[44] Dantzig, G. B., “Fourier-motzkin elimination and its dual”, STANFORD UNIV CA DEPT OF OPERATIONS RESEARCH, Tech. Rep., 1972.

[45] Dantzig, G. B., “Reminiscences about the origins of linear programming.”, STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB, Tech. Rep., 1981.

[46] Motzkin, T. S., Beitrage zur theorie der linearen Ungleichungen: Inaugural-Dissertation. Azriel, 1936.

[47] Miller, H. R., Optimization: foundations and applications. John Wiley & Sons, 2011.

[48] Cauchy, A., “Méthode générale pour la résolution des systemes d’équations simultanées”, Comp. Rend. Sci. Paris, Vol. 25, No. 1847, 1847, str. 536–538.

[49] Kauder, E., History of marginal utility theory. Princeton University Press, 2015.

[50] Faccarello, G., Kurz, H., Handbook on the history of economic analysis. Volume I. Great economists since Petty and Boisguilbert, 07 2016.

[51] Gossen, H. H., Entwickelung der Gesetze des menschlichen Verkehrs, und der daraus fließenden Regeln für menschliches Handeln. Friedrich Vieweg und Sohn, 1854.

[52] Georgescu-Roegen, N., “Hermann heinrich gossen: His life and work in historical perspective”, The laws of human relations and the rules of human action derived therefrom, 1983.

[53] Cournot, A. A., Recherches sur les principes mathématiques de la théorie des richesses, 1838.

[54] Sandmo, A., Economics evolving: A history of economic thought. Princeton University Press, 2011.

[55] Walras, L., Éléments d’économie politique pure, ou, Théorie de la richesse sociale. F. Rouge, 1896.

[56] De Fermat, P., Oeuvres de Fermat. Gauthier-Villars et fils, 1891, Vol. 1.

[57] Perlick, V., Ray optics, Fermat’s principle, and applications to general relativity. Springer Science & Business Media, 2000, Vol. 61.

[58] Leibniz, G., Huggard, E., Farrer, A., Theodicy. Lightning Source, 2010.

[59] Maupertuis, Accord de différentes loix de la nature qui avoient jusqu'ici paru incompatibles. Institut de France, 1748.

[60] Gerhardt, C. I., Ueber die vier Briefe von Leibniz, die Samuel König in dem Appel au public, Leide MDCCLIII, veröffentlich hat, 1898.

[61] Euler, L., Methodus inveniendi lineas curvas maximi minimive proprietate gaudentes. apud Marcum-Michaelem Bousquet, 1744.

[62] Hancock, H., Theory of maxima and minima. Ginn, 1917.

[63] MENGER, K., "Das botenproblem. k. menger (ed.)(1932)", Ergebnisse eines Mathematischen Kollo, 1930.

[64] Lawler, E. L., "The traveling salesman problem: a guided tour of combinatorial optimization", Wiley-Interscience Series in Discrete Mathematics, 1985.

[65] Schrijver, A., "On the history of combinatorial optimization (till 1960)", Handbooks in operations research and management science, Vol. 12, 2005, str. 1–68.

[66] Dantzig, G. B., Ramser, J. H., "The truck dispatching problem", Management science, Vol. 6, No. 1, 1959, str. 80–91.

[67] Toth, P., Vigo, D., The vehicle routing problem. SIAM, 2002.

[68] Dijkstra, E. W. *et al.*, "A note on two problems in connexion with graphs", Numerische mathematik, Vol. 1, No. 1, 1959, str. 269–271.

[69] Papadimitriou, C. H., Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity. Dover Publications, 1998.

[70] Pettie, S., Ramachandran, V., "An optimal minimum spanning tree algorithm", Journal of the ACM (JACM), Vol. 49, No. 1, 2002, str. 16–34.

[71] Boruvka, O., "O jistem problemu minimaalnim. moravske prirodovedecke spolecnosti 3, 37-58", 1926.

[72] Kruskal, J. B., "On the shortest spanning subtree of a graph and the traveling salesman problem", Proceedings of the American Mathematical society, Vol. 7, No. 1, 1956, str. 48–50.

[73] Gass, S. I., "George b. dantzig", in Profiles in Operations Research. Springer, 2011, str. 217–240.

[74] Pierre, D. A., Optimization theory with applications. Courier Corporation, 1986.

[75] Dantzig, G. B., “Maximization of a linear function of variables subject to linear inequalities”, Activity analysis of production and allocation, Vol. 13, 1951, str. 339–347.

[76] Dantzig, G. B., “Linear programming”, History of mathematical programming, 1991, str. 19–31.

[77] VALLEE POUSSIN, C., “Sur la methode de l’approximation minimum”, Annales Soc. Scient. Bruxelles, Vol. 35, 1911, str. 1–16.

[78] Kantorovich, L. V., “The mathematical method of production planning and organization”, Management Science, Vol. 6, No. 4, 1939, str. 363–422.

[79] Hitchcock, F. L., “The distribution of a product from several sources to numerous localities”, Journal of mathematics and physics, Vol. 20, No. 1-4, 1941, str. 224–230.

[80] Dantzig, G. B., “Origins of the simplex method”, in A history of scientific computing, 1990, str. 141–151.

[81] Dantzig, G., “Linear programming and extensions, princeton, univ”, Press, Princeton, NJ, 1963.

[82] Kantorovich, L. V., “My journey in science (proposed report to the moscow mathematical society)”, Russian Mathematical Surveys, Vol. 42, No. 2, 1987, str. 233.

[83] Leifman, L. J., Functional analysis, optimization, and mathematical economics. Oxford univ. press New York, 1990.

[84] Gass, S. I., Rosenhead, J., “Leonid vital’evich kantorovich”, in Profiles in Operations Research. Springer, 2011, str. 157–170.

[85] Rosenhead, J., “Ifors’operational research hall of fame: Leonid vitaliyevich kantorovich”, International Transactions in Operational Research, Vol. 10, No. 6, 2003, str. 665–667.

[86] Kantorovich, L. V. *et al.*, “The best use of economic resources.”, The best use of economic resources., 1965.

[87] Nocedal, J., Wright, S., Numerical optimization. Springer Science & Business Media, 2006.

[88] Karush, W., “Minima of functions of several variables with inequalities as side constraints”, M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago, 1939.

[89] Kuhn, H. W., Tucker, A. W., “Nonlinear programming, in (j. neyman, ed.) proceedings of the second berkeley symposium on mathematical statistics and probability”, 1951.

[90] Ford, L. R., Fulkerson, D. R., “Maximal flow through a network”, Canadian Journal of Mathematics, Vol. 8, 1956, str. 399–404.

[91] Gomory, R., “Outline of an algorithm for integer solutions to linear programs”, Bulletin of the American Mathematical Society, Vol. 64, 09 1958, str. 275-278.

[92] Dantzig, G. B., “Linear programming under uncertainty”, in Stochastic programming. Springer, 2010, str. 1–11.

[93] Dixon, L., Szegö, G., Towards global optimisation: proceedings of a workshop at the University of Cagliari, Italy, October 1974, ser. North Holland mathematical library. North-Holland Pub. Co., 1975, No. v. 1.

[94] Dixon, L., Szegö, G., Towards global optimisation 2, ser. Towards global optimisation / eds L. C. W. Dixon and G. P. Szegö. North-Holland Pub. Co., 1978.

[95] Russell, S., Norvig, P., Artificial Intelligence: A Modern Approach, ser. Always learning. Pearson, 2016.

[96] Sörensen, K., Sevaux, M., Glover, F., “A history of metaheuristics”, Handbook of heuristics, 2018, str. 1–18.

[97] Simon, H. A., Newell, A., “Heuristic problem solving: The next advance in operations research”, Operations research, Vol. 6, No. 1, 1958, str. 1–10.

[98] Aarts, E., Aarts, E. H., Lenstra, J. K., Local search in combinatorial optimization. Princeton University Press, 2003.

[99] Savage, L. J., “The theory of statistical decision”, Journal of the American Statistical association, Vol. 46, No. 253, 1951, str. 55–67.

[100] Michiels, W., Aarts, E., Korst, J., Theoretical aspects of local search. Springer Science & Business Media, 2007.

[101] Flood, M. M., “The traveling-salesman problem”, Operations research, Vol. 4, No. 1, 1956, str. 61–75.

[102] Croes, G. A., “A method for solving traveling-salesman problems”, Operations research, Vol. 6, No. 6, 1958, str. 791–812.

[103] Bock, F., “An algorithm for solving travelling-salesman and related network optimization problems”, in Operations Research, Vol. 6, No. 6. INST OPERATIONS RESEARCH MANAGEMENT SCIENCES 901 ELKRIDGE LANDING RD, STE . . . , 1958, str. 897–897.

[104] Mart, R., Pardalos, P. M., Resende, M. G. C., Handbook of Heuristics, 1st ed. Springer Publishing Company, Incorporated, 2018.

[105] Gendreau, M., Potvin, J.-Y. *et al.*, Handbook of metaheuristics, second edition ed. Springer, 2010.

[106] TURING, I. B. A., “Computing machinery and intelligence-am turing”, Mind, Vol. 59, No. 236, 1950, str. 433.

[107] Barricelli, N. A. *et al.*, “Esempi numerici di processi di evoluzione”, Methodos, Vol. 6, No. 21-22, 1954, str. 45–68.

[108] Barricelli, N. A., Symbiogenetic evolution processes realized by artificial methods, 1957.

[109] Box, G. E., “Evolutionary operation: A method for increasing industrial productivity”, Journal of the Royal Statistical Society: Series C (Applied Statistics), Vol. 6, No. 2, 1957, str. 81–101.

[110] Friedman, G. J., “Digital simulation of an evlutionary process”, General Systems Yearbook, 1959, str. 171–184.

[111] Rechenberg, I., “Cybernetic solution path of an experimental problem”, Royal Aircraft Establishment Library Translation 1122, 1965.

[112] Rechenberg, I., “Evolutionsstrategie—optimierung technischer systeme nach prinzipien der biologischen information”, Stuttgart-Bad Cannstatt: Friedrich Frommann Verlag, 1973.

[113] Rechenberg, I., “Evolutionsstrategien”, in Simulationsmethoden in der Medizin und Biologie. Springer, 1978, str. 83–114.

[114] Fogel, D., Evolutionary Computation: The Fossil Record. Wiley, 1998.

[115] Fogel, L. J., Owens, A. J., Walsh, M. J., “Artificial intelligence through simulated evolution”, 1966.

[116] Holland, J., “Adaptation in natural and artificial systems: an introductory analysis with application to biology”, Control and artificial intelligence, 1975.

[117] Goldberg, D., Genetic Algorithms in Search, Optimization, and Machine Learning, ser. Artificial Intelligence. Addison-Wesley Publishing Company, 1989.

[118] Lighthill, J., “Artificial intelligence: A general survey”, Artificial Intelligence: a paper symposium, 1973.

[119] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., “Optimization by simulated annealing”, science, Vol. 220, No. 4598, 1983, str. 671–680.

[120] Pincus, M., “Letter to the editor—a monte carlo method for the approximate solution of certain types of constrained optimization problems”, Operations Research, Vol. 18, No. 6, 1970, str. 1225–1228.

[121] van Laarhoven, P., Aarts, E., Simulated Annealing: Theory and Applications, ser. Mathematics and Its Applications. Springer Netherlands, 1987.

[122] Nikolaev, A. G., Jacobson, S. H., “Simulated annealing”, in Handbook of metaheuristics. Springer, 2010, str. 1–39.

[123] Dorigo, M., Colorni, A., Maniezzo, V., “Distributed optimization by ant colonies”, 1991.

[124] Dorigo, M., Stützle, T., Ant Colony Optimization. Scituate, MA, USA: Bradford Company, 2004.

[125] Kennedy, J., Eberhart, R., “Particle swarm optimization”, in Proceedings of ICNN’95 - International Conference on Neural Networks, Vol. 4, 1995, str. 1942-1948 vol.4.

[126] Clerc, M., Particle swarm optimization. John Wiley & Sons, 2010, Vol. 93.

[127] Lourenço, H. R., Martin, O. C., Stützle, T., “Iterated local search: Framework and applications”, in Handbook of metaheuristics. Springer, 2019, str. 129–168.

[128] Baxter, J., “Local optima avoidance in depot location”, Journal of the Operational Research Society, Vol. 32, No. 9, 1981, str. 815–819.

[129] Glover, F., “Future paths for integer programming and links to artificial intelligence”, Computers & operations research, Vol. 13, No. 5, 1986, str. 533–549.

[130] Glover, F., “Tabu search—part i”, ORSA Journal on computing, Vol. 1, No. 3, 1989, str. 190–206.

[131] Glover, F., “Tabu search—part ii”, ORSA Journal on computing, Vol. 2, No. 1, 1990, str. 4–32.

[132] Gendreau, M., Potvin, J.-Y., Tabu Search. Boston, MA: Springer US, 2010, str. 41–59.

[133] Feo, T. A., Resende, M. G., “A probabilistic heuristic for a computationally difficult set covering problem”, Operations Research Letters, Vol. 8, No. 2, 1989, str. 67 - 71.

[134] Festa, P., Resende, M. G., “Grasp: An annotated bibliography”, in Essays and surveys in metaheuristics. Springer, 2002, str. 325–367.

[135] Resende, M. G., Ribeiro, C. C., “Greedy randomized adaptive search procedures: Advances and extensions”, in Handbook of metaheuristics. Springer, 2019, str. 169–220.

[136] Mladenović, N., Hansen, P., “Variable neighborhood search”, Computers & operations research, Vol. 24, No. 11, 1997, str. 1097–1100.

[137] Hansen, P., Mladenović, N., Brimberg, J., Pérez, J. A. M., “Variable neighborhood search”, in Handbook of metaheuristics. Springer, 2019, str. 57–97.

[138] Gödel, K., “Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i”, Monatshefte für mathematik und physik, Vol. 38, No. 1, 1931, str. 173–198.

[139] Gödel, K., On formally undecidable propositions of Principia Mathematica and related systems. Courier Corporation, 1992.

[140] Church, A., “An unsolvable problem of elementary number theory”, American journal of mathematics, Vol. 58, No. 2, 1936, str. 345–363.

[141] Turing, A. M., “On computable numbers, with an application to the entscheidungsproblem”, J. of Math, Vol. 58, No. 345-363, 1936, str. 5.

[142] Knuth, D., The Art of Computer Programming: Volume 1: Fundamental Algorithms. Pearson Education, 1997.

[143] Cook, S. A., “The complexity of theorem-proving procedures”, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, ser. STOC ’71. New York, NY, USA: Association for Computing Machinery, 1971, str. 151–158, available at: https://doi.org/10.1145/800157.805047

[144] Wolpert, D. H., Macready, W. G. *et al.*, “No free lunch theorems for search”, Technical Report SFI-TR-95-02-010, Santa Fe Institute, Tech. Rep., 1995.

