Rješavanje problema raspoređivanja medicinskih sestara u smjene korištenjem algoritama evolucijskog računanja

Jurič, Katarina

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 451

RJEŠAVANJE PROBLEMA RASPOREĐIVANJA MEDICINSKIH SESTARA U SMJENE KORIŠTENJEM ALGORITAMA EVOLUCIJSKOG RAČUNANJA

Katarina Jurič

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 451

RJEŠAVANJE PROBLEMA RASPOREĐIVANJA MEDICINSKIH SESTARA U SMJENE KORIŠTENJEM ALGORITAMA EVOLUCIJSKOG RAČUNANJA

Katarina Jurič

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 451

Pristupnica:	**Katarina Jurič (0036522259)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Rješavanje problema raspoređivanja medicinskih sestara u smjene korištenjem algoritama evolucijskog računanja**

Opis zadatka:

Problem raspoređivanja medicinskih sestara u smjene složen je optimizacijski problem koji ima tvrda i meka ograničenja, a čije učinkovito rješenje može imati značajnu praktičnu primjenu. Problem razmatra skupinu medicinskih sestara koje trebaju biti dodijeljene jednoj od mogućih smjena, a sve u cilju ispunjenja minimalnog ograničenja pokrivenosti smjene te drugih ograničenja specifičnih za pojedini slučaj. Cilj je postići najbolju kvalitetu dodijeljenih radnih smjena. U ovom radu potrebno je razmotriti skup podataka iz knjižnice NSPLib, iskoristiti optimizacijske postupke evolucijskog računanja pri čemu je potrebno varirati broj medicinskih sestara kao i hiperparametre pojedinih optimizacijskih postupaka, te prikazati i usporediti rezultate provedenih postupaka. Potrebno je razmotriti najmanje sljedeće algoritme: genetski algoritam, genetski algoritam bez elitizma, optimizaciju rojem čestica i lokalnu pretragu. Postupak treba vizualizirati po koracima tako da se omogući jednostavnije praćenje načina na koji pojedini optimizacijski postupak djeluje na rješenje ovog problema. Za rješavanje ovog problema potrebno je iskoristiti neki postojeći radni okvir ili knjižnicu za algoritme evolucijskog računanja (npr. DEAP).

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod

Planiranje poslovnih rasporeda još je ne tako davno bio zahtjevan zadatak koji se odrađivao ručno, korištenjem papira i olovke. Odjel za ljudske resurse bi na papiru raspoređivao svakog pojedinog zaposlenika i za svakog od njih računao zadovoljava li potrebne uvjete. To ne predstavlja velik problem ako je u pitanju izrada rasporeda za mali broj zaposlenika, no ako se radi o realnijim uvjetima i tvrtkama ili bolnicama koje zapošljavaju više stotina ljudi tada izrada takvog rasporeda zahtijeva velike napore. Pokušaji automatizacije procesa unošenja podataka o rasporedu uključivali su rane verzija alata kao što je *Microsoft Office Excel*. Međutim to je samo pomoglo u analizi radnih vremena zaposlenika, ali ne i u izradi rasporeda [1].

Rani pokušaji rješavanja optimizacijskih problema pomoću računala javljaju se 1950-ih kada su računala postala dostupnija, a njihova procesorska snaga viša. Prvi poznati problem implementiran na računalu bio je popularni algoritam Simplex koji je 1947. razvio George B. Dantzig. Samo par godina kasnije, točnije 1951., algoritam je bio implementiran na SEAC-u – jednom od prvih digitalnih računala [2]. U istom razdoblju javljaju se razne inačice problema raspoređivanja zaposlenika u smjene.

Sukladno pojavi optimizacijskih problema javljaju se i razni načini njihovog rješavanja, a među njima su i metaheuristike. One služe za rješavanje složenih optimizacijskih problema kod kojih nije važno naći savršeno optimalno rješenje nego ono koje je dovoljno dobro. Prva predstavljena metaheuristika javila se također sredinom 20-tog stoljeća, a to je bio algoritam simuliranog kaljenja (engl. *simmulated annealing*). Ubrzo se javlja i popularni genetski algoritam kojeg je javnosti objavio John Holland, a slijedi ga Fred Glover s tabu-pretragom, Marco Dorigo s mravljim algoritmom te James Kennedy i Russel Eberhart s algoritmom roja čestica [5].

Metaheuristike su zbog svoje prilagodljivosti pogodne za primjenu u raznim područjima. Često se koriste u području strojnog učenja za ugađanje hiperparametara i u odabiru značajki, a u bioinformatici imaju ulogu u pronalasku odnosa između sekvenci aminokiselina. U području logistike, gdje spada i problem raspoređivanja medicinskih sestara u smjene, metaheuristike se koriste za izradu rasporeda, optimizacije opskrbnih lanaca i pronalaska optimalnih ruta vozila.