[145] Wolpert, D. H., Macready, W. G. *et al.*, “No free lunch theorems for optimization”, IEEE transactions on evolutionary computation, Vol. 1, No. 1, 1997, str. 67–82.

[146] “The sveriges riksbank prize in economic sciences in memory of alfred nobel 1975”, available at: https://www.nobelprize.org/prizes/economic-sciences/1975/press-release/

[147] "The sveriges riksbank prize in economic sciences in memory of alfred nobel 1990", available at: https://www.nobelprize.org/prizes/economic-sciences/1990/press-release/

[148] Chahuat, B., "Formulating an optimization problem", available at: http://macc.mcmaster.ca/maccfiles/chachuatnotes/01-Formulation_handout.pdf 2011.

[149] Michalewicz, Z., Fogel, D., How to Solve It: Modern Heuristics. Springer Berlin Heidelberg, 2013.

[150] Gendreau, M., Potvin, J.-Y. *et al.*, Handbook of metaheuristics, third edition ed. Springer, 2019.

[151] Cavazzuti, M., Deterministic Optimization. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, str. 77–102.

[152] The Editors of Encyclopaedia Britannica, "Determinism", available at: https://www.britannica.com/topic/determinism 2020.

[153] Cormen, T., Leiserson, C., Rivest, R., Stein, C., Introduction to Algorithms, ser. Computer science. MIT Press, 2009.

[154] Motwani, R., Raghavan, P., Randomized Algorithms, ser. Cambridge International Series on Parallel Computation. Cambridge University Press, 1995.

[155] Rosen, K., Handbook of Discrete and Combinatorial Mathematics, ser. Discrete Mathematics and Its Applications. Taylor & Francis, 1999.

[156] Žubrinić, D., Diskretna matematika, Elezović, N., (ur.). Element, 2001.

[157] Shikin, E. V., Handbook and atlas of curves. CRC Press, 2014.

[158] Weisstein, E. W., "Smooth function", available at: https://mathworld.wolfram.com/SmoothFunction.html

[159] Lang, S., Undergraduate algebra. Springer Science & Business Media, 2005.

[160] Bartle, R. G., Sherbert, D. R., Introduction to real analysis. Wiley New York, 2000, Vol. 2.

[161] Weisstein, E. W., "Continuous function", available at: https://mathworld.wolfram.com/ContinuousFunction.html 2020.

[162] Blum, C., Roli, A., "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", ACM computing surveys (CSUR), Vol. 35, No. 3, 2003, str. 268–308.

[163] Vazirani, V. V., Approximation algorithms. Springer Science & Business Media, 2013.

[164] Miettinen, K., Nonlinear multiobjective optimization. Springer Science & Business Media, 2012, Vol. 12.

[165] Branke, J., Branke, J., Deb, K., Miettinen, K., Słowiński, R., Multiobjective optimization: Interactive and evolutionary approaches. Springer Science & Business Media, 2008, Vol. 5252.

[166] Dantzig, G. B., Linear programming and extensions. Princeton university press, 1998, Vol. 48.

[167] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., Vance, P. H., "Branch-and-price: Column generation for solving huge integer programs", Operations research, Vol. 46, No. 3, 1998, str. 316–329.

[168] Bixby, R. E., Gregory, J. W., Lustig, I. J., Marsten, R. E., Shanno, D. F., "Very large-scale linear programming: A case study in combining interior point and simplex methods", Operations Research, Vol. 40, No. 5, 1992, str. 885–897.

[169] Pecin, D., Contardo, C., Desaulniers, G., Uchoa, E., "New enhancements for the exact solution of the vehicle routing problem with time windows", INFORMS Journal on Computing, Vol. 29, No. 3, 2017, str. 489–502.

[170] Tsurkov, V., Large-scale Optimization: Problems and Methods. Springer Science & Business Media, 2013, Vol. 51.

[171] Jin, Y., "Surrogate-assisted evolutionary computation: Recent advances and future challenges", Swarm and Evolutionary Computation, Vol. 1, No. 2, 2011, str. 61–70.

[172] Jin, Y., "A comprehensive survey of fitness approximation in evolutionary computation", Soft computing, Vol. 9, No. 1, 2005, str. 3–12.

[173] Shi, L., Rasheed, K., "A survey of fitness approximation methods applied in evolutionary algorithms", in Computational intelligence in expensive optimization problems. Springer, 2010, str. 3–28.

[174] Caceres, L. P., López-Ibáñez, M., Stützle, T., "Ant colony optimization on a limited budget of evaluations", Swarm Intelligence, Vol. 9, No. 2-3, 2015, str. 103–124.

[175] Tenne, Y., Goh, C.-K., Computational intelligence in expensive optimization problems. Springer Science & Business Media, 2010, Vol. 2.

[176] Teixeira, C., Covas, J., Stützle, T., Gaspar-Cunha, A., “Multi-objective ant colony optimization for the twin-screw configuration problem”, Engineering Optimization, Vol. 44, No. 3, 2012, str. 351–371.

[177] Fu, M. C. *et al.*, Handbook of simulation optimization. Springer, 2015, Vol. 216.

[178] Amaran, S., Sahinidis, N. V., Sharda, B., Bury, S. J., “Simulation optimization: a review of algorithms and applications”, Annals of Operations Research, Vol. 240, No. 1, 2016, str. 351–380.

[179] Andradóttir, S., “Simulation optimization”, Handbook of simulation: Principles, methodology, advances, applications, and practice, 1998, str. 307–333.

[180] April, J., Glover, F., Kelly, J. P., Laguna, M., “Practical introduction to simulation optimization”, in Proceedings of the 35th conference on Winter simulation: driving innovation. Citeseer, 2003, str. 71–78.

[181] Jorge, D., Molnar, G., de Almeida Correia, G. H., “Trip pricing of one-way station-based carsharing networks with zone and time of day price variations”, Transportation Research Part B: Methodological, Vol. 81, 2015, str. 461–482.

[182] Molnar, G., de Almeida Correia, G. H., “Long-term vehicle reservations in one-way free-floating carsharing systems: A variable quality of service model”, Transportation Research Part C: Emerging Technologies, Vol. 98, 2019, str. 298–322.

[183] Jin, Y., Olhofer, M., Sendhoff, B., “Managing approximate models in evolutionary aerodynamic design optimization”, in Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546), Vol. 1. IEEE, 2001, str. 592–599.

[184] Schneider, G., “Neural networks are useful tools for drug design”, Neural Networks, Vol. 13, No. 1, 2000, str. 15–16.

[185] Piccolboni, A., Mauri, G., “Application of evolutionary algorithms to protein folding prediction”, in European Conference on Artificial Evolution. Springer, 1997, str. 123–135.

[186] Fernandez, S., Alvarez, S., Díaz, D., Iglesias, M., Ena, B., “Scheduling a galvanizing line by ant colony optimization”, in International Conference on Swarm Intelligence. Springer, 2014, str. 146–157.

[187] PIERRET, S., VAN DEN BRAEMBUSSCHE, R., “Turbomachinery blade design using a navier-stokes solver and artificial neural network”, Journal of turbomachinery, Vol. 121, No. 2, 1999, str. 326–332.

[188] Thévenin, D., Janiga, G., Optimization and computational fluid dynamics. Springer Science & Business Media, 2008.

[189] Giannakoglou, K., "Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence", Progress in Aerospace Sciences, Vol. 38, No. 1, 2002, str. 43–76.

[190] Dumas, L., "Cfd-based optimization for automotive aerodynamics", in Optimization and computational fluid dynamics. Springer, 2008, str. 191–215.

[191] Sims, K., "Artificial evolution for computer graphics", in Proceedings of the 18th annual conference on Computer graphics and interactive techniques, 1991, str. 319–328.

[192] Takagi, H., "Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation", Proceedings of the IEEE, Vol. 89, No. 9, 2001, str. 1275–1296.

[193] Romero, J. J., The art of artificial evolution: A handbook on evolutionary art and music. Springer Science & Business Media, 2008.

[194] Daoud, S., Chehade, H., Yalaoui, F., Amodeo, L., "Efficient metaheuristics for pick and place robotic systems optimization", Journal of Intelligent Manufacturing, Vol. 25, No. 1, 2014, str. 27–41.

[195] Knuth, D. E., Selected papers on analysis of algorithms. CSLI Publications, 2000.

[196] Knuth, D. E., "The art of computer programming, vol 1: Fundamental", Algorithms. Reading, MA: Addison-Wesley, 1968.

[197] Papadimitriou, C., Computational Complexity. Addison-Wesley, 1994.

[198] Aho, A., Hopcroft, J., The Design and Analysis of Computer Algorithms, ser. Always learning. Pearson Education, 1974.

[199] Garey, M. R., Johnson, D. S., "Computers and intractability. 1979", F reeman, San Francisco, 1979.

[200] Yu, X., Gen, M., Introduction to Evolutionary Algorithms, ser. Decision Engineering. Springer London, 2010, available at: https://books.google.hr/books?id=rHQf_2Dx2ucC

[201] Sedgewick, R., Flajolet, P., An Introduction to the Analysis of Algorithms: Introdu Analysi Algori_p2. Pearson Education, 2013.

[202] Bachmann, P., Die analytische zahlentheorie. Teubner, 1894, Vol. 2.

[203] Landau, E., Handbuch der Lehre von der Verteilung der Primzahlen. Ripol Classic, 2000, Vol. 1.

[204] Goldreich, O., “Computational complexity: a conceptual perspective”, ACM Sigact News, Vol. 39, No. 3, 2008, str. 35–39.

[205] Cobham, A., “The intrinsic computational difficulty of functions”, 1965.

[206] Wegener, I., Complexity theory: exploring the limits of efficient algorithms. Springer Science & Business Media, 2005.

[207] Bellare, M., Rogaway, P., “The complexity of approximating a nonlinear program”, in Complexity in numerical optimization. World Scientific, 1993, str. 16–32.

[208] Rotman, B., “Will the digital computer transform classical mathematics?”, Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, Vol. 361, No. 1809, 2003, str. 1675–1690.

[209] Miller, R., Boxer, L., Algorithms sequential & parallel: A unified approach. Cengage Learning, 2012.

[210] “Definition of computation by merriam-webster”, available at: https://www.merriam-webster.com/dictionary/computation , accessed Feb. 2020.

[211] Hopcroft, J. E., Motwani, R., Ullman, J. D., “Introduction to automata theory, languages, and computation”, Acm Sigact News, Vol. 32, No. 1, 2001, str. 60–65.

[212] Sipser, M., Introduction to the Theory of Computation. Thomson Course Technology, 2006.

[213] Kleinberg, J., Tardos, E., Algorithm design. Pearson Education India, 2006.

[214] Levin, L., “Universal’nye perebornye zadachi”, Problemy Peredachi Informatsii, Vol. 9, No. 3, 1973, str. 265–266.

[215] Levin, L. A., “Universal sequential search problems”, Problemy peredachi informatsii, Vol. 9, No. 3, 1973, str. 115–116.

[216] Knuth, D. E., “A terminological proposal”, ACM SIGACT News, Vol. 6, No. 1, 1974, str. 12–18.

[217] Knuth, D. E., Larrabee, T., Roberts, P. M., Roberts, P. M., Mathematical writing. Mathematical Association of America Washington, DC, 1989, Vol. 14.

[218] Schumacher, C., Vose, M. D., Whitley, L. D., "The no free lunch and problem description length", in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), 2001, str. 565–570.

[219] Koehler, G. J., "Conditions that obviate the no-free-lunch theorems for optimization", INFORMS Journal on Computing, Vol. 19, No. 2, 2007, str. 273–279.

[220] Ho, Y.-C., Pepyne, D. L., "Simple explanation of the no-free-lunch theorem and its implications", Journal of optimization theory and applications, Vol. 115, No. 3, 2002, str. 549–570.

[221] Geer, D., "Chip makers turn to multicore processors", Computer, Vol. 38, No. 5, 2005, str. 11–13.

[222] Sanders, P., Algorithm Engineering. Boston, MA: Springer US, 2011, str. 33–38, available at: https://doi.org/10.1007/978-0-387-09766-4_89

[223] Strongin, R. G., Sergeyev, Y. D., Global optimization with non-convex constraints: Sequential and parallel algorithms. Springer Science & Business Media, 2013, Vol. 45.

[224] Censor, Y., Zenios, S. A. *et al.*, Parallel optimization: Theory, algorithms, and applications. Oxford University Press on Demand, 1997.

[225] Bennett, C. H., Bernstein, E., Brassard, G., Vazirani, U., "Strengths and weaknesses of quantum computing", SIAM journal on Computing, Vol. 26, No. 5, 1997, str. 1510–1523.

[226] Grover, L. K., "A fast quantum mechanical algorithm for database search", in Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, str. 212–219.

[227] Aaronson, S., "Quantum computing and hidden variables", Physical Review A, Vol. 71, No. 3, 2005, str. 032325.

[228] "Ibm quantum computing", available at: https://www.ibm.com/quantum-computing/ , accessed February 2020.

[229] of Sciences, N. A., Quantum computing: progress and prospects. National Academies Press, 2019.

[230] Martí, R., Panos, P., Resende, M., Handbook of Heuristics, ser. Handbook of Heuristics. Springer International Publishing, 2017.

[231] Zabinsky, Z. B., Stochastic adaptive search for global optimization. Springer Science & Business Media, 2013, Vol. 72.

[232] Trevisan, L., “Inapproximability of combinatorial optimization problems”, arXiv preprint cs/0409043, 2004.

[233] Pearl, J., “Intelligent search strategies for computer problem solving”, Addision Wesley, 1984.

[234] Sörensen, K., Glover, F. W., “Metaheuristics”, Encyclopedia of operations research and management science, 2013, str. 960–970.

[235] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S., “Hyper-heuristics: An emerging direction in modern search technology”, in Handbook of metaheuristics. Springer, 2003, str. 457–474.

[236] Lenstra, J. K., Kan, A. R., “Complexity of vehicle routing and scheduling problems”, Networks, Vol. 11, No. 2, 1981, str. 221–227.

[237] Cooper, T. B., Kingston, J. H., “The complexity of timetable construction problems”, in International conference on the practice and theory of automated timetabling. Springer, 1995, str. 281–295.

[238] Even, S., Itai, A., Shamir, A., “On the complexity of time table and multi-commodity flow problems”, in 16th Annual Symposium on Foundations of Computer Science (sfcs 1975). IEEE, 1975, str. 184–193.

[239] Van Harmelen, F., Lifschitz, V., Porter, B., Handbook of knowledge representation. Elsevier, 2008.