1. Problem raspoređivanja medicinskih sestara u smjene

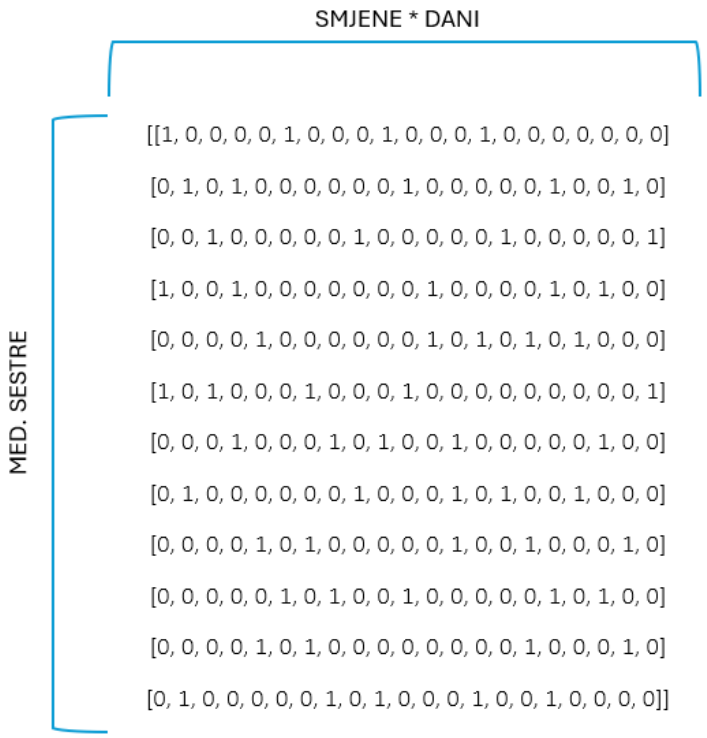
Problem raspoređivanja medicinskih sestara u smjene (engl. *NSP – Nurse Scheduling Problem/ NRP – Nurse Rostering Problem*) spada u grupu kombinatornih optimizacijskih problema. Takvi problemi pripadaju grupi optimizacijskih problema koji imaju diskretne varijable odluke, a rješenje im se nalazi u konačnom prostoru. Vjerojatno najpoznatiji optimizacijski problem koji pripada istoj grupi je problem trgovačkog putnika (*engl. TSP – Traveling Salesman Problem*) gdje je cilj pronaći najkraću moguću rutu između gradova tako da trgovački putnik posjeti svaki grad samo jednom. Među poznatijima iz iste grupe problema nalazi se i problem ruksaka (*engl. KP – Knapsack Problem*) gdje je u cilju ispuniti ruksak predmetima najveće vrijednosti pazeći pritom na težinu svakog predmeta i ukupnu nosivost ruksaka.

Problem raspoređivanja medicinskih sestara u smjene u svom sadašnjem obliku objavljen je 1976. godine u dva različita rada: *Nurse Scheduling Using Mathematical Programming* [3] i *Scheduling Nursing Personnel According to Nursing Preference: A Mathematical Programming Approach* [4].

NSP je NP-težak problem. NP-teški problemi predstavljaju klasu problema koja opisuje složenost problema odlučivanja. NP-teški problemi smatraju se među najtežim problemima s obzirom na to da su barem toliko teški koliko i najteži problemi u klasi NP (engl. *Nondeterministic Polynomial time*). NP-problemi predstavljaju drugu srodnu klasu koja također opisuje složenost problema, a njihovo se rješenje može provjeriti u polinomijalnom vremenu, dakle brzo. Isto vrijedi i za NP-težak problem raspoređivanja medicinskih sestara u smjene. S druge strane, optimalno rješenje čitavog problema ne može se naći u polinomijalnom vremenu, ali za neki dobiveni raspored koji predstavlja potencijalno rješenje može se brzo provjeriti zadovoljava li uvjete problema.

Sva rješenja NSP-a nalaze se u diskretnom binarnom prostoru. Na Slika 1.1 prikazan je primjer rješenja NSP-a koje predstavlja raspored smjena medicinskih sestara. Svaki red matrice je tjedni raspored jedne od med. sestara, a unutar tog reda za svaki dan u tjednu nalaze se, u ovom slučaju, tri smjene. Prostor rješenja je diskretan i binaran budući da svaka medicinska sestra za svaki dan u tjednu može dobiti smjenu (1) ili ne (0). Raspored na slici ima 2^{252} mogućih rješenja budući da se raspored izrađuje za 12 medicinskih sestara i za tri smjene dnevno za svaki dan u tjednu. Ako bi se u raspored dodale samo tri medicinske sestre više, tada bi broj mogućih

rješenja bio 2^{315}. Taj je broj otprilike jednak $6{,}67*10^{94}$ i za nekoliko je redova veličine veći od broja atoma u svemiru [6]. Jasno je da onda detaljno pretraživanje cijelog prostora rješenja ne dolazi u obzir kao jedna od metoda rješavanja NSP-a, pa čak ni ako se pritom koriste kvantna računala.



Slika 1.1 Primjer rješenja NSP problema

1.1. NSPLib

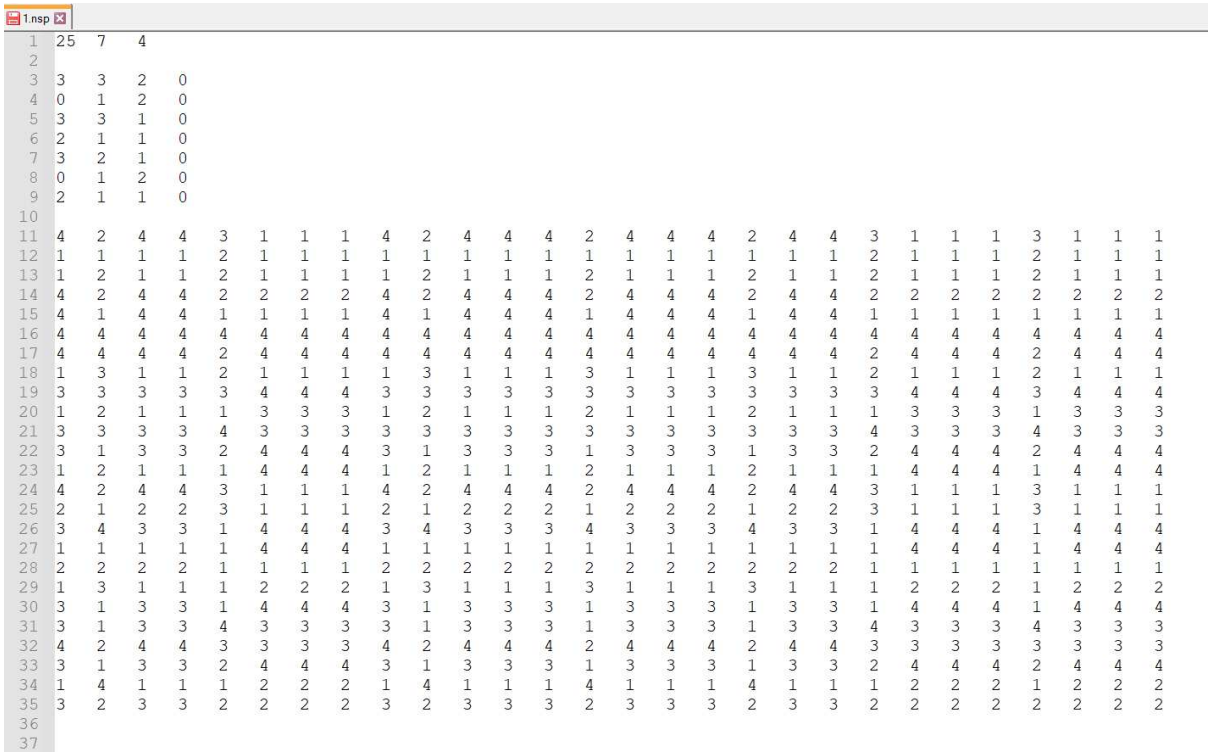
Za rješavanje problema raspoređivanja medicinskih sestara u smjene korištena je baza podataka *NSPLib–A nurse scheduling problem library: a tool to evaluate (meta-) heuristic procedures* koju su objavili dvojica istraživača sa Sveučilišta u Ghentu: Mario Vanhoucke i Broos Maenhout [7].

NSPLib sadrži velik broj primjera problema različitih složenosti. Svaki primjer iz baze podataka koji se koristi kao ulaz u algoritme mora sadržavati sljedeće informacije:

- ograničenja pokrivenosti,
- matricu preferencija,
- specifična ograničenja za određeni slučaj.

Ograničenja pokrivenosti (engl. *coverage constraints*) odnose se na pokrivanje minimalnog broja medicinskih sestara po smjeni. Za svaki dan i za svaku pripadnu smjenu postoji minimalan broj medicinskih sestara koje tada moraju raditi.

Matrica preferencija (engl. *preference matrix*) predstavlja jačinu sklonosti svake pojedine medicinske sestre prema radu u određenoj smjeni i to za svaki dan u rasporedu.



Slika 1.2 - Primjer ulazne datoteke iz NSPLiba

Na Slika 1.2 prikazana je jedna od ulaznih datoteka koja sadrži ograničenja pokrivenosti i matricu preferencija. Svaka datoteka u prvom retku ima redom nabrojane vrijednosti:

- broj medicinskih sestara,
- broj dana za koje se izrađuje raspored i
- broj smjena u jednom danu.

U primjeru na Slika 1.2 radi se o tjednom rasporedu za 25 medicinskih sestara gdje svaki dan ima četiri smjene (jutarnju, dnevnu, noćnu i slobodnu).

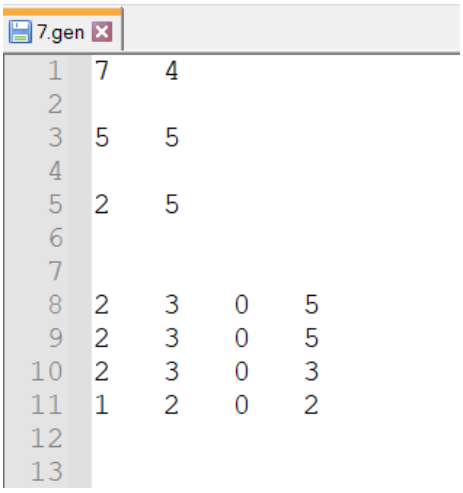
Redci 3-9 predstavljaju ograničenja pokrivenosti gdje je svaki redak jedan dan u tjednu. Npr. u ponedjeljak u jutarnjoj smjeni moraju biti minimalno tri medicinske sestre, u dnevnoj smjeni također minimalno tri, a u noćnoj smjeni minimalno dvije medicinske sestre. Za slobodnu smjenu nikad nema minimalnih ograničenja.

Redci 11-35 predstavljaju matricu preferencija. Vrijednosti od jedan do četiri pokazuju koliko je jaka sklonost medicinske sestre prema radu u toj smjeni na određeni dan. Može se gledati i na suprotan način, tada vrijednosti umjesto jačine sklonosti pokazuju koliko je jaka odbojnost prema radu u određenom razdoblju. Za rješenje problema svejedno je koji se način koristi, a u ovom radu uzeto je da vrijednosti predstavljaju sklonost prema radu.

Važno je reći da su ograničenja pokrivenosti zapravo tvrda ograničenja problema (engl. *hard constraints*), a preferencije i pripadna matrica preferencija su meka ograničenja (engl. *soft constraints*). Tvrda ograničenja problema su oni uvjeti koje rješenje mora zadovoljiti, a ako ih potencijalno rješenje ne zadovoljava tada ono nije valjano. Meka ograničenja rješenje ne mora zadovoljiti, ali ona pomažu razlikovati bolja od lošijih rješenja. Ako neka dva rješenja oba zadovoljavaju tvrda ograničenja, a jedno od njih zadovoljava više mekih ograničenja tada je ono bolje rješenje. U slučaju NSP-a tvrda ograničenja podrazumijevaju zadovoljavanje minimalnog potrebnog broja medicinskih sestara po smjenama i zadovoljavanje uvjeta specifičnih za neki slučaj – npr. broj maksimalnih uzastopnih smjena, a ispunjavanje što više mekih ograničenja rezultira ispunjavanjem osobnih afiniteta medicinskih sestara.

Tvrda i meka ograničenja nisu koncept specifičan samo za NSP. Koriste se i u mnogim drugim optimizacijskim problemima. Za već spomenuti problem trgovačkog putnika tvrdo ograničenje znači da trgovac kroz svaki grad smije proći samo jednom te se mora vratiti u grad iz kojeg je krenuo. Meko ograničenje podrazumijeva minimiziranje ukupne dužine puta. Slično, za problem ruksaka tvrdo ograničenje je ukupna nosivost ruksaka, a meko ograničenje uključuje maksimizaciju ukupne vrijednosti predmeta u ruksaku.

Osim ograničenja pokrivenosti i matrice preferencija, NSPLib sadrži i ograničenja specifična za određene slučajeve. Jedna od datoteka koja sadrži takva ograničenja prikazana je na Slika 1.3.