[240] Schaefer, T. J., “The complexity of satisfiability problems”, in Proceedings of the tenth annual ACM symposium on Theory of computing, 1978, str. 216–226.

[241] Williams, R., Gomes, C. P., Selman, B., “Backdoors to typical case complexity”, in IJCAI, Vol. 3, 2003, str. 1173–1178.

[242] Korte, B., Vygen, J., Korte, B., Vygen, J., Combinatorial optimization. Springer, 2012, Vol. 2.

[243] Wolsey, L. A., Nemhauser, G. L., Integer and combinatorial optimization. John Wiley & Sons, 1999, Vol. 55.

[244] Wolsey, L. A., Integer programming. John Wiley & Sons, 1998, Vol. 52.

[245] Schrijver, A., Theory of linear and integer programming. John Wiley & Sons, 1998.

[246] Papadimitriou, C. H., “On the complexity of integer programming”, Journal of the ACM (JACM), Vol. 28, No. 4, 1981, str. 765–768.

[247] Lenstra Jr, H. W., “Integer programming with a fixed number of variables”, Mathematics of operations research, Vol. 8, No. 4, 1983, str. 538–548.

[248] Dakin, R. J., “A tree-search algorithm for mixed integer programming problems”, The computer journal, Vol. 8, No. 3, 1965, str. 250–255.

[249] Kellerer, H., Pferschy, U., Pisinger, D., Knapsack problems. Springer, 2004.

[250] Martello, S., Pisinger, D., Toth, P., “New trends in exact algorithms for the 0–1 knapsack problem”, European Journal of Operational Research, Vol. 123, No. 2, 2000, str. 325–332.

[251] Kolesar, P. J., “A branch and bound algorithm for the knapsack problem”, Management science, Vol. 13, No. 9, 1967, str. 723–735.

[252] Horowitz, E., Sahni, S., “Computing partitions with applications to the knapsack problem”, Journal of the ACM (JACM), Vol. 21, No. 2, 1974, str. 277–292.

[253] Lawler, E. L., “Fast approximation algorithms for knapsack problems”, Mathematics of Operations Research, Vol. 4, No. 4, 1979, str. 339–356.

[254] Hanafi, S., Freville, A., “An efficient tabu search approach for the 0–1 multidimensional knapsack problem”, European Journal of Operational Research, Vol. 106, No. 2-3, 1998, str. 659–675.

[255] Hanafi, S., Freville, A., El Abdellaoui, A., “Comparison of heuristics for the 0–1 multidimensional knapsack problem”, in Meta-Heuristics. Springer, 1996, str. 449–465.

[256] Chu, P. C., Beasley, J. E., “A genetic algorithm for the multidimensional knapsack problem”, Journal of heuristics, Vol. 4, No. 1, 1998, str. 63–86.

[257] Glover, F., Kochenberger, G. A., “Critical event tabu search for multidimensional knapsack problems”, in Meta-heuristics. Springer, 1996, str. 407–427.

[258] Kang, J., Park, S., “Algorithms for the variable sized bin packing problem”, European Journal of Operational Research, Vol. 147, No. 2, 2003, str. 365–372.

[259] Johnson, D. S., “Fast algorithms for bin packing”, Journal of Computer and System Sciences, Vol. 8, No. 3, 1974, str. 272–314.

[260] De La Vega, W. F., Lueker, G. S., “Bin packing can be solved within 1+ ε in linear time”, Combinatorica, Vol. 1, No. 4, 1981, str. 349–355.

[261] Coffman Jr, E., Garey, M., Johnson, D., “Approximation algorithms for bin packing: A survey”, Approximation algorithms for NP-hard problems, 1996, str. 46–93.

[262] Yue, M., “A simple proof of the inequality FFD (L) $\leq$ 11/9 OPT (L) + 1, $\forall$ L for the FFD bin-packing algorithm”, Acta mathematicae applicatae sinica, Vol. 7, No. 4, 1991, str. 321–331.

[263] Bansal, N., Correa, J. R., Kenyon, C., Sviridenko, M., “Bin packing in multiple dimensions: inapproximability results and approximation schemes”, Mathematics of operations research, Vol. 31, No. 1, 2006, str. 31–49.

[264] Chekuri, C., Khanna, S., “On multidimensional packing problems”, SIAM journal on computing, Vol. 33, No. 4, 2004, str. 837–851.

[265] Woeginger, G. J., “There is no asymptotic PTAS for two-dimensional vector packing”, Information Processing Letters, Vol. 64, No. 6, 1997, str. 293–297.

[266] Held, M., Karp, R. M., “A dynamic programming approach to sequencing problems”, Journal of the Society for Industrial and Applied mathematics, Vol. 10, No. 1, 1962, str. 196–210.

[267] Bellmore, M., Nemhauser, G. L., “The traveling salesman problem: a survey”, Operations Research, Vol. 16, No. 3, 1968, str. 538–558.

[268] Fischetti, M., Lodi, A., Toth, P., “Solving real-world atsp instances by branch-and-cut”, in Combinatorial Optimization—Eureka, You Shrink! Springer, 2003, str. 64–77.

[269] Woeginger, G. J., “Exact algorithms for np-hard problems: A survey”, in Combinatorial optimization—eureka, you shrink! Springer, 2003, str. 185–207.

[270] Karpinski, M., Lampis, M., Schmied, R., “New inapproximability bounds for TSP”, Journal of Computer and System Sciences, Vol. 81, No. 8, 2015, str. 1665–1677.

[271] Lampis, M., “Improved inapproximability for TSP”, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. Springer, 2012, str. 243–253.

[272] Sahni, S., Gonzalez, T., “P-complete approximation problems”, Journal of the ACM (JACM), Vol. 23, No. 3, 1976, str. 555–565.

[273] Papadimitriou, C. H., Vempala, S., “On the approximability of the traveling salesman problem”, Combinatorica, Vol. 26, No. 1, 2006, str. 101–120.

[274] Toth, P., Vigo, D., Vehicle routing: problems, methods, and applications. SIAM, 2014.

[275] Augerat, P., “Approche polyèdrale du problème de tournées de véhicules”, Doktorski rad, Institut National Polytechnique de Grenoble-INPG, 1995.

[276] Savelsbergh, M. W., “Local search in routing problems with time windows”, Annals of Operations research, Vol. 4, No. 1, 1985, str. 285–305.

[277] Dror, M., “Note on the complexity of the shortest path models for column generation in VRPTW”, Operations Research, Vol. 42, No. 5, 1994, str. 977–978.

[278] Archetti, C., Mansini, R., Speranza, M. G., “Complexity and reducibility of the skip delivery problem”, Transportation Science, Vol. 39, No. 2, 2005, str. 182–187.

[279] Kallehauge, B., Larsen, J., Madsen, O. B., “Lagrangian duality applied to the vehicle routing problem with time windows”, Computers & Operations Research, Vol. 33, No. 5, 2006, str. 1464–1487.

[280] Cook, W., Rich, J. L., “A parallel cutting-plane algorithm for the vehicle routing problem with time windows”, Tech. Rep., 1999.

[281] Larsen, J., Parallelization of the vehicle routing problem with time windows. Institute of Mathematical Modelling, Technical University of Denmark Lyngby, 1999.

[282] Baldacci, R., Mingozzi, A., Roberti, R., “Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints”, European Journal of Operational Research, Vol. 218, No. 1, 2012, str. 1–6.

[283] Sörensen, K., Sevaux, M., Schittekat, P., ““multiple neighbourhood” search in commercial VRP packages: Evolving towards self-adaptive methods”, in Adaptive and multilevel metaheuristics. Springer, 2008, str. 239–253.

[284] Gehring, H., Homberger, J., “A parallel two-phase metaheuristic for routing problems with time windows”, Asia-Pacific Journal of Operational Research, Vol. 18, No. 1, 2001, str. 35.

[285] Chen, P., Huang, H.-k., Dong, X.-Y., “Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem”, Expert Systems with Applications, Vol. 37, No. 2, 2010, str. 1620–1627.

[286] Subramanian, A., Uchoa, E., Ochi, L. S., “A hybrid algorithm for a class of vehicle routing problems”, Computers & Operations Research, Vol. 40, No. 10, 2013, str. 2519–2531.

[287] Kytöjoki, J., Nuortio, T., Bräysy, O., Gendreau, M., “An efficient variable neighborhood search heuristic for very large scale vehicle routing problems”, Computers & operations research, Vol. 34, No. 9, 2007, str. 2743–2757.

[288] Prins, C., “A simple and effective evolutionary algorithm for the vehicle routing problem”, Computers & operations research, Vol. 31, No. 12, 2004, str. 1985–2002.

[289] Prins, C., “A GRASP× evolutionary local search hybrid for the vehicle routing problem”, in Bio-inspired algorithms for the vehicle routing problem. Springer, 2009, str. 35–53.

[290] Hashimoto, H., Yagiura, M., Ibaraki, T., “An iterated local search algorithm for the time-dependent vehicle routing problem with time windows”, Discrete Optimization, Vol. 5, No. 2, 2008, str. 434–456.

[291] Gendreau, M., Hertz, A., Laporte, G., “A tabu search heuristic for the vehicle routing problem”, Management science, Vol. 40, No. 10, 1994, str. 1276–1290.

[292] Pinedo, M., Scheduling. Springer, 2012, Vol. 29.

[293] Garey, M. R., Johnson, D. S., Sethi, R., “The complexity of flowshop and jobshop scheduling”, Mathematics of operations research, Vol. 1, No. 2, 1976, str. 117–129.

[294] Burke, E., Jackson, K., Kingston, J. H., Weare, R., “Automated university timetabling: The state of the art”, The computer journal, Vol. 40, No. 9, 1997, str. 565–571.

[295] Weare, R., Burke, E., Elliman, D., “A hybrid genetic algorithm for highly constrained timetabling problems”, Department of Computer Science, 1995.

[296] Burke, E. K., Newall, J. P., Weare, R. F., “A memetic algorithm for university exam timetabling”, in international conference on the practice and theory of automated timetabling. Springer, 1995, str. 241–250.

[297] Socha, K., Sampels, M., Manfrin, M., “Ant algorithms for the university course timetabling problem with regard to the state-of-the-art”, in Workshops on Applications of Evolutionary Computation. Springer, 2003, str. 334–345.

[298] Corne, D., Ross, P., Fang, H.-L., “Fast practical evolutionary timetabling”, in AISB Workshop on Evolutionary Computing. Springer, 1994, str. 250–263.

[299] Coffman, E., Coffman, E., Coffman, E., Bruno, J., Computer and Job-shop Scheduling Theory, ser. Wiley-Interscience publication. Wiley, 1976.

[300] Błażewicz, J., Domschke, W., Pesch, E., "The job shop scheduling problem: Conventional and new solution techniques", European journal of operational research, Vol. 93, No. 1, 1996, str. 1–33.

[301] Adams, J., Balas, E., Zawack, D., "The shifting bottleneck procedure for job shop scheduling", Management science, Vol. 34, No. 3, 1988, str. 391–401.

[302] Brucker, P., Jurisch, B., Sievers, B., "A branch and bound algorithm for the job-shop scheduling problem", Discrete applied mathematics, Vol. 49, No. 1-3, 1994, str. 107–127.

[303] Van Laarhoven, P. J., Aarts, E. H., Lenstra, J. K., "Job shop scheduling by simulated annealing", Operations research, Vol. 40, No. 1, 1992, str. 113–125.

[304] Yamada, T., Nakano, R., "A genetic algorithm applicable to large-scale job-shop problems.", in PPSN, Vol. 2, 1992, str. 281–290.

[305] Davis, L., "Job shop scheduling with genetic algorithms", in Proceedings of an international conference on genetic algorithms and their applications, Vol. 140, 1985.

[306] Colorni, A., Dorigo, M., Maniezzo, V., Trubian, M., "Ant system for job-shop scheduling", JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science, Vol. 34, No. 1, 1994, str. 39–53.

[307] Tsai, C.-W., Rodrigues, J. J., "Metaheuristic scheduling for cloud: A survey", IEEE Systems Journal, Vol. 8, No. 1, 2013, str. 279–291.

[308] Javanmardi, S., Shojafar, M., Amendola, D., Cordeschi, N., Liu, H., Abraham, A., "Hybrid job scheduling algorithm for cloud computing environment", in Proceedings of the fifth international conference on innovations in bio-inspired computing and applications IBICA 2014. Springer, 2014, str. 43–52.

[309] Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H. S.-H., Li, Y., "Cloud computing resource scheduling and a survey of its evolutionary approaches", ACM Computing Surveys (CSUR), Vol. 47, No. 4, 2015, str. 1–33.

[310] Kumar, D., Raza, Z., "A pso based vm resource scheduling model for cloud computing", in 2015 IEEE International Conference on Computational Intelligence & Communication Technology. IEEE, 2015, str. 213–219.

[311] Pandey, S., Wu, L., Guru, S. M., Buyya, R., "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments", in 2010 24th IEEE international conference on advanced information networking and applications. IEEE, 2010, str. 400–407.

[312] Genaud, S., Gossa, J., "Cost-wait trade-offs in client-side resource provisioning with elastic clouds", in 2011 IEEE 4th International Conference on Cloud Computing. IEEE, 2011, str. 1–8.

[313] Sindelar, M., Sitaraman, R. K., Shenoy, P., "Sharing-aware algorithms for virtual machine colocation", in Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures, 2011, str. 367–378.

[314] Ghribi, C., Hadji, M., Zeghlache, D., "Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms", in 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. IEEE, 2013, str. 671–678.

[315] Grange, A., Kacem, I., Martin, S., "Algorithms for the bin packing problem with overlapping items", Computers & Industrial Engineering, Vol. 115, 2018, str. 331–341.

[316] Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X., Gu, Z., "Online optimization for scheduling preemptable tasks on iaas cloud systems", Journal of Parallel and Distributed Computing, Vol. 72, No. 5, 2012, str. 666–677.

[317] Lin, C.-C., Liu, P., Wu, J.-J., "Energy-aware virtual machine dynamic provision and scheduling for cloud computing", in 2011 IEEE 4th International Conference on Cloud Computing. IEEE, 2011, str. 736–737.

[318] Takahashi, S., Nakada, H., Takefusa, A., Kudoh, T., Shigeno, M., Yoshise, A., "Virtual machine packing algorithms for lower power consumption", in 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings. IEEE, 2012, str. 161–168.

[319] Garg, S. K., Yeo, C. S., Anandasivam, A., Buyya, R., "Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers", Journal of Parallel and Distributed Computing, Vol. 71, No. 6, 2011, str. 732–749.

[320] Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B. *et al.*, "A comparison of the performance of different metaheuristics on the timetabling problem", in International Conference on the Practice and Theory of Automated Timetabling. Springer, 2002, str. 329–351.

[321] Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O., “An effective hybrid algorithm for university course timetabling”, Journal of Scheduling, Vol. 9, No. 5, 2006, str. 403–432.

[322] Kostuch, P., “The university course timetabling problem with a three-phase approach”, in International Conference on the Practice and Theory of Automated Timetabling. Springer, 2004, str. 109–125.

[323] Petrovic, S., Burke, E. K., “University timetabling.”, 2004.

[324] “Metaheuristic network”, available at: http://www.metaheuristics.net/ (July 2019).

[325] Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L., “Personnel scheduling: A literature review”, European Journal of Operational Research, Vol. 226, No. 3, 2013, str. 367–385.

[326] De Causmaecker, P., Demeester, P., Berghe, G. V., Verbeke, B., “Analysis of real-world personnel scheduling problems”, in Proceedings of the 5th international conference on practice and theory of automated timetabling, Pittsburgh, 2004, str. 183–197.

[327] Avramidis, A. N., Chan, W., Gendreau, M., L’ecuyer, P., Pisacane, O., “Optimizing daily agent scheduling in a multiskill call center”, European Journal of Operational Research, Vol. 200, No. 3, 2010, str. 822–832.

[328] Bartholdi III, J. J., Orlin, J. B., Ratliff, H. D., “Cyclic scheduling via integer programs with circular ones”, Operations Research, Vol. 28, No. 5, 1980, str. 1074–1085.

[329] Pinedo, M., Zacharias, C., Zhu, N., “Scheduling in the service industries: An overview”, Journal of Systems Science and Systems Engineering, Vol. 24, No. 1, 2015, str. 1–48.

[330] Slack, N., Chambers, S., Johnston, R., Operations management. Pearson education, 2010.

[331] Pankaj, S., Call Centre. A.P.H. Publishing Corporation, 2005.

[332] Atlason, J., Epelman, M. A., Henderson, S. G., “Call center staffing with simulation and cutting plane methods”, Annals of Operations Research, Vol. 127, No. 1-4, 2004, str. 333–358.

[333] Henderson, S., Mason, A., Ziedins, I., Thomson, R., “A heuristic for determining efficient staffing requirements for call centres”, Citeseer, Tech. Rep., 1999.

[334] Gans, N., Shen, H., Zhou, Y., Korolev, K., McCord, A., Ristock, H., "Parametric stochastic programming models for call-center workforce scheduling", working paper, Tech. Rep., 2009.

[335] Bhulai, S., Koole, G., Pot, A., "Simple methods for shift scheduling in multiskill call centers", Manufacturing & Service Operations Management, Vol. 10, No. 3, 2008, str. 411–420.

[336] Cezik, M. T., L'Ecuyer, P., "Staffing multiskill call centers via linear programming and simulation", Management Science, Vol. 54, No. 2, 2008, str. 310–323.

[337] Alfares, H. K., "Operator staffing and scheduling for an it-help call centre", European Journal of Industrial Engineering, Vol. 1, No. 4, 2007, str. 414–430.

[338] Dietz, D. C., "Practical scheduling for call center operations", Omega, Vol. 39, No. 5, 2011, str. 550–557.

[339] Fukunaga, A., Hamilton, E., Fama, J., Andre, D., Matan, O., Nourbakhsh, I., "Staff scheduling for inbound call and customer contact centers", AI Magazine, Vol. 23, No. 4, 2002, str. 30.

[340] Morwood, J., Taylor, J., The pocket Oxford classical Greek dictionary. Oxford University Press, USA, 2002.

[341] ""meta.", merriam-webster.com dictionary, merriam-webster", available at: https://www.merriam-webster.com/dictionary/meta , accessed Feb. 2020.

[342] ""heuristic.", merriam-webster.com dictionary, merriam-webster", available at: https://www.merriam-webster.com/dictionary/heuristic , accessed Feb. 2020.

[343] Luke, S., Essentials of Metaheuristics, 2nd ed. Lulu, 2013, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[344] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., "Hyper-heuristics: A survey of the state of the art", Journal of the Operational Research Society, Vol. 64, No. 12, 2013, str. 1695–1724.

[345] Bouyssou, D., Forder, R., Merchant, S., Nance, R., Pierskalla, W., Roper, M., Ryan, D., van der Duyn Schouten, F., "Review of research status of operational research in the uk", EPSRC/ESRC/ORS. Swindon, UK, Tech. Rep., 2004.

[346] McCollum, B., "A perspective on bridging the gap between theory and practice in university timetabling", in International Conference on the Practice and Theory of Automated Timetabling. Springer, 2006, str. 3–23.

[347] Škvorc, U., Eftimov, T., Korošec, P., “Gecco black-box optimization competitions: progress from 2009 to 2018”, in Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, str. 275–276.

[348] Loshchilov, I., Schoenauer, M., Sebag, M., “Black-box optimization benchmarking of ipop-saacm-es and bipop-saacm-es on the bbob-2012 noiseless testbed”, in Proceedings of the 14th annual conference companion on Genetic and evolutionary computation, 2012, str. 175–182.

[349] Loshchilov, I., Schoenauer, M., Sebag, M., “Bi-population cma-es agorithms with surrogate models and line searches”, in Proceedings of the 15th annual conference companion on Genetic and evolutionary computation, 2013, str. 1177–1184.

[350] Auger, A., Finck, S., Hansen, N., Ros, R., “Bbob 2009: Comparison tables of all algorithms on all noiseless functions”, 2010.

[351] Dorigo, M., Stützle, T., “Ant colony optimization: overview and recent advances”, in Handbook of metaheuristics. Springer, 2019, str. 311–351.

[352] De Backer, B., Furnon, V., Shaw, P., Kilby, P., Prosser, P., “Solving vehicle routing problems using constraint programming and metaheuristics”, Journal of Heuristics, Vol. 6, No. 4, 2000, str. 501–523.

[353] Burke, E., De Causmaecker, P., Petrovic, S., Berghe, G. V., “Variable neighbourhood search for nurse rostering problems”, in MCG RESENDE & J. PINHO DE SOUSA (EDS.), METAHEURISTICS: COMPUTER DECISION-MAKING, CHAPTER 7, KLUWER. Citeseer, 2002.

[354] Eskandari, H., Mahmoodi, E., Fallah, H., Geiger, C. D., “Performance analysis of comercial simulation-based optimization packages: Optquest and witness optimizer”, in Proceedings of the 2011 Winter Simulation Conference (WSC). IEEE, 2011, str. 2358–2368.

[355] Rogers, P., “Optimum-seeking simulation in the design and control of manufacturing systems: experience with optquest for arena”, in Proceedings of the winter simulation conference, Vol. 2. IEEE, 2002, str. 1142–1150.

[356] “Opttek systems”, available at: https://www.opttek.com/ (January 2020).

[357] Raidl, G. R., Puchinger, J., Blum, C., “Metaheuristic hybrids”, in Handbook of metaheuristics. Springer, 2010, str. 469–496.

[358] Rothberg, E., “An evolutionary algorithm for polishing mixed integer programming solutions”, INFORMS Journal on Computing, Vol. 19, No. 4, 2007, str. 534–541.

[359] “Ilog cplex optimization studio - overview”, available at: https://www.ibm.com/products/ilog-cplex-optimization-studio (January 2020).

[360] Glover, F., Sörensen, K., “Metaheuristics”, Scholarpedia, Vol. 10, No. 4, 2015, str. 6532, revision #149834.

[361] “Coin-or: Computational infrastructure for operations research”, available at: https://www.coin-or.org/ (January 2020).

[362] “Optaplanner - constraint satisfaction solver (java™, open source)”, available at: https://www.optaplanner.org/ (January 2020).

[363] “Localsolver : Home”, available at: https://www.localsolver.com/ (December 2019).

[364] Glover, F., Kochenberger, G., “Handbook of metaheuristics. 2003”.

[365] Sörensen, K., “Metaheuristics—the metaphor exposed”, International Transactions in Operational Research, Vol. 22, No. 1, 2015, str. 3–18.

[366] Blum, C., Roli, A., Sampels, M., Hybrid metaheuristics: an emerging approach to optimization. Springer, 2008, Vol. 114.

[367] Raidl, G. R., “A unified view on hybrid metaheuristics”, in International workshop on hybrid metaheuristics. Springer, 2006, str. 1–12.

[368] Gutjahr, W. J., Montemanni, R., “Stochastic search in metaheuristics”, in Handbook of Metaheuristics. Springer, 2019, str. 513–540.

[369] Gendreau, M., Potvin, J.-Y., Tabu Search. Cham: Springer International Publishing, 2019, str. 37–55, available at: https://doi.org/10.1007/978-3-319-91086-4_2

[370] Delahaye, D., Chaimatanan, S., Mongeau, M., “Simulated annealing: From basics to applications”, in Handbook of Metaheuristics. Springer, 2019, str. 1–35.

[371] Dorigo, M., Maniezzo, V., Colorni, A., “Positive feedback as a search strategy”, 1991.

[372] De Jong, K. A., “Analysis of the behavior of a class of genetic adaptive systems”, Tech. Rep., 1975.

[373] Whitley, D., “Next generation genetic algorithms: A user’s guide and tutorial”, in Handbook of Metaheuristics. Springer, 2019, str. 245–274.

[374] Rudolph, G., “Convergence analysis of canonical genetic algorithms”, IEEE transactions on neural networks, Vol. 5, No. 1, 1994, str. 96–101.

[375] Stützle, T., "Local search algorithms for combinatorial problems", Darmstadt University of Technology PhD Thesis, Vol. 20, 1998.

[376] Glover, F., Laguna, M., "Tabu search", in Handbook of combinatorial optimization. Springer, 1998, str. 2093–2229.

[377] Tarantilis, C. D., Kiranoudis, C. T., "Boneroute: An adaptive memory-based method for effective fleet management", Annals of operations Research, Vol. 115, No. 1-4, 2002, str. 227–241.

[378] Cotta, C., Sevaux, M., Sörensen, K., Adaptive and multilevel metaheuristics. Springer, 2008, Vol. 136.

[379] Battiti, R., Brunato, M., Mascia, F., Reactive search and intelligent optimization. Springer Science & Business Media, 2008, Vol. 45.

[380] Battiti, R., Brunato, M., "Reactive search optimization: learning while optimizing", in Handbook of Metaheuristics. Springer, 2010, str. 543–571.

[381] Burke, E. K., Li, J., Qu, R., "A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems", European Journal of Operational Research, Vol. 203, No. 2, 2010, str. 484–493.

[382] Ray, T., Kang, T., Chye, S. K., "An evolutionary algorithm for constrained optimization", in Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Inc., 2000, str. 771–777.

[383] Colorni, A., Dorigo, M., Maniezzo, V., "Genetic algorithms and highly constrained problems: The time-table case", in International Conference on Parallel Problem Solving from Nature. Springer, 1990, str. 55–59.

[384] Coello, C. A. C., Carlos, A., "A survey of constraint handling techniques used with evolutionary algorithms", Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada, 1999.

[385] Dasgupta, D., Michalewicz, Z., Evolutionary algorithms in engineering applications. Springer Science & Business Media, 2013.

[386] Coello, C. A. C., "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art", Computer methods in applied mechanics and engineering, Vol. 191, No. 11-12, 2002, str. 1245–1287.

[387] Michalewicz, Z., Janikow, C. Z., "Handling constraints in genetic algorithms.", in ICGA, 1991, str. 151–157.

[388] Michalewicz, Z., Dasgupta, D., Le Riche, R. G., Schoenauer, M., “Evolutionary algorithms for constrained engineering problems”, Computers & Industrial Engineering, Vol. 30, No. 4, 1996, str. 851–870.

[389] Michalewicz, Z., “Constraint-handling techniques—introduction”, in Handbook of Evolutionary Computation. IOP Publishing Ltd., Techno House Redcliffe Way Bristol BS1 6NX, United Kingdom, 1997, str. C5.1:1–C5.7.8 (344–377).

[390] Bäck, T., Fogel, D. B., Michalewicz, Z., Handbook of evolutionary computation. CRC Press, 1997.

[391] Eiben, A., “Evolutionary algorithms and constraint satisfaction: Definitions, survey, methodology, and research directions”, in Theoretical aspects of evolutionary computing. Springer, 2001, str. 13–30.

[392] Lewis, R., “A survey of metaheuristic-based techniques for university timetabling problems”, OR spectrum, Vol. 30, No. 1, 2008, str. 167–190.

[393] Maniezzo, V., Milandri, M., “An ant-based framework for very strongly constrained problems”, in International Workshop on Ant Algorithms. Springer, 2002, str. 222–227.

[394] Raidl, G. R., Puchinger, J., Blum, C., Metaheuristic Hybrids. Cham: Springer International Publishing, 2019, str. 385–417, available at: https://doi.org/10.1007/978-3-319-91086-4_12

[395] Molnar, G., Jakobović, D., Pavelić, M., “Workforce scheduling in inbound customer call centres with a case study”, in European Conference on the Applications of Evolutionary Computation. Springer, 2016, str. 831–846.

[396] Michalewicz, Z., “Constraint-handling techniques—constraint-preserving operators”, in Handbook of Evolutionary Computation. IOP Publishing Ltd., Techno House Redcliffe Way Bristol BS1 6NX, United Kingdom, 1997, str. C5.5:1–C5.5.5 (361–365).

[397] Blum, C., Puchinger, J., Raidl, G. R., Roli, A., “Hybrid metaheuristics in combinatorial optimization: A survey”, Applied soft computing, Vol. 11, No. 6, 2011, str. 4135–4151.

[398] Eiben, A. E., Ruttkay, Z., “Constraint-handling techniques—constraint-satisfaction problems”, in Handbook of Evolutionary Computation. IOP Publishing Ltd., Techno House Redcliffe Way Bristol BS1 6NX, United Kingdom, 1997, str. C5.7:1–C5.7.8 (370–377).

[399] Feltl, H., Raidl, G. R., “An improved hybrid genetic algorithm for the generalized assignment problem”, in Proceedings of the 2004 ACM symposium on Applied computing. ACM, 2004, str. 990–995.

[400] Colorni, A., Dorigo, M., Maniezzo, V., “Metaheuristics for high school timetabling”, Computational optimization and applications, Vol. 9, No. 3, 1998, str. 275–298.

[401] Michalewicz, Z., “Constraint-handling techniques—repair algorithms”, in Handbook of Evolutionary Computation. IOP Publishing Ltd., Techno House Redcliffe Way Bristol BS1 6NX, United Kingdom, 1997, str. C5.4:1–C5.4.5 (356–360).

[402] Beddoe, G., Petrovic, S., Li, J., “A hybrid metaheuristic case-based reasoning system for nurse rostering”, Journal of Scheduling, Vol. 12, No. 2, 2009, str. 99.