Slika 1.3 – Ograničenja specifičnih slučajeva

Takva ograničenja ubrajaju se u tvrda ograničenja. Takva datoteka sadrži informacije koje se odnose na svaku medicinsku sestru tako da primjer ne sadrži informaciju o ukupnom broju medicinskih sestara. U prvom retku nalazi se informacija o broju dana i broju smjena u svakom danu. Treći i petak redak odnose se redom na minimalni i maksimalni broj ukupno dodijeljenih smjena te na minimalni i maksimalni broj uzastopnih smjena u razdoblju na koje se raspored odnosi. Zadnja četiri reda odnose se na svaku vrstu smjene zasebno. Prve dvije vrijednosti u tim redcima odnose se na minimalni i maksimalni broj dodijeljenih takvih smjena zaredom, a zadnje dvije vrijednosti na minimalni i maksimalni broj ukupno dodijeljenih takvih smjena. Dakle, redak 8 govori da za svaku medicinsku sestru broj uzastopno dodijeljenih jutarnjih smjena mora biti u intervalu [2, 3], a da broj ukupno dodijeljenih jutarnjih smjena mora biti u intervalu [0, 5].

Cijela baza podataka NSPLib je podijeljena u dva glavna skupa – raznoliki (engl. *diverse*) i realistični (engl. *realistic*). Raznoliki skup ima 4 podskupa, svaki za 7 dana u tjednu, ali za različit broj medicinskih sestara koji može biti 25, 50, 75 ili 100. Svaki podskup ima 7290 instanci. Realistični skup je generiran za 28 dana u mjesecu i ima dva podskupa, jedan za 30 medicinskih sestara, a drugi za 60 medicinskih sestara. Svaki od tih podskupova ima 960 instanci. Ukupna baza podataka, uključujući ograničenja za specifične slučajeve, ima 31.096 instanci.

2. Metaheuristike i evolucijsko računanje

Kao što je ranije spomenuto, postoje problemi koji su preteški za iscrpno pretraživanje niti za njih postoje poznati efektivni načini pronalaska točnog rješenja. Međutim, odlična stvar je da kod takvih problema u praksi najčešće ne tražimo najbolje moguće rješenje nego ono koje je dovoljno dobro. U tom slučaju postoje razni efikasni algoritmi koji dolaze do dobrih rješenja.

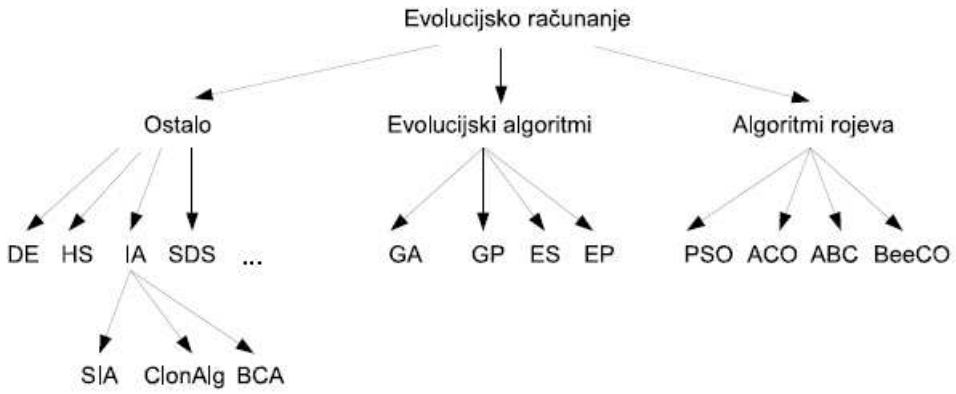
Algoritme koji pronalaze rješenja koja su zadovoljavajuće dobra, ali ne nude nikakve garancije da će uspjeti pronaći optimalno rješenje, te koji imaju relativno nisku računsku složenost (tipično polinomijalnu složenost) nazivaju se približni algoritmi, heurističke metode, heuristički algoritmi ili jednostavno heuristike. Heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja zove se metaheuristika. Simulirano kaljenje, genetski algoritam, optimizacija rojem čestica, tabu-pretraga, algoritam kolonije mrava – svi ovi algoritmi spadaju pod metaheuristike [9].

Metaheuristike se mogu podijeliti u dvije velike porodice algoritama: algoritmi koji rade nad jednim rješenjem (kao što je tabu-pretraga) i populacijski algoritmi koji rade sa skupovima rješenja (npr. genetski algoritam ili algoritam roja čestica) [8].

Jedna od podgrupa populacijskih algoritama je evolucijsko računanje. Ono se dalje dijeli na svoje podgrupe kao što je prikazano na Slika 2.1, a one su:

- evolucijski algoritmi (npr. genetski algoritam),
- algoritmi rojeva (npr. mravlji algoritam i algoritam optimizacije rojem čestica) i
- ostali algoritmi (npr. algoritam umjetnog imunološkog sustava).

Kao što se da primijetiti i iz naziva algoritama, većina ih crpi inspiraciju iz prirode. Nove metaheuristike se i dalje razvijaju, pa tako danas postoji algoritam optimizacije leteće lisice (engl. *Flying Fox Optimization Algorithm* [10]), algoritam crvenog jelena (engl. *Red Deer Algorithm* [11]) i optimizacija carskog pingvina (engl. *Emperor Penguin Optimizer* [12]). Većina najnovijih takvih algoritama je inačica nekih od prethodnih.



Slika 2.1 – Podjela evolucijskog računanja [7]

2.1. Genetski algoritam

Genetski algoritam pripada porodici evolucijskih algoritama. Osim njega u istoj porodici nalaze se i genetsko programiranje, evolucijske strategije i evolucijsko programiranje. Svima im je zajednička ideja da rade s populacijom rješenja koja kroz iteracije postaje sve bolja. U svakoj iteraciji na populaciju se primjenjuju evolucijski operatori koje čine:

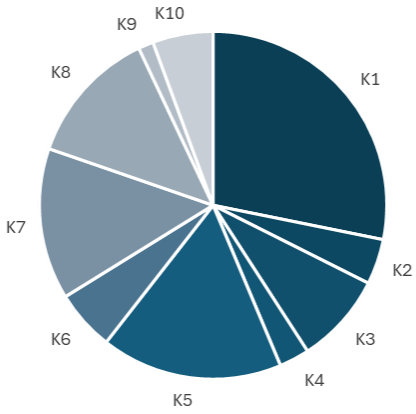
- selekcija,
- križanje,
- mutacija i
- zamjena.

Glavna zadaća selekcije je osigurati da se u odabiru jedinki za križanje biraju one najbolje koje će dati dobre genetske materijale za sljedeću populaciju. Jedinke koje se odabiru za križanje zovu se roditelji. Selekcija mora birati roditelje proporcionalno njihovoj dobroti (engl. *fitness value*). Dobrota je atribut koji opisuje koliko je rješenje u skladu s funkcijom dobrote (engl. *fitness function*). U slučaju da se u problemu želi nešto maksimizirati, npr. vrijednost predmeta u problemu naprtnjače, tada će veću dobrotu imati rješenje koje ima veću vrijednost naprtnjače. Suprotno, ako se u problemu nešto minimizira, npr. ukupna udaljenost u problemu trgovačkog putnika, tada će veću dobrotu imati rješenja s kraćim ukupnim putem.

Iako je zadaća selekcije birati roditelje s visokom dobrotom kako bi se u sljedeću generaciju prenosili dobri geni, važno je ne uvijek odabirati samo najbolje roditelje jer će takva selekcija rezultirati zapinjanjem rješenja u lokalnom optimumu. Dvije popularne selekcije koje imaju to

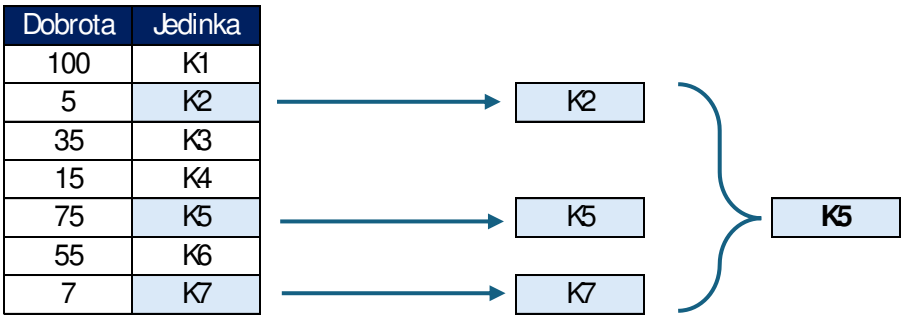
na umu su proporcionalna selekcija (engl. *fitness proportionate selection*), poznatija pod nazivom engl. *roulette-wheel selection,* i turnirska selekcija (engl. *tournament selection*).

Proporcionalna selekcija dobila je svoj sinonim rulet-selekcije po tome što se njen način rada lako može vizualizirati kao kotač ruleta gdje svaka jedinka zauzima onoliko prostora koliko je proporcionalno njezinoj dobroti. Na Slika 2.2 prikazana je skica proporcionalne selekcije u slučaju kada populacija ima deset jedinki. Jedinke K1 i K5 imaju najveću dobrotu pa s time i najveću vjerojatnost da budu izabrane za roditelje, no to ne znači da jedinka kao što je K9 ne može biti izabrana.



Slika 2.2: Proporcionalna selekcija

Druga popularna vrste selekcije je turnirska selekcija. U njoj je slučajnost implementirana tako da se nasumično odabire broj jedinki definiran veličinom turnira i iz njega se bira najbolja jedinka kao sljedeći roditelj. Na Slika 2.3 prikazan je primjer turnirske selekcije s veličinom turnira tri. Nasumično su odabrane jedinke K2, K5 i K7, a među njima je najbolja K5 tako da ona postaje roditelj.

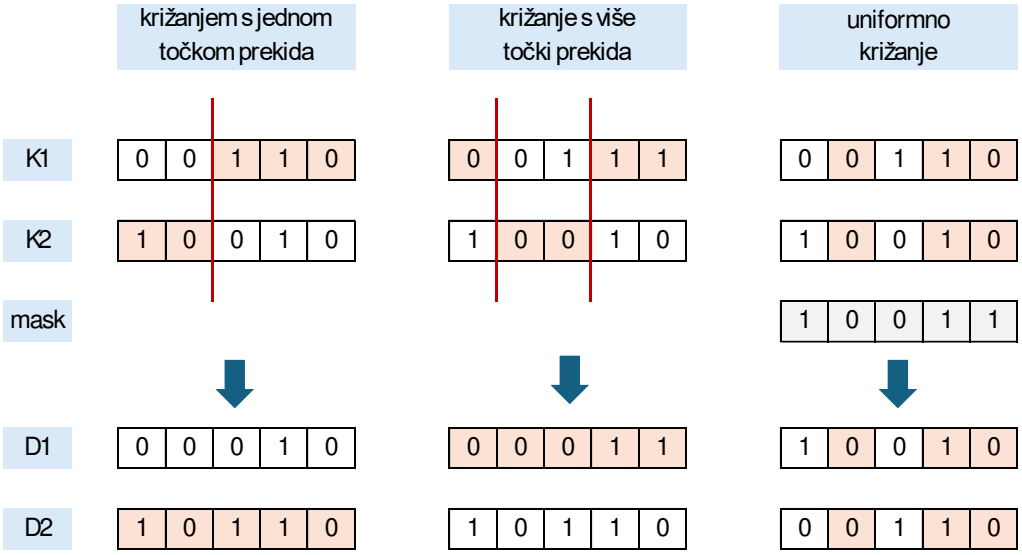


Slika 2.3: Turnirska selekcija

Nakon što su selekcijom odabrani roditelji slijedi sljedeći korak – križanje (engl. *crossover*). Križanje binarnih jedinki može se izvesti uniformno ili križanjem s jednom ili više točki prekida. Na Slika 2.4 skicirane su sve tri vrste križanja. Kod križanja s jednom točkom prekida odabire se jedna točka prekida na istoj poziciji kod oba roditelja, a rezultat su dvije nove jedinke – dva djeteta gdje svaki nosi po jedan dio genetskog materijala od oba roditelja. Križanje s više točaka prekida funkcionira na sličan način i također rezultira s dvije nove jedinke gdje se genetski dijelovi roditelja prenose u *cik-cak* segmentima. Uniformno križanje nešto je drugačije i ono se računa po izrazima (1) i (2) gdje je R slučajno generirani niz bitova – maska, iste duljine kao i roditelj. K1 i K2 su roditelji, a D1 i D2 djeca [8].