[403] Duarte, A. R., Ribeiro, C. C., Urrutia, S., “A hybrid ILS heuristic to the referee assignment problem with an embedded MIP strategy”, in International Workshop on Hybrid Metaheuristics. Springer, 2007, str. 82–95.

[404] Smith, A. E., Coit, D. E., “Constraint-handling techniques—penalty functions”, in Handbook of Evolutionary Computation. IOP Publishing Ltd., Techno House Redcliffe Way Bristol BS1 6NX, United Kingdom, 1997, str. C5.2:1–C5.2.6 (347–352).

[405] Talbi, E.-G., Metaheuristics: from design to implementation. John Wiley & Sons, 2009, Vol. 74.

[406] Ehrgott, M., Multicriteria optimization. Springer Science & Business Media, 2005, Vol. 491.

[407] Coello, C. A. C., “A comprehensive survey of evolutionary-based multiobjective optimization techniques”, Knowledge and Information systems, Vol. 1, No. 3, 1999, str. 269–308.

[408] Sawaragi, Y., NAKAYAMA, H., TANINO, T., Theory of multiobjective optimization. Elsevier, 1985.

[409] Michalewicz, Z., “Genetic algorithms, numerical optimization, and constraints”, in Proceedings of the sixth international conference on genetic algorithms, Vol. 195, 1995, str. 151–158.

[410] Souza, M. J. F., Maculan, N., Ochi, L. S., “A grasp-tabu search algorithm for solving school timetabling problems”, in Metaheuristics: Computer decision-making. Springer, 2003, str. 659–672.

[411] Talbi, E.-G., “A taxonomy of hybrid metaheuristics”, Journal of heuristics, Vol. 8, No. 5, 2002, str. 541–564.

[412] Glover, F., Laguna, M., “General purpose heuristics for integer programming—part i”, Journal of Heuristics, Vol. 2, No. 4, 1997, str. 343–358.

[413] Yagiura, M., Ibaraki, T., "On metaheuristic algorithms for combinatorial optimization problems", Systems and Computers in Japan, Vol. 32, No. 3, 2001, str. 33–55.

[414] Eiben, A. E., Schippers, C. A., "On evolutionary exploration and exploitation", Fundamenta Informaticae, Vol. 35, No. 1-4, 1998, str. 35–50.

[415] Battiti, R., "Reactive search: Toward self-tuning heuristics", Modern heuristic search methods, Vol. 61, 1996, str. 83.

[416] Mitchell, M., An introduction to genetic algorithms. MIT press, 1998.

[417] Stützle, T., Dorigo, M. *et al.*, "Aco algorithms for the traveling salesman problem", Evolutionary algorithms in engineering and computer science, Vol. 4, 1999, str. 163–183.

[418] Zhigljavsky, A. A., Theory of global random search. Springer Science & Business Media, 2012, Vol. 65.

[419] Anderson, R. L., "Recent advances in finding best operating conditions", Journal of the American Statistical Association, Vol. 48, No. 264, 1953, str. 789–798.

[420] Brooks, S. H., "A discussion of random methods for seeking maxima", Operations research, Vol. 6, No. 2, 1958, str. 244–251.

[421] Karnopp, D. C., "Random search techniques for optimization problems", Automatica, Vol. 1, No. 2-3, 1963, str. 111–121.

[422] Watson, J.-P., "An introduction to fitness landscape analysis and cost models for local search", in Handbook of metaheuristics. Springer, 2010, str. 599–623.

[423] Langdon, W. B., McKay, R. I., Spector, L., "Genetic programming", in Handbook of metaheuristics. Springer, 2010, str. 185–225.

[424] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M., Complexity and approximation: Combinatorial optimization problems and their approximability properties. Springer Science & Business Media, 2012.

[425] Hirsch, M. J., Meneses, C., Pardalos, P. M., Resende, M. G., "Global optimization by continuous grasp", Optimization Letters, Vol. 1, No. 2, 2007, str. 201–212.

[426] Bang-Jensen, J., Gutin, G., Yeo, A., "When the greedy algorithm fails", Discrete optimization, Vol. 1, No. 2, 2004, str. 121–127.

[427] Potvin, J.-Y., Rousseau, J.-M., "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows", European Journal of Operational Research, Vol. 66, No. 3, 1993, str. 331–340.

[428] Shore, H. H., “The transportation problem and the vogel approximation method”, Decision Sciences, Vol. 1, No. 3-4, 1970, str. 441–457.

[429] Hromkovič, J., Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics. Springer Science & Business Media, 2013.

[430] Lin, S., Kernighan, B. W., “An effective heuristic algorithm for the traveling-salesman problem”, Operations research, Vol. 21, No. 2, 1973, str. 498–516.

[431] Helsgaun, K., “An effective implementation of the lin–kernighan traveling salesman heuristic”, European Journal of Operational Research, Vol. 126, No. 1, 2000, str. 106–130.

[432] Moscato, P., Cotta, C., “An accelerated introduction to memetic algorithms”, in Handbook of Metaheuristics. Springer, 2019, str. 275–309.

[433] Johnson, D. S., Aragon, C. R., McGeoch, L. A., Schevon, C., “Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning”, Operations research, Vol. 37, No. 6, 1989, str. 865–892.

[434] Johnson, D. S., McGeoch, L. A., “The traveling salesman problem: A case study in local optimization”.

[435] Schreiber, G. R., Martin, O. C., “Cut size statistics of graph bisection heuristics”, SIAM Journal on Optimization, Vol. 10, No. 1, 1999, str. 231–251.

[436] Duarte, A. R., Ribeiro, C. C., Urrutia, S., Haeusler, E. H., “Referee assignment in sports leagues”, in International Conference on the Practice and Theory of Automated Timetabling. Springer, 2006, str. 158–173.

[437] Nascimento, M. C., Resende, M., Toledo, F. M., “Grasp with path-relinking for the multi-plant capacitated lot sizing problem”, European Journal of Operational Research, 2008.

[438] Baum, E., “Iterated descent: A better algorithm for local search in combinatorial optimization problems”, Manuscript, 1986.

[439] Baum, E. B., “Towards practical ‘neural’computation for combinatorial optimization problems”, in AIP Conference Proceedings, Vol. 151, No. 1. American Institute of Physics, 1986, str. 53–58.

[440] Martin, O., Otto, S. W., Felten, E. W., Large-step Markov chains for the traveling salesman problem. Citeseer, 1991.

[441] Johnson, D. S., “Local optimization and the traveling salesman problem”, in International colloquium on automata, languages, and programming. Springer, 1990, str. 446–461.

[442] Martin, O. C., Otto, S. W., “Combining simulated annealing with local search heuristics”, Annals of Operations Research, Vol. 63, No. 1, 1996, str. 57–75.

[443] De Paula, M. R., Ravetti, M. G., Mateus, G. R., Pardalos, P. M., “Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search”, IMA Journal of Management Mathematics, Vol. 18, No. 2, 2007, str. 101–115.

[444] de Werra, D., Hertz, A., “Tabu search techniques”, Operations-Research-Spektrum, Vol. 11, No. 3, 1989, str. 131–141.

[445] The Editors of Encyclopaedia Britannica, “Annealing”, available at: https://www.britannica.com/science/annealing-heat-treatment 2011.

[446] Aarts, E., Aarts, E., Korst, J., Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing, ser. Wiley Series in Discrete Mathematics & Optimization. Wiley, 1989, available at: https://books.google.hr/books?id=K_pQAAAAMAAJ

[447] Černỳ, V., “Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm”, Journal of optimization theory and applications, Vol. 45, No. 1, 1985, str. 41–51.

[448] Cardoso, M. F., Salcedo, R. L., de Azevedo, S. F., “Nonequilibrium simulated annealing: a faster approach to combinatorial minimization”, Industrial & engineering chemistry research, Vol. 33, No. 8, 1994, str. 1908–1918.

[449] Cohn, H., Fielding, M., “Simulated annealing: searching for an optimal temperature schedule”, SIAM Journal on Optimization, Vol. 9, No. 3, 1999, str. 779–802.

[450] Fox, B., Heine, G., “Simulated annealing with overrides”, Technical, Department of Mathematics, University of Colorado, Denver, Colorado, 1993.

[451] Nourani, Y., Andresen, B., “A comparison of simulated annealing cooling strategies”, Journal of Physics A: Mathematical and General, Vol. 31, No. 41, 1998, str. 8373.

[452] Ogbu, F., Smith, D. K., “The application of the simulated annealing algorithm to the solution of the n/m/cmax flowshop problem”, Computers & Operations Research, Vol. 17, No. 3, 1990, str. 243–253.

[453] Goss, S., Aron, S., Deneubourg, J.-L., Pasteels, J. M., "Self-organized shortcuts in the argentine ant", Naturwissenschaften, Vol. 76, No. 12, 1989, str. 579–581.

[454] PP, G., "La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la thorie de la stigmergie: essai d'interprtation du comportement des termites constructeurs", Insectes Sociaux, Vol. 6, 1959, str. 4181.

[455] Socha, K., Dorigo, M., "Ant colony optimization for continuous domains", European journal of operational research, Vol. 185, No. 3, 2008, str. 1155–1173.

[456] Dorigo, M., Gambardella, L. M., "Ant colony system: a cooperative learning approach to the traveling salesman problem", IEEE Transactions on evolutionary computation, Vol. 1, No. 1, 1997, str. 53–66.

[457] Stutzle, T., Hoos, H., "Max-min ant system and local search for the traveling salesman problem", in Proceedings of 1997 IEEE international conference on evolutionary computation (ICEC'97). IEEE, 1997, str. 309–314.

[458] Stützle, T., Hoos, H., "Improvements on the ant-system: Introducing the max-min ant system", in Artificial neural nets and genetic algorithms. Springer, 1998, str. 245–249.

[459] Stützle, T., Hoos, H., "The max-min ant system and local search for combinatorial optimization problems", in Meta-heuristics. Springer, 1999, str. 313–329.

[460] Turing, A. M., "Intelligent machinery", 1948.

[461] Rechenberg, I., "Evolution strategy: Nature's way of optimization", in Optimization: Methods and applications, possibilities and limitations. Springer, 1989, str. 106–126.

[462] Schwefel, H.-P., "Evolutionsstrategien für die numerische optimierung", in Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Springer, 1977, str. 123–176.

[463] Fogel, L. J., "Toward inductive inference automata.", in IFIP Congress, Vol. 62, 1962, str. 395–400.

[464] Darwin, C., On the Origin of Species by Means of Natural Selection Or the Preservation of Favoured Races in the Struggle for Life. H. Milford; Oxford University Press, 1859.

[465] Stearns, S. C., Hoekstra, R. F., Evolution, an introduction. Oxford University Press, 2000.

[466] Whitley, D., "A genetic algorithm tutorial", Statistics and computing, Vol. 4, No. 2, 1994, str. 65–85.

[467] Baker, J. E., “Reducing bias and inefficiency in the selection algorithm”, in Proceedings of the second international conference on genetic algorithms, Vol. 206, 1987, str. 14–21.

[468] Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., De Garis, H., “An overview of evolutionary computation”, in European Conference on Machine Learning. Springer, 1993, str. 442–459.

[469] Calégari, P., Coray, G., Hertz, A., Kobler, D., Kuonen, P., “A taxonomy of evolutionary algorithms in combinatorial optimization”, Journal of Heuristics, Vol. 5, No. 2, 1999, str. 145–158.

[470] Moscato, P. *et al.*, “On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms”, Caltech concurrent computation program, C3P Report, Vol. 826, 1989, str. 1989.

[471] Deb, K., Agrawal, S., “Understanding interactions among genetic algorithm parameters”, in in Foundations of Genetic Algorithms 5. Morgan Kaufmann, 1999, str. 265–286.

[472] Audet, C., Denni, J., Moore, D., Booker, A., Frank, P., “A surrogate-model-based method for constrained optimization”, in 8th symposium on multidisciplinary analysis and optimization, 2000, str. 4891.

[473] Basudhar, A., Dribusch, C., Lacaze, S., Missoum, S., “Constrained efficient global optimization with support vector machines”, Structural and Multidisciplinary Optimization, Vol. 46, No. 2, 2012, str. 201–221.

[474] Bouhlel, M. A., Bartoli, N., Otsmane, A., Morlier, J., “Improving kriging surrogates of high-dimensional design models by partial least squares dimension reduction”, Structural and Multidisciplinary Optimization, Vol. 53, No. 5, 2016, str. 935–952.

[475] Boukouvala, F., Hasan, M. F., Floudas, C. A., “Global optimization of general constrained grey-box models: new method and its application to constrained pdes for pressure swing adsorption”, Journal of Global Optimization, Vol. 67, No. 1-2, 2017, str. 3–42.

[476] Bagheri, S., Konen, W., Allmendinger, R., Branke, J., Deb, K., Fieldsend, J., Quagliarella, D., Sindhya, K., “Constraint handling in efficient global optimization”, in Proceedings of the Genetic and Evolutionary Computation Conference, 2017, str. 673–680.

[477] Bagheri, S., Konen, W., Emmerich, M., Bäck, T., “Self-adjusting parameter control for surrogate-assisted constrained optimization under limited budgets”, Applied Soft Computing, Vol. 61, 2017, str. 377–393.

[478] Amine Bouhlel, M., Bartoli, N., Regis, R. G., Otsmane, A., Morlier, J., “Efficient global optimization for high-dimensional constrained problems by using the kriging models combined with the partial least squares method”, Engineering Optimization, Vol. 50, No. 12, 2018, str. 2038–2053.

[479] Regis, R. G., “A survey of surrogate approaches for expensive constrained black-box optimization”, in World Congress on Global Optimization. Springer, 2019, str. 37–47.

[480] Hüsken, M., Jin, Y., Sendhoff, B., “Structure optimization of neural networks for evolutionary design optimization”, Soft Computing, Vol. 9, No. 1, 2005, str. 21–28.

[481] Jordan, M. I., Mitchell, T. M., “Machine learning: Trends, perspectives, and prospects”, Science, Vol. 349, No. 6245, 2015, str. 255–260.

[482] Michie, D., Spiegelhalter, D. J., Taylor, C. *et al.*, “Machine learning”, Neural and Statistical Classification, Vol. 13, No. 1994, 1994, str. 1–298.

[483] Montabone, L., Forget, F., Millour, E., Wilson, R., Lewis, S., Cantor, B., Kass, D., Kleinböhl, A., Lemmon, M., Smith, M. *et al.*, “Eight-year climatology of dust optical depth on mars”, Icarus, Vol. 251, 2015, str. 65–95.

[484] Goel, T., Haftka, R. T., Shyy, W., Queipo, N. V., “Ensemble of surrogates”, Structural and Multidisciplinary Optimization, Vol. 33, No. 3, 2007, str. 199–216.

[485] Gräning, L., Jin, Y., Sendhoff, B., “Efficient evolutionary optimization using individual-based evolution control and neural networks: A comparative study.”, in ESANN, 2005, str. 273–278.

[486] Viana, F. A., Haftka, R. T., Watson, L. T., “Efficient global optimization algorithm assisted by multiple surrogate techniques”, Journal of Global Optimization, Vol. 56, No. 2, 2013, str. 669–689.

[487] Zhou, Z., Ong, Y. S., Nair, P. B., Keane, A. J., Lum, K. Y., “Combining global and local surrogate models to accelerate evolutionary optimization”, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), Vol. 37, No. 1, 2006, str. 66–76.

[488] Zhou, Z., Ong, Y. S., Nair, P. B., “Hierarchical surrogate-assisted evolutionary optimization framework”, in Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753), Vol. 2. IEEE, 2004, str. 1586–1593.

[489] Santana-Quintero, L. V., Montano, A. A., Coello, C. A. C., “A review of techniques for handling expensive functions in evolutionary multi-objective optimization”, in Computational intelligence in expensive optimization problems. Springer, 2010, str. 29–59.

[490] Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., Numerical recipes 3rd edition: The art of scientific computing. Cambridge university press, 2007.

[491] Goel, T., Vaidyanathan, R., Haftka, R. T., Shyy, W., Queipo, N. V., Tucker, K., “Response surface approximation of pareto optimal front in multi-objective optimization”, Computer methods in applied mechanics and engineering, Vol. 196, No. 4-6, 2007, str. 879–893.

[492] Cressie, N., “The origins of kriging”, Mathematical geology, Vol. 22, No. 3, 1990, str. 239–252.

[493] Goovaerts, P. *et al.*, Geostatistics for natural resources evaluation. Oxford University Press on Demand, 1997.

[494] Ripley, B. D., Spatial statistics. John Wiley & Sons, 2005, Vol. 575.

[495] Krige, D. G., “A statistical approach to some basic mine valuation problems on the witwatersrand”, Journal of the Southern African Institute of Mining and Metallurgy, Vol. 52, No. 6, 1951, str. 119–139.

[496] Randolph, M., “Current trends in mining”, SME mining engineering handbook, 3rd edn. Society for Mining, Metallurgy, and Exploration, Inc, Englewood, 2011, str. 11–20.

[497] Lee-Moreno, J. L., Darlin, P., “Minerals prospecting and exploration”, in SME Mining Engineering Handbook. Society for Mining, Metallurgy, Exploration, 2011, str. 105–112.

[498] Minnitt, R., Assibey-Bonsu, W., “Professor dg krige (1919–2013)”, 2013.

[499] Wackernagel, H., Multivariate geostatistics: an introduction with applications. Springer Science & Business Media, 2013.

[500] Jones, D. R., Schonlau, M., Welch, W. J., “Efficient global optimization of expensive black-box functions”, Journal of Global optimization, Vol. 13, No. 4, 1998, str. 455–492.

[501] Jones, D. R., “A taxonomy of global optimization methods based on response surfaces”, Journal of global optimization, Vol. 21, No. 4, 2001, str. 345–383.

[502] Jeong, S., Murayama, M., Yamamoto, K., “Efficient optimization design method using kriging model”, Journal of aircraft, Vol. 42, No. 2, 2005, str. 413–420.

[503] Simpson, T. W., Mauery, T. M., Korte, J. J., Mistree, F., "Kriging models for global approximation in simulation-based multidisciplinary design optimization", AIAA journal, Vol. 39, No. 12, 2001, str. 2233–2241.

[504] Huang, D., Allen, T. T., Notz, W. I., Zeng, N., "Global optimization of stochastic black-box systems via sequential kriging meta-models", Journal of global optimization, Vol. 34, No. 3, 2006, str. 441–466.

[505] Müller, B., Reinhardt, J., Strickland, M. T., Neural networks: an introduction. Springer Science & Business Media, 1995.

[506] Kröse, B., Krose, B., van der Smagt, P., Smagt, P., "An introduction to neural networks", 1993.

[507] Anderson, J. A., An introduction to neural networks. MIT press, 1995.

[508] Boser, B. E., Guyon, I. M., Vapnik, V. N., "A training algorithm for optimal margin classifiers", in Proceedings of the fifth annual workshop on Computational learning theory, 1992, str. 144–152.

[509] Guyon, I., Boser, B., Vapnik, V., "Automatic capacity tuning of very large vc-dimension classifiers", in Advances in neural information processing systems, 1993, str. 147–155.

[510] Vapnik, V., Golowich, S. E., Smola, A. J., "Support vector method for function approximation, regression estimation and signal processing", in Advances in neural information processing systems, 1997, str. 281–287.

[511] Vapnik, V., Chervonenkis, A., "Theory of pattern recognition", 1974.

[512] Steinwart, I., Christmann, A., Support vector machines. Springer Science & Business Media, 2008.

[513] Shawe-Taylor, J., Cristianini, N., Support vector machines. Cambridge University Press Cambridge, 2000, Vol. 2.

[514] Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., Scholkopf, B., "Support vector machines", IEEE Intelligent Systems and their applications, Vol. 13, No. 4, 1998, str. 18–28.

[515] Devillers, J., Neural networks in QSAR and drug design. Academic Press, 1996.

[516] Hong, Y.-S., Lee, H., Tahk, M.-J., "Acceleration of the convergence speed of evolutionary algorithms using multi-layer neural networks", Engineering Optimization, Vol. 35, No. 1, 2003, str. 91–102.

[517] Papadrakakis, M., Lagaros, N. D., Tsompanakis, Y., "Optimization of large-scale 3-d trusses using evolution strategies and neural networks", International Journal of Space Structures, Vol. 14, No. 3, 1999, str. 211–223.

[518] Bhattacharya, M., Lu, G., "A dynamic approximate fitness-based hybrid ea for optimization problems", in Proc. Congr. Evol. Comput, 2003, str. 1879–1886.

[519] Abboud, K., Schoenauer, M., "Surrogate deterministic mutation: Preliminary results", in International Conference on Artificial Evolution (Evolution Artificielle). Springer, 2001, str. 104–116.

[520] Ibrahim, D., Jobson, M., Li, J., Guillén-Gosálbez, G., "Optimization-based design of crude oil distillation units using surrogate column models and a support vector machine", Chemical Engineering Research and Design, Vol. 134, 2018, str. 212–225.

[521] Smith, R. E., Dike, B. A., Stegmann, S., "Fitness inheritance in genetic algorithms", in Proceedings of the 1995 ACM symposium on Applied computing, 1995, str. 345–350.

[522] Sastry, K., Goldberg, D. E., Pelikan, M., "Don't evaluate, inherit", in Proceedings of the Genetic and Evolutionary Computation Conference. San Francisco, California, USA: Morgan Kaufmann, 2001, str. 551–558.

[523] Ducheyne, E., De Baets, B., De Wulf, R., "Is fitness inheritance useful for real-world applications?", in International Conference on Evolutionary Multi-Criterion Optimization. Springer, 2003, str. 31–42.

[524] Jin, R., Chen, W., Simpson, T. W., "Comparative studies of metamodelling techniques under multiple modelling criteria", Structural and multidisciplinary optimization, Vol. 23, No. 1, 2001, str. 1–13.

[525] Lim, D., Ong, Y.-S., Jin, Y., Sendhoff, B., "A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation", in Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, str. 1288–1295.

[526] Carpenter, W., Barthelemy, J.-F., "A comparison of polynomial approximations and artificial neural nets as response surfaces", Structural Optimization, Vol. 5, No. 3, 1993, str. 166–174.

[527] Rasheed, K., Ni, X., Vattam, S., "Comparison of methods for developing dynamic reduced models for design optimization", Soft Computing, Vol. 9, No. 1, 2005, str. 29–37.

[528] Giunta, A., Watson, L., “A comparison of approximation modeling techniques-polynomial versus interpolating models”, in 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 1998, str. 4758.

[529] Simpson, T. W., Booker, A. J., Ghosh, D., Giunta, A. A., Koch, P. N., Yang, R.-J., “Approximation methods in multidisciplinary analysis and optimization: a panel discussion”, Structural and multidisciplinary optimization, Vol. 27, No. 5, 2004, str. 302–313.

[530] Simpson, T., Mistree, F., Korte, J., Mauery, T., “Comparison of response surface and kriging models for multidisciplinary design optimization”, in 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 1998, str. 4755.

[531] Willmes, L., Back, T., Jin, Y., Sendhoff, B., “Comparing neural networks and kriging for fitness approximation in evolutionary optimization”, in The 2003 Congress on Evolutionary Computation, 2003. CEC’03., Vol. 1. IEEE, 2003, str. 663–670.

[532] Jin, Y., Olhofer, M., Sendhoff, B., “On evolutionary optimization with approximate fitness functions.”, in GECCO, 2000, str. 786–793.

[533] Ong, Y., Keane, A. J., Nair, P. B., “Surrogate-assisted coevolutionary search”, in Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP’02., Vol. 3. IEEE, 2002, str. 1140–1145.

[534] Jin, Y., Olhofer, M., Sendhoff, B., “A framework for evolutionary optimization with approximate fitness functions”, IEEE Transactions on evolutionary computation, Vol. 6, No. 5, 2002, str. 481–494.

[535] Furuya, H., Haftka, R. T., “Combining genetic and deterministic algorithms for locating actuators on space structures”, Journal of spacecraft and rockets, Vol. 33, No. 3, 1996, str. 422–427.

[536] Liu, B., Zhang, Q., Gielen, G. G., “A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems”, IEEE Transactions on Evolutionary Computation, Vol. 18, No. 2, 2013, str. 180–192.

[537] Couckuyt, I., Declercq, F., Dhaene, T., Rogier, H., Knockaert, L., “Surrogate-based infill optimization applied to electromagnetic problems”, International Journal of RF and Microwave Computer-Aided Engineering, Vol. 20, No. 5, 2010, str. 492–501.

[538] Bernardino, H. S., Barbosa, H. J., Fonseca, L. G., “A faster clonal selection algorithm for expensive optimization problems”, in International Conference on Artificial Immune Systems. Springer, 2010, str. 130–143.

[539] Singh, H. K., Ray, T., Smith, W., “Surrogate assisted simulated annealing (sasa) for constrained multi-objective optimization”, in IEEE congress on evolutionary computation. IEEE, 2010, str. 1–8.

[540] Sun, C., Jin, Y., Cheng, R., Ding, J., Zeng, J., “Surrogate-assisted cooperative swarm optimization of high-dimensional expensive problems”, IEEE Transactions on Evolutionary Computation, Vol. 21, No. 4, 2017, str. 644–660.

[541] Yu, H., Tan, Y., Zeng, J., Sun, C., Jin, Y., “Surrogate-assisted hierarchical particle swarm optimization”, Information Sciences, Vol. 454, 2018, str. 59–72.

[542] Birattari, M., Kacprzyk, J., Tuning metaheuristics: a machine learning perspective. Springer, 2009, Vol. 197.

[543] Langdon, W. B., Poli, R., Foundations of genetic programming. Springer Science & Business Media, 2013.

[544] Epitropakis, M. G., Burke, E. K., “Hyper-heuristics”, Handbook of Heuristics, 2018, str. 489–545.

[545] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., Stützle, T., “The irace package: Iterated racing for automatic algorithm configuration”, Operations Research Perspectives, Vol. 3, 2016, str. 43–58.

[546] Stützle, T., López-Ibáñez, M., “Automated design of metaheuristic algorithms”, in Handbook of metaheuristics. Springer, 2019, str. 541–579.

[547] Pellegrini, P., Birattari, M., “Implementation effort and performance”, in International Workshop on Engineering Stochastic Local Search Algorithms. Springer, 2007, str. 31–45.

[548] Rice, J. R., “The algorithm selection problem”, in Advances in computers. Elsevier, 1976, Vol. 15, str. 65–118.

[549] Hooker, J. N., “Testing heuristics: We have it all wrong”, Journal of heuristics, Vol. 1, No. 1, 1995, str. 33–42.

[550] Fukunaga, A. S., “Genetic algorithm portfolios”, in Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512), Vol. 2. IEEE, 2000, str. 1304–1311.

[551] Gomes, C. P., Selman, B., “Algorithm portfolios”, Artificial Intelligence, Vol. 126, No. 1-2, 2001, str. 43–62.

[552] Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y., "A portfolio approach to algorithm selection", in IJCAI, Vol. 3, 2003, str. 1542–1543.

[553] Silberholz, J., Golden, B., "Comparison of metaheuristics", in Handbook of metaheuristics. Springer, 2010, str. 625–640.

[554] Stützle, T., "Some thoughts on engineering stochastic local search algorithms", in Proceedings of the EU/MEeting 2009: Debating the Future: New Areas of Application and Innovative Approaches, 2009, str. 47–52.

[555] Merelo, J.-J., Cotta, C., "Building bridges: the role of subfields in metaheuristics", ACM SIGEVOlution, Vol. 1, No. 4, 2006, str. 9–15.

[556] Maniezzo, V., Stützle, T., Voß, S., (ur.), Matheuristics - Hybridizing Metaheuristics and Mathematical Programming, ser. Annals of Information Systems. Springer, 2010.

[557] Birattari, M., Zlochin, M., Dorigo, M., "Towards a theory of practice in metaheuristics design: A machine learning perspective", RAIRO-Theoretical Informatics and Applications, Vol. 40, No. 2, 2006, str. 353–369.

[558] Teytaud, O., Vazquez, E., "Designing an optimal search algorithm with respect to prior information", in Theory and Principled Methods for the Design of Metaheuristics. Springer, 2014, str. 111–128.

[559] Burke, E. K., Curtois, T., Kendall, G., Hyde, M., Ochoa, G., Vazquez-Rodriguez, J. A., "Towards the decathlon challenge of search heuristics", in Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. ACM, 2009, str. 2205–2208.

[560] "Metaheuristic network - the course timetable problem", available at: http://www.metaheuristics.net/index.php%3Fmain=4&sub=44.html (July 2019).

[561] Beck, K., Test-driven development: by example. Addison-Wesley Professional, 2003.

[562] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. *et al.*, "Manifesto for agile software development", 2001.

[563] Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N. B., "A decade of agile methodologies: Towards explaining agile software development", 2012.

[564] "Sls2019: International workshop on stochastic local search algorithms", available at: https://sls2019.sciencesconf.org/ (July 2019).

[565] Zäpfel, G., Braune, R., Bögl, M., Metaheuristic search concepts: A tutorial with applications to production and logistics. Springer Science & Business Media, 2010.

[566] Rardin, R. L., Uzsoy, R., "Experimental evaluation of heuristic optimization algorithms: A tutorial", Journal of Heuristics, Vol. 7, No. 3, 2001, str. 261–304.