$$D1 = K1\ and\ K2\ \ or\ R\ and\ (K1\ xor\ K2) \quad (1)$$

$$D2 = K1\ and\ K2\ \ or\ not(R)\ and\ (K1\ xor\ K2) \quad (2)$$



Slika 2.4: Vrste križanja

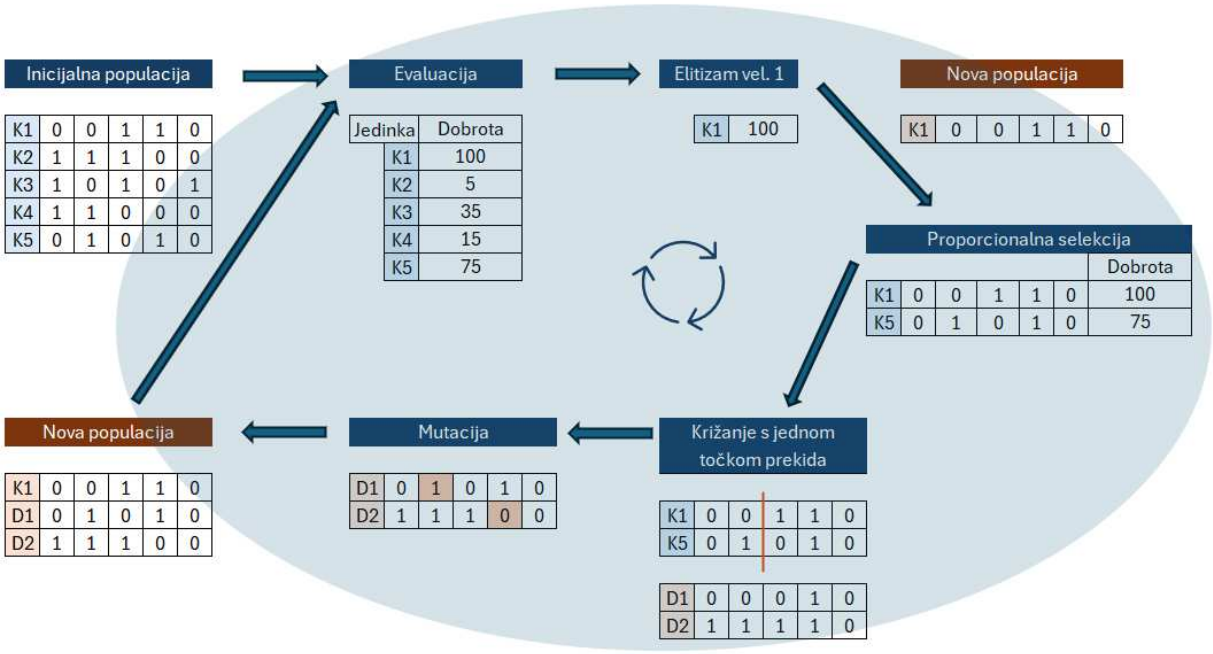
Nakon što se križanjem dobiju nove jedinke, svaka od njih prije nego uđe u novu populaciju prolazi još jedan važan korak, a to je mutacija. Mutacija osigurava raznolikost populacije i pomaže pri izbjegavanju lokalnog optimuma budući da unosi slučajnost u genom populacije. Za binarno rješenje to bi značilo da se kroz svaki bit jedinke prolazi zasebno i on se mijenja s određenom vjerojatnošću mutacije koja se odredi na početku izvođenja algoritma.

Operacija koja se često koristi u genetskom algoritmu kako bi se sačuvala najbolja rješenja i spriječilo gubljenje najboljih gena je elitizam. On se jednostavno implementira na način da se u sljedeću generaciju prenosi *N* najboljih rješenja gdje je *N* broj elitističkih jedinki (engl. *hall of fame size*).

U kodu 2.1 prikazan je pseudokod genetskog algoritma s elitizmom, a odmah ispod njega na Slika 2.5: Iteracija genetskog algoritma prikazana je skica jedne iteracije algoritma.

```
generacija = 0;
inicijaliziraj populaciju;
dok generacija < max_generacija:
      evaluiraj populaciju;
      za i od 1 do vel_elitizam:
            izaberi najbolje jedinke;
      kraj za;
      za k od vel_elitizam do vel_populacije:
            selektiraj roditelje;
            križaj roditelje -> dijete;
            mutiraj dijete -> nova jedinka;
            dodaj jedinku u novu populaciju;
      kraj za;
      evaluiraj novu populaciju;
      generacija ++;
kraj dok;
```

Kôd 2.1 – Pseudokod genetskog algoritma [8]



Slika 2.5: Iteracija genetskog algoritma

2.2. Algoritam roja čestica

Motivacija iza algoritma roja čestica (*engl. Particle Swarm Optimization – PSO*), koji pripada populacijskim algoritmima, nalazi se u kretanjima životinja kao što je roj čestica ili jato ptica. Svaka čestica u roju, tj. jedinka u populaciji, ima pridružen trenutni položaj, brzinu i funkciju dobrote, a uz to pamti i svoj najbolji položaj i najbolji položaj u cijeloj populaciji.

Na početku se svakoj jedinki dodijeli nasumični položaj u prostoru i nasumična brzina koja određuje smjer kretanja čestice. U svakoj sljedećoj iteraciji se svaka jedinka evaluira i uspoređuje svoju trenutnu dobrotu s dobrotom svog najboljeg zapamćenog položaja i najboljeg globalnog položaja. Najbolji osobni i globalni položaj se zatim po potrebi ažuriraju. Uz položaje se pamte i vrijednosti funkcije dobrote kako bi se rješenja mogla uspoređivati.

Nakon ažuriranja osobnih i globalnih optimuma svaka čestica ažurira brzinu, a zatim položaj. Brzina se mijenja prema izrazu (3) gdje je w faktor inercije (engl. *inertia weight*), $\vec{v}(t)$ je brzina iz prošlog koraka, c_1 je faktor individualnosti, $rand()$ je slučajni broj iz intervala [0, 1],

$\overrightarrow{p_{best}}$ je individualno najbolji položaj, c_2 je faktor socijalnosti, a $\overrightarrow{g_{best}}$ je globalni najbolji položaj.

$$\vec{v}(t+1) = w * \vec{v}(t) + c_1 * rand() * (\overrightarrow{p_{best}} - \vec{x}) + c_2 * rand() * (\overrightarrow{g_{best}} - \vec{x}) \qquad (3)$$

Faktor inercije određuje koliko utjecaja prošla brzina ima na novu i obično se njegova vrijednost nalazi u intervalu [0.4, 0.9]. Veći faktor inercije podržava istraživanje novih prostora i koristi se na početku algoritma dok niži faktor inercije služi za fino podešavanje već postojećih rješenja.