[567] Swan, J., Adriaensen, S., Bishr, M., Burke, E. K., Clark, J. A., De Causmaecker, P., Durillo, J., Hammond, K., Hart, E., Johnson, C. G. *et al.*, "A research agenda for metaheuristic standardization", in Proceedings of the XI metaheuristics international conference, 2015.

[568] Løkketangen, A., "The importance of being careful", in International Workshop on Engineering Stochastic Local Search Algorithms. Springer, 2007, str. 1–15.

[569] Alexander, C., A pattern language: towns, buildings, construction. Oxford university press, 1977.

[570] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-oriented Software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[571] Dooley, J. F., Dooley, Software Development, Design and Coding. Springer, 2017.

[572] Krawiec, K., Simons, C., Swan, J., Woodward, J., "Metaheuristic design patterns: New perspectives for larger-scale search architectures", in Handbook of Research on Emergent Applications of Optimization Algorithms. IGI Global, 2018, str. 1–36.

[573] Grady, B., James, R., Ivar, J., "The unified modeling language user guide", Reading: Addison-Wesley, 1999.

[574] GECCO '14: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. New York, NY, USA: ACM, 2014.

[575] Silva, S., (ur.), GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. New York, NY, USA: ACM, 2015.

[576] Lones, M. A., "Metaheuristics in nature-inspired algorithms", in Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, 2014, str. 1419–1422.

[577] López-Ibáñez, M., Mascia, F., Marmion, M.-É., Stützle, T., "A template for designing single-solution hybrid metaheuristics", in Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, 2014, str. 1423–1426.

[578] Kovitz, B., Swan, J., “Tagging in metaheuristics”, in Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, 2014, str. 1411–1414.

[579] Kovitz, B., Swan, J., “Structural stigmergy: A speculative pattern language for metaheuristics”, in Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, 2014, str. 1407–1410.

[580] Woodward, J., Swan, J., Martin, S., “The’composite’design pattern in metaheuristics”, in Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, 2014, str. 1439–1444.

[581] Shackelford, M. R., Simons, C. L., “Metaheuristic design pattern: interactive solution presentation.”, in GECCO (Companion). Citeseer, 2014, str. 1431–1434.

[582] Brownlee, A. E., Woodward, J. R., Swan, J., “Metaheuristic design pattern: surrogate fitness functions”, in Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. ACM, 2015, str. 1261–1264.

[583] François, J.-L., Ortiz-Servin, J. J., Martín-del Campo, C., Castillo, A., Esquivel-Estrada, J., “Comparison of metaheuristic optimization techniques for bwr fuel reloads pattern design”, Annals of Nuclear Energy, Vol. 51, 2013, str. 189–195.

[584] Hartmann, S., Kolisch, R., “Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem”, European Journal of Operational Research, Vol. 127, No. 2, 2000, str. 394–407.

[585] Antosiewicz, M., Koloch, G., Kamiński, B., “Choice of best possible metaheuristic algorithm for the travelling salesman problem with limited computational time: quality, uncertainty and speed”, Journal of Theoretical and Applied Computer Science, Vol. 7, No. 1, 2013, str. 46–55.

[586] Shaheen, S. A., Sperling, D., Wagner, C., “A short history of carsharing in the 90’s”, 1999.

[587] Slack, N., Chambers, S., Johnston, R., Operations management. Pearson Education, 2009.

[588] Ernst, A. T., Jiang, H., Krishnamoorthy, M., Sier, D., “Staff scheduling and rostering: A review of applications, methods and models”, European journal of operational research, Vol. 153, No. 1, 2004, str. 3–27.

[589] Aksin, Z., Armony, M., Mehrotra, V., “The modern call center: A multi-disciplinary perspective on operations management research”, Production and Operations Management, Vol. 16, No. 6, 2007, str. 665–688.

[590] Saltzman, R. M., “A hybrid approach to minimize the cost of staffing a call center”, International Journal of Operations and Quantitative Management, Vol. 11, No. 1, 2005.

[591] Robbins, T. R., Harrison, T. P., “A stochastic programming model for scheduling call centers with global service level agreements”, European Journal of Operational Research, Vol. 207, 2010, str. 1608–1619.

[592] EPSRC/ESRC, “Review of research status of operational research in the uk”, European Journal of Operational Research, Vol. 125, No. 2, 2004, str. 359–369.

[593] Matijaš, V. D., Molnar, G., Čupić, M., Jakobović, D., Bašić, B. D., “University course timetabling using aco: a case study on laboratory exercises”, in Knowledge-Based and Intelligent Information and Engineering Systems. Springer, 2010, str. 100–110.

[594] Lourenço, H. R., Martin, O. C., Stützle, T., Iterated local search. Springer, 2003.

[595] Hart, J. P., Shogan, A. W., “Semi-greedy heuristics: An empirical study”, Operations Research Letters, Vol. 6, No. 3, 1987, str. 107–114.

[596] de Almeida Correia, G. H., Antunes, A. P., “Optimization approach to depot location and trip selection in one-way carsharing systems”, Transportation Research Part E: Logistics and Transportation Review, Vol. 48, No. 1, 2012, str. 233–247.

[597] Shaheen, S. A., Chan, N. D., Micheaux, H., “One-way carsharing’s evolution and operator perspectives from the americas”, Transportation, Vol. 42, No. 3, 2015, str. 519–536.

[598] Namazu, M., Dowlatabadi, H., “Vehicle ownership reduction: A comparison of one-way and two-way carsharing systems”, Transport Policy, Vol. 64, 2018, str. 38–50.

[599] Litman, T., “Evaluating carsharing benefits”, Transportation Research Record, Vol. 1702, No. 1, 2000, str. 31–35.

[600] Schuster, T. D., Byrne, J., Corbett, J., Schreuder, Y., “Assessing the potential extent of carsharing: A new method and its implications”, Transportation research record, Vol. 1927, No. 1, 2005, str. 174–181.

[601] Nourinejad, M., Zhu, S., Bahrami, S., Roorda, M. J., “Vehicle relocation and staff rebalancing in one-way carsharing systems”, Transportation Research Part E: Logistics and Transportation Review, Vol. 81, 2015, str. 98–113.

[602] Kaspi, M., Raviv, T., Tzur, M., Galili, H., “Regulating vehicle sharing systems through parking reservation policies: Analysis and performance bounds”, European Journal of Operational Research, Vol. 251, No. 3, 2016, str. 969–987.

[603] Shaheen, S. A., Cohen, A. P., “Carsharing and personal vehicle services: worldwide market developments and emerging trends”, International journal of sustainable transportation, Vol. 7, No. 1, 2013, str. 5–34.

[604] Wappelhorst, S., Sauer, M., Hinkeldein, D., Bocherding, A., Glaß, T., “Potential of electric carsharing in urban and rural areas”, Transportation Research Procedia, Vol. 4, 2014, str. 374–386.

[605] Bignami, D. F., Vitale, A. C., Lué, A., Nocerino, R., Rossi, M., Savaresi, S. M., “Electric vehicle sharing services for smarter cities”, Spinger, 2017.

[606] Brandstätter, G., Gambella, C., Leitner, M., Malaguti, E., Masini, F., Puchinger, J., Ruthmair, M., Vigo, D., “Overview of optimization problems in electric car-sharing system design and management”, in Dynamic perspectives on managerial decision making. Springer, 2016, str. 441–471.

[607] Yang, J., Guo, F., Zhang, M., “Optimal planning of swapping/charging station network with customer satisfaction”, Transportation Research Part E: Logistics and Transportation Review, Vol. 103, 2017, str. 174–197.

[608] Boyacı, B., Zografos, K. G., Geroliminis, N., “An integrated optimization-simulation framework for vehicle and personnel relocations of electric carsharing systems with reservations”, Transportation Research Part B: Methodological, Vol. 95, 2017, str. 214–237.

[609] Schönbeck, C., Linßen-Robertz, A., Schwoll, M., Handbuch für AutoTeiler. StadtteilAUTO eV, 1992.

[610] Harms, S., Truffer, B., “The emergence of a nation-wide carsharing co-operative in switzerland”, A case-study for the EC-supported rsearch project “Strategic Niche Management as a tool for transition to a sustainable transport system”, EAWAG: Zürich, 1998.

[611] Shaheen, S., Sperling, D., Wagner, C., “Carsharing in europe and north america: past, present, and future”, 1998.

[612] D’WELLES, J., INGÉNIEUR, É., “A propos de circulation urbaine...”, Urbanisme Revue Francaise, Vol. 11, 1951, str. 56.

[613] “Bluecar web site”, available at: https://www.bluecar.fr/ , accessed October, 2019.

[614] “Bluecity web site”, available at: https://www.blue-city.co.uk/ , accessed October, 2019.

[615] Doherty, M., Sparrow, F., Sinha, K., “Public use of autos: mobility enterprise project”, Journal of transportation engineering, Vol. 113, No. 1, 1987, str. 84–94.

[616] Bendixson, T., Richards, M. G., “Witkar: Amsterdam’s self-drive hire city car”, Transportation, Vol. 5, No. 1, 1976, str. 63–72.

[617] IPC Building and Control Journals Limited, “What about witkar? surveyor-public authority technology”, IPC Build. Control J, 1974.

[618] Mindur, L., Sierpiński, G., Turoń, K., “Car-sharing development–current state and perspective”, Logistics and Transport, Vol. 39, 2018.

[619] Shaheen, S., Cohen, A., Jaffee, M., “Innovative mobility: Carsharing outlook”, 2018.

[620] Shaheen, S., Cohen, A., Jaffee, M., “Innovative mobility carsharing outlook-spring 2018”, University of California, Institute of Transportation Studies, 2018.

[621] Ryden, C., Morin, E., “Mobility services for urban sustainability: Environmental assessment”, Moses Report WP6, Trivector Traffic AB, Stockholm, Sweden, 2005.

[622] Martin, E. W., Shaheen, S. A., “Greenhouse gas emission impacts of carsharing in north america”, IEEE transactions on intelligent transportation systems, Vol. 12, No. 4, 2011, str. 1074–1086.

[623] Firnkorn, J., Müller, M., “What will be the environmental effects of new free-floating car-sharing systems? The case of car2go in ulm”, Ecological Economics, Vol. 70, No. 8, 2011, str. 1519–1528.

[624] Ter Schure, J., Napolitan, F., Hutchinson, R., “Cumulative impacts of carsharing and unbundled parking on vehicle ownership and mode choice”, Transportation Research Record, Vol. 2319, No. 1, 2012, str. 96–104.

[625] Sioui, L., Morency, C., Trépanier, M., “How carsharing affects the travel behavior of households: A case study of montréal, canada”, International Journal of Sustainable Transportation, Vol. 7, No. 1, 2013, str. 52–69.

[626] Shaheen, S. A., Meyn, M., Wipyewski, K., “US shared-use vehicle survey findings on carsharing and station car growth: Obstacles and opportunities”, Transportation Research Record, Vol. 1841, No. 1, 2003, str. 90–98.

[627] Shaheen, S. A., Cohen, A., “Innovative mobility carsharing outlook-carsharing market overview, analysis, and trends–fall 2012, transportation sustainability research center-university of california, berkeley”, 2014.

[628] Shaheen, S., Chan, N., Bansal, A., Cohen, A., “Definitions, industry developments, and early understanding”, Berkeley, CA: University of California Berkeley Transportation Sustainability Research Center. http://innovativemobility. org/wp-content/uploads/2015/11/SharedMobility_WhitePaper_FINAL. pdf, 2015.

[629] “Movmi carsharing market & growth analysis 2019”, available at: http://movmi.net/carsharing-market-growth-2019/ , accessed October, 2019.

[630] “Carsharing association web site”, available at: https://carsharing.org/ , accessed October, 2019.

[631] “Zipcar web site”, available at: https://www.zipcar.com/ , accessed October, 2019.

[632] “Share now web site”, available at: https://www.share-now.com/ , accessed October, 2019.

[633] “Spincity web site”, available at: https://www.spincity.hr/ , accessed October, 2019.

[634] Copeland, D. G., McKenney, J. L., “Airline reservations systems: lessons from history”, MIS quarterly, 1988, str. 353–370.

[635] Wang, H., Cheu, R. L., “Operations of a taxi fleet for advance reservations using electric vehicles and charging stations”, Transportation Research Record, Vol. 2352, No. 1, 2013, str. 1–10.

[636] Lo, H. K., An, K., Lin, W.-h., “Ferry service network design under demand uncertainty”, Transportation Research Part E: Logistics and Transportation Review, Vol. 59, 2013, str. 48–70.

[637] Lu, C.-C., Yan, S., Huang, Y.-W., “Optimal scheduling of a taxi fleet with mixed electric and gasoline vehicles to service advance reservations”, Transportation Research Part C: Emerging Technologies, Vol. 93, 2018, str. 479–500.

[638] Kaspi, M., Raviv, T., Tzur, M., “Parking reservation policies in one-way vehicle sharing systems”, Transportation Research Part B: Methodological, Vol. 62, 2014, str. 35–50.

[639] “Car2go web site”, available at: https://www.car2go.com/ , accessed October, 2019.

[640] “Zipcar flex web site”, available at: https://www.zipcar.com/en-gb/flex , accessed October, 2019.

[641] “Drivenow pricing web site”, available at: https://www.drive-now.com/de/en/pricing/ , accessed October, 2019.

[642] “Enjoy web site”, available at: https://enjoy.eni.com/en , accessed October, 2019.

[643] Jorge, D., Correia, G. H., Barnhart, C., “Comparing optimal relocation operations with simulated relocation policies in one-way carsharing systems”, IEEE Transactions on Intelligent Transportation Systems, Vol. 15, No. 4, 2014, str. 1667–1675.

[644] Weikl, S., Bogenberger, K., “Relocation strategies and algorithms for free-floating car sharing systems”, IEEE Intelligent Transportation Systems Magazine, Vol. 5, No. 4, 2013, str. 100–111.

[645] Weikl, S., Bogenberger, K., Geroliminis, N., “Simulation framework for proactive relocation strategies in free-floating carsharing systems”, Tech. Rep., 2016.

[646] Boyacı, B., Zografos, K. G., Geroliminis, N., “An optimization framework for the development of efficient one-way car-sharing systems”, European Journal of Operational Research, Vol. 240, No. 3, 2015, str. 718–733.

[647] Deng, Y., Cardin, M.-A., “Integrating operational decisions into the planning of one-way vehicle-sharing systems under uncertainty”, Transportation Research Part C: Emerging Technologies, Vol. 86, 2018, str. 407–424.

[648] Huang, K., de Almeida Correia, G. H., An, K., “Solving the station-based one-way carsharing network planning problem with relocations and non-linear demand”, Transportation Research Part C: Emerging Technologies, Vol. 90, 2018, str. 1–17.