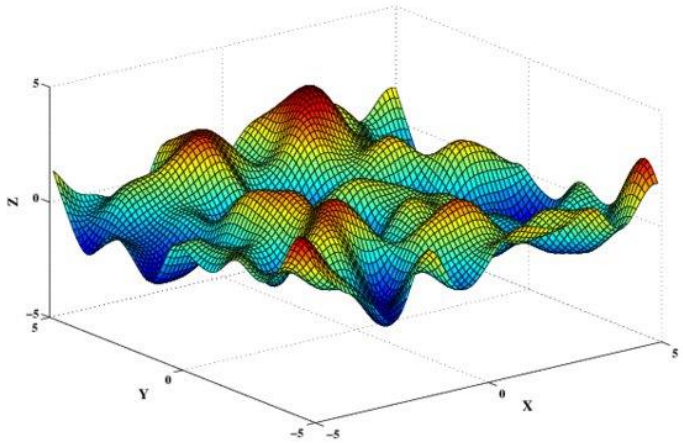
Faktor individualnosti (engl. *cognitive coefficient*) i socijalni faktor (engl. *social coefficient*) se najčešće nalaze u intervalu [1.5, 2]. Veća vrijednost faktora individualnosti potiče istraživanje prostora oko lokalno pronađenog najboljeg rješenja dok veći stupanj socijalnog faktora podržava potragu u smjeru globalno najboljeg rješenja.

Iz izraza (3) vidi se da jedinke mijenjaju položaj na temelju svog iskustva i iskustva susjeda. Ako se kao susjedstvo gleda cijela populacija tada je globalno najbolja jedinka stvarno najbolja jedinka populacije, no budući da su populacije često velike i radi boljeg istraživanja prostora često se kao susjedstvo uzima manji dio populacije. U tom slučaju globalni optimum je zapravo najbolja jedinka iz susjedstva koje se promatra.

U slučaju binarnih varijabli, kao što je to slučaj u NSP-u, položaj jedinke se mijenja drugačije nego kada je riječ o realnim varijablama. Koristi se izraz (4) gdje je S funkcija sigmoide i na taj način se mijenja položaj jedinke, tj. vrijednost njenih bitova:

$$X(t+1) = \begin{cases} 0, & if\ rand() \geq S(v(t+1)) \\ 1, & if\ rand() < S(v(t+1)) \end{cases} \qquad (4)$$

PSO ostvaraje odlične rezultate kod funkcija koje imaju više lokalnih optimuma (kao što je ploha na Slika 2.6), pogotovo kod vektorskih funkcija više varijabli kod kojih drugi algortimi imaju problema sa zapinjanjem u lokalnom optimumu. To se događa budući da se drugi algoritmi, kao što je npr. genetski, oslanjaju na derivaciju funkcije, a PSO ne radi s derivacijama funkcije nego s usmjerenjima čestica na različitim položajima koje konstantno imaju informaciju i o lokalnom/vlastitom i o globalnom optimumu. Pseudokod algoritma prikazan je u kodu 2.2.



Slika 2.6: Funkcija s više lokalnih optimuma [13]

```
// inicijaliziraj_populaciju:
za i = 1 do VEL_POP
	za d iz 1 do DIM
		x[i][d] = random(xmin[d], xmax[d])
		v[i][d] = random(vmin[d], vmax[d])
	kraj
kraj

ponavljaj dok nije kraj
	// evaluiraj_populaciju:
	za i = 1 do VEL_POP
		f[i] = funkcija(x[i]);
	kraj

	// ima li čestica svoje bolje rješenje?
	za i = 1 do VEL_POP
		ako je f[i] bolji od pbest_f[i] tada
			pbest_f[i] = f[i]
			pbest[i] = x[i]
		kraj
	kraj

	// ima li čestica globano najbolje rješenje?
	za i = 1 do VEL_POP
		ako je f[i] bolji od gbest_f[i] tada
			gbest_f[i] = f[i]
			gbest[i] = x[i]
		kraj
	kraj
```

```
        // ažuriraj brzinu i poziciju £estice
        za i = 1 do VEL_POP
               za d iz 1 do DIM
                      v[i][d] = v[i][d]
                                    + c1*rand()*(pbest[i][d]- x[i][d])
                                    + c2*rand()*(gbest[d]-x[i][d])
                      v[i][d] = iz_opsega(v[i][d], vmin[d], vmax[d])
                      x[i][d] = x[i][d] + v[i][d]
               kraj
        kraj
kraj
```

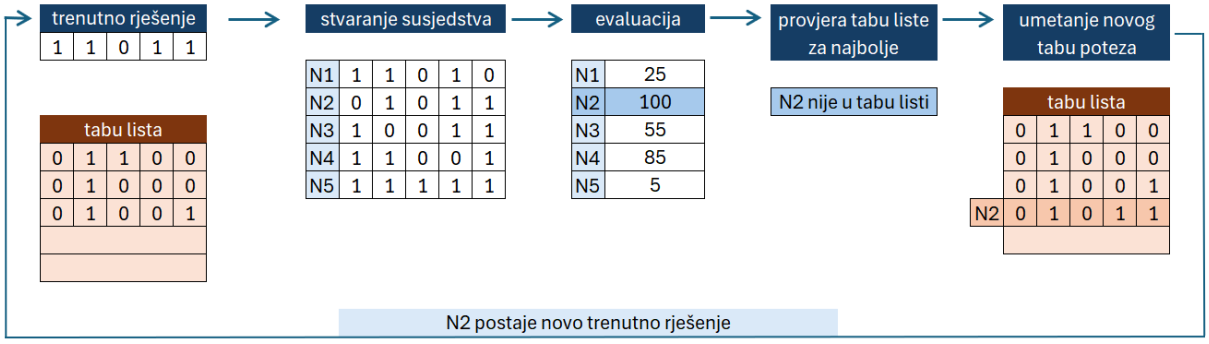
Kôd 2.2 – Pseudokod algoritma roja čestica [8]

2.3. Tabu-lokalna pretraga

Tabu-lokalna pretraga ne spada pod algoritme evolucijskog računanja, nego u drugu porodicu metaheuristika koje rade nad jednim rješenjem. Iako ne radi s populacijom rješenja, kod kombinatornih problema kao što je raspoređivanja medicinskih sestara u smjene pokazat će odlične rezultate.

Algoritam kreće s inicijalizacijom samo jednog rješenja. Iz tog rješenja stvara se novo susjedstvo rješenja na način da se nad početnim rješenjem rade blage modifikacije. Cijelo susjedstvo se evaluira i odabire se jedno najbolje rješenje s kojim se ulazi u sljedeću iteraciju algoritma. No, prije prelaska u sljedeću iteraciju prolazi se korak po kojem je algoritam i dobio ime, a to je provjera tabu-liste. Tabu-lista pamti zadnjih N starih početnih rješenja i to su rješenja koja se ne smiju uzeti kao sljedeća. Na taj se način izbjegavaju ciklusi čime se osigurava da se ne pretražuju stalno isti prostori rješenja. Dakle, prije nego što se najbolje rješenje iz susjedstva prenese u sljedeću iteraciju provjerava se je li ono već u tabu-listi. Ako je, uzima se sljedeće najbolje iz susjedstva, a ako nije, tada se prenosi u sljedeću iteraciju i dodaje se na kraj liste. U slučaju kada se tabu-lista ispuni do kraja jednostavno se brišu najstarija rješenja koja su stavljena u nju i na taj način se oslobađa prostor za nove tabu-poteze.

Na Slika 2.7 prikazana je jedna iteracija tabu-lokalne pretrage gdje se novo rješenje nije prethodno nalazilo u tabu-listi.



Slika 2.7 – Iteracija tabu-pretrage

3. Primjena evolucijskih algoritama na NSP

Problem raspoređivanja medicinskih sestara u smjene optimizacijski je problem nad kojim se mogu primijeniti razne metaheuristike. Kako ne postoji generalno „najbolja“ metaheuristika koja će neovisno o problemu dati dobre rezultate, u ovom radu korištene su tri metahuristike iz različitih porodica. Implementiran je genetski algoritam koji pripada evolucijskom računanju, algoritam roja čestica koji spada pod populacijske algoritme i tabu-pretraga koja je dio algoritama koji rade s jednim rješenjem.

Bez obzira koji algoritam se koristi prvo je potrebno definirati funkciju dobrote koja rješenju pridružuje numeričku vrijednost proporcionalnu tome koliko je rješenje dobro/loše. No, da bi se uopće mogla definirati funkcija dobrote prvo je potrebno učitati podatke iz NSPLiba.

Sve su klase implementirane koristeći programski jezik Python, a raspoređene su u sljedeće datoteke:

- `LoadData.py`
- `Evaluator.py`
- `GA_DEAP.py`
- `GA_PyGAD.py`
- `GA_DEAP_NoElitism.py`
- `GA_PyGAD_NoElitism.py`
- `PSO_PySwarms.py`
- `TABU.py` .

Klasa *LoadData.py* odgovorna je za učitavanje podataka iz NSPLiba. Sadrži metodu *load_data()* koja iz datoteke NSPLib\N25\1.nsp učitava opća tvrda ograničenja rasporeda i preference medicinskih sestara, a iz datoteke NSPLib\Cases\1.gen učitava još tvrdih ograničenja koja su specifična za slučaj.

Evaluator.py je klasa u kojoj je implementirana funkcija dobrote. Ta funkcija prima jedno rješenje i vraća broj prekršaja ograničenja pomnožen konstantom, što znači da funkciju dobrote treba minimizirati jer su bolja ona rješenja s manjim brojem prekršaja. Ostvarena je kroz funkciju *eval()* prikazanu u nastavku.

```
def eval(self, individual):
      return 100000 * hard_constraint_violations(schedules_by_nurse)
                    + soft_constraint_violation(schedules_by_nurse)
```

Funkcija *hard_constraint_violations()* broji koliko prekršaja tvrdih ograničenja ima rješenje. U njoj se pozivaju druge funkcije koje broje koliko puta i za koliko je nezadovoljen minimalni broj medicinskih sestara u smjenama i one prekršaje koji se ne uklapaju u minimalna i maksimalna ograničenja dodijeljenih i uzastopnih smjena. Funkcija *soft_constraint_violation()* mjeri nezadovoljstvo medicinskih sestara rasporedom, odnosno koliko raspored nije u skladu s njihovim preferencijama.

Kršenje tvrdih ograničenja kažnjava se 10^5 puta više nego kršenje mekih ograničenja budući da postojanje mekih ograničenja uopće ne dovodi u pitanje valjanost rješenja dok se tvrda ograničenja moraju zadovoljiti. Dakle, tvrda ograničenja važnija su od mekih i zato se njihovo kršenje mora više kažnjavati.

Za implementiranje genetskog algoritma korištene su dvije različite knjižnice, DEAP (*Distributed Evolutionary Algorithms In Python*) i PyGAD (*Python Genetic Algorithm*).

Knjižnica DEAP ima implementiranu klasu *eaSimple* čija je inicijalizacija dana u nastavku.

```
deap.algorithms.eaSimple(population, toolbox, cxpb, mutpb, ngen,
                         [stats, halloffame, verbose])
```

Klasi se pri definiranju predaje populacija, *toolbox* i parametri *cxpb, mutpb* i *ngen.* Opcionalni parametri su *stats* (objekt klase *Statistics* gdje se definiraju statistike koje se žele pratiti kao što je npr. srednja pogreška), *halloffame* (objekt klase *HallOfFame* gdje se spremaju najbolje jedinke dobivene elitizmom) i *verbose* (parametar koji može biti *True* ili *False* ovisno o tome hoće li se ispisivati međurezultati).

Parametri *cxpb, mutpb* i *ngen* redom označavaju vjerojatnost križanja roditelja, vjerojatnost mutacije djeteta