[649] Santos, G. G. D., de Almeida Correia, G. H., “Finding the relevance of staff-based vehicle relocations in one-way carsharing systems through the use of a simulation-based optimization tool”, Journal of Intelligent Transportation Systems, 2019, str. 1–22.

[650] Di Febbraro, A., Sacco, N., Saeednia, M., “One-way carsharing: solving the relocation problem”, Transportation research record, Vol. 2319, No. 1, 2012, str. 113–120.

[651] Nourinejad, M., Roorda, M. J., “A dynamic carsharing decision support system”, Transportation research part E: logistics and transportation review, Vol. 66, 2014, str. 36–50.

[652] Kek, A. G., Cheu, R. L., Meng, Q., Fung, C. H., “A decision support system for vehicle relocation operations in carsharing systems”, Transportation Research Part E: Logistics and Transportation Review, Vol. 45, No. 1, 2009, str. 149–158.

[653] Correia, G. H. D. A., Jorge, D. R., Antunes, D. M., "The added value of accounting for users' flexibility and information on the potential of a station-based one-way carsharing system: An application in lisbon, portugal", Journal of Intelligent Transportation Systems, Vol. 18, No. 3, 2014, str. 299–308.

[654] Lourenço, H., Martin, O., Stützle, T., "A beginner's introduction to iterated local search", 06 2001, str. 1-11.

[655] Yang, H., Fung, C., Wong, K. I., Wong, S. C., "Nonlinear pricing of taxi services", Transportation Research Part A: Policy and Practice, Vol. 44, No. 5, 2010, str. 337–348.

[656] Angelopoulos, A., Gavalas, D., Konstantopoulos, C., Kypriadis, D., Pantziou, G., "Incentivized vehicle relocation in vehicle sharing systems", Transportation Research Part C: Emerging Technologies, Vol. 97, 2018, str. 175–193.

[657] Xu, M., Meng, Q., Liu, Z., "Electric vehicle fleet size and trip pricing for one-way carsharing services considering vehicle relocation and personnel assignment", Transportation Research Part B: Methodological, Vol. 111, 2018, str. 60–82.

[658] He, F., Wang, X., Lin, X., Tang, X., "Pricing and penalty/compensation strategies of a taxi-hailing platform", Transportation Research Part C: Emerging Technologies, Vol. 86, 2018, str. 263–279.

[659] Martínez, L. M., Correia, G. H. d. A., Moura, F., Mendes Lopes, M., "Insights into carsharing demand dynamics: outputs of an agent-based model application to lisbon, portugal", International Journal of Sustainable Transportation, Vol. 11, No. 2, 2017, str. 148–159.

[660] Hancock, P. J., "An empirical comparison of selection methods in evolutionary algorithms", in AISB Workshop on Evolutionary Computing. Springer, 1994, str. 80–94.

[661] Correia, G., Viegas, J. M., "Carpooling and carpool clubs: Clarifying concepts and assessing value enhancement possibilities through a stated preference web survey in lisbon, portugal", Transportation Research Part A: Policy and Practice, Vol. 45, No. 2, 2011, str. 81–90.

[662] Lopes, M. M., Martinez, L. M., de Almeida Correia, G. H., "Simulating carsharing operations through agent-based modelling: an application to the city of lisbon, portugal", Transportation Research Procedia, Vol. 3, 2014, str. 828–837.

[663] "Lisboa: o desafio da mobilidade", available at: https://fenix.tecnico.ulisboa.pt/downloadFile/3779574470659/mobilidade_Lisboa.pdf 2015.

[664] Jorge, D., Correia, G., “Carsharing systems demand estimation and defined operations: a literature review”, European Journal of Transport and Infrastructure Research, Vol. 13, No. 3, 2013.

[665] “Interfile”, available at: https://excel.interfile.de/Autokostenrechner/autokostenrechner.html , accessed October 2019.

[666] “Emel lisboa (in portuguese)”, available at: http://www.emel.pt/pt/onde-estacionar/via-publica/tarifarios 2018.

[667] “Car2go web site”, available at: https://www.car2go.com/ , accessed 2017.

[668] “Numbeo”, available at: https://www.numbeo.com/taxi-fare/in/Lisbon , accessed October 2019.

[669] Correia, G., Carpooling and carpool clubs: Clarifying concepts and assessing value enhancement possibilities. PhD in Transportation, Department of Civil Engineering, Instituto Superior . . . , 2009.

[670] Klintman, M., “Between the private and the public: Formal carsharing as part of a sustainable traffic system. an exploratory study”, Tech. Rep., 1998.

[671] Celsor, C., Millard-Ball, A., “Where does carsharing work? using geographic information systems to assess market potential”, Transportation Research Record, Vol. 1992, No. 1, 2007, str. 61–69.

[672] Rotaris, L., Danielis, R., “The role for carsharing in medium to small-sized towns and in less-densely populated rural areas”, Transportation Research Part A: Policy and Practice, Vol. 115, 2018, str. 49–62.

[673] Comission, E., “Car-sharing in small cities, fact sheet.”, More Options for Energy Efficient Mobility through Car-Sharing, MOMO project, 2004.

[674] Barth, M., Todd, M., “Simulation model performance analysis of a multiple station shared vehicle system”, Transportation Research Part C: Emerging Technologies, Vol. 7, No. 4, 1999, str. 237–259.

[675] Barth, M., Han, J., Todd, M., “Performance evaluation of a multi-station shared vehicle system”, in ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 01TH8585). IEEE, 2001, str. 1218–1223.

[676] Kek, A. G., Cheu, R. L., Chor, M. L., “Relocation simulation model for multiple-station shared-use vehicle systems”, Transportation research record, Vol. 1986, No. 1, 2006, str. 81–88.

[677] Nair, R., Miller-Hooks, E., “Fleet management for vehicle sharing operations”, Transportation Science, Vol. 45, No. 4, 2011, str. 524–540.

[678] Fan, W., Machemehl, R. B., Lownes, N. E., “Carsharing: Dynamic decision-making problem for vehicle allocation”, Transportation Research Record, Vol. 2063, No. 1, 2008, str. 97–104.

[679] Barth, M., Todd, M., Xue, L., “User-based vehicle relocation techniques for multiple-station shared-use vehicle systems”, 2004.

[680] Uesugi, K., Mukai, N., Watanabe, T., “Optimization of vehicle assignment for car sharing system”, in International Conference on Knowledge-Based and Intelligent Information and Engineering Systems. Springer, 2007, str. 1105–1111.

[681] Pfrommer, J., Warrington, J., Schildbach, G., Morari, M., “Dynamic vehicle redistribution and online price incentives in shared mobility systems”, IEEE Transactions on Intelligent Transportation Systems, Vol. 15, No. 4, 2014, str. 1567–1578.

[682] Mitchell, W. J., Borroni-Bird, C. E., Burns, L. D., Reinventing the automobile: Personal urban mobility for the 21st century. MIT press, 2010.

[683] Zhou, S., “Dynamic incentive scheme for rental vehicle fleet management”, Doktorski rad, Massachusetts Institute of Technology, 2012.

[684] Wie, B.-W., Tobin, R. L., “Dynamic congestion pricing models for general traffic networks”, Transportation Research Part B: Methodological, Vol. 32, No. 5, 1998, str. 313–327.

[685] Joksimovic, D., Bliemer, M., Bovy, P., “Dynamic optimal toll design problem–travel behavior analysis including departure time choice and heterogeneous users”, in 11th International Conference on Tavle Behaviour Reserch, Kyoto, Japan, 2006.

[686] Wang, J. Y., Lindsey, R., Yang, H., “Nonlinear pricing on private roads with congestion and toll collection costs”, Transportation Research Part B: Methodological, Vol. 45, No. 1, 2011, str. 9–40.

[687] Do Chung, B., Yao, T., Friesz, T. L., Liu, H., “Dynamic congestion pricing with demand uncertainty: A robust optimization approach”, Transportation Research Part B: Methodological, Vol. 46, No. 10, 2012, str. 1504–1518.

[688] Lou, Y., Yin, Y., Laval, J. A., “Optimal dynamic pricing strategies for high-occupancy/toll lanes”, Transportation Research Part C: Emerging Technologies, Vol. 19, No. 1, 2011, str. 64–74.

[689] Schön, C., “Integrated airline schedule design, fleet assignment and pricing”, DSOR-Beiträge zur Wirtschaftsinformatik, DSOR Contributions to Information Systems, Vol. 5, 2008, str. 73–88.

[690] Atasoy, B., Salani, M., Bierlaire, M., “An integrated airline scheduling, fleeting, and pricing model for a monopolized market”, Computer-Aided Civil and Infrastructure Engineering, Vol. 29, No. 2, 2014, str. 76–90.

[691] de Almeida Correia, G. H., Antunes, A. P., “Optimization approach to depot location and trip selection in one-way carsharing systems”, Transportation Research Part E: Logistics and Transportation Review, Vol. 48, No. 1, 2012, str. 233–247.

[692] Jorge, D., Barnhart, C., de Almeida Correia, G. H., “Assessing the viability of enabling a round-trip carsharing system to accept one-way trips: Application to logan airport in boston”, Transportation Research Part C: Emerging Technologies, Vol. 56, 2015, str. 359–372.

[693] Bordley, R., “An overlapping choice set model of automotive price elasticities”, Transportation Research Part B: Methodological, Vol. 28, No. 6, 1994, str. 401–408.

[694] Taplin, J. H., Hensher, D. A., Smith, B., “Preserving the symmetry of estimated commuter travel elasticities”, Transportation Research Part B: Methodological, Vol. 33, No. 3, 1999, str. 215–232.

[695] Fouquet, R., “Trends in income and price elasticities of transport demand (1850–2010)”, Energy Policy, Vol. 50, 2012, str. 62–71.

[696] de Dios Ortuzar, J., Willumsen, L. G., Modelling transport. John wiley & sons, 2011.

[697] Sinha, K. C., Labi, S., Transportation decision making: Principles of project evaluation and programming. John Wiley & Sons, 2011.

[698] Bussieck, M. R., Vigerske, S., “MINLP solver software”, Wiley encyclopedia of operations research and management science, 2010.

[699] Powell, M. J., “An efficient method for finding the minimum of a function of several variables without calculating derivatives”, The computer journal, Vol. 7, No. 2, 1964, str. 155–162.

[700] Powell, M. J. D., “Restart procedures for the conjugate gradient method”, Mathematical programming, Vol. 12, No. 1, 1977, str. 241–254.

List of Figures

Biography

Goran Molnar was born in 1985 in Bjelovar, where he finished his elementary and secondary education. He obtained his MSc degree in computer science from the Faculty of Electrical Engineering and Computing of the University of Zagreb in 2011 by defending the master thesis 'Parallel Ant Colony Optimisation for Laboratory Exercise Timetabling Problem' under the supervision of prof. dr. Bojana Dalbelo Bašić. As an MSc student, in May 2009, he was awarded Best student paper award within the proceedings of MIPRO conference, and in November 2009 he was awarded the Prize for Best Student Computer Program by the Faculty of Electrical Engineering and Computing. In 2012, he started his PhD studies at the Faculty of Electrical Engineering and Computing under the supervision of prof. dr. Domagoj Jakobović, and defended his thesis topic titled 'Metaheuristics for Problems with Limited Budget of Evaluations'.

Corollary to his doctoral education, he acquired work experience in both private and public sector. Between May 2011 and May 2013 he was an employee of Asseco SEE, based in Zagreb, where he acquired his first professional experiences as a junior developer for workforce scheduling systems. His next employment was in Agencija za komercijalnu djelatnost Ltd. from May 2013 until April 2014, where he worked as Smart Card Application developer. Between April 2014 and September 2015 he was based in Portugal, where he worked as a researcher at the Faculty of Science and Technology of the University of Coimbra, cooperating as algorithm expert with the InnoVshare Project, coordinated by prof. dr. Gonçalo Homem de Almeida Correia. After his return from Portugal, from September 2015 onwards, he works as an independent consultant specialized in scheduling and transport optimisation with OptaPlanner, cooperating, among others, with companies based in the United States, Germany and Norway.

Publications

1. Jorge, D., Molnar, G., & de Almeida Correia, G. H. (2015). Trip pricing of one-way station-based carsharing networks with zone and time of day price variations. Transportation Research Part B: Methodological, 81, 461-482.
2. Molnar, G., & de Almeida Correia, G. H. (2019). Long-term vehicle reservations in one-way free-floating carsharing systems: A variable quality of service model. Transportation

Research Part C: Emerging Technologies, 98, 298-322.

3. Molnar, G., Jakobović, D., & Pavelić, M. (2016, March). Workforce Scheduling in Inbound Customer Call Centres with a Case Study. In European Conference on the Applications of Evolutionary Computation (pp. 831-846). Springer, Cham.

Životopis

Goran Molnar rođen je 1985. godine u Bjelovaru, gdje je završio osnovno i srednje obrazovanje. Diplomski studij računarstva završio je na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu 2011. godine obranom rada 'Paralelni algoritam mravlje kolonije za izradu rasporeda laboratorijskih vježbi' pod mentorstvom prof. dr. sc. Bojane Dalbelo Bašić. Kao diplomski student računarstva, u svibnju 2009. osvojio je nagradu za najbolji studentski rad u sklopu Međunarodnog skupa za informacijsku komunikacijsku i elektroničku tehnologiju (MIPRO), a iste je godine u studenom osvojio i nagradu Fakulteta elektrotehnike i računarstva za najbolji studentski računalni program. Doktorski studij računarstva upisao je 2012. godine, gdje 2016. godine pod mentorstvom prof. dr. sc. Domagoja Jakobovića brani temu doktorske disertacije pod naslovom 'Metaheuristike za probleme s ograničenim brojem evaluacija'.

Uz doktorsko školovanje, radno iskustvo stjecao je u privatnom i javnom sektoru. Od svibnja 2011. do svibnja 2013. zaposlen je u tvrtki Asseco SEE u Zagrebu, gdje stječe prva profesionalna iskustva u radu na sustavima izrade rasporeda. Potom se od svibnja 2013. do travnja 2014. godine zapošljava u Agenciji za komercijalnu djelatnost, gdje radi kao razvojni inženjer na aplikacijama za pametne kartice. Nakon toga, između travnja 2014. i rujna 2015. seli u Portugal te se zapošljava kao istraživač pri Fakultetu znanosti i tehnologije Sveučilišta u Coimbri, gdje kao stručnjak za algoritme surađuje na projektu InnoVshare pod vodstvom prof. dr. sc. Gonçala Homema de Almeida Correie. Po povratku iz Portugala, od rujna 2015. do danas radi kao nezavisni konzultant, specijaliziran za primjenu optimizacije rasporeda i transporta u OptaPlanneru, pri čemu sklapa suradnje s nizom tvrtki iz Sjedinjenih Američkih Država, Njemačke i Norveške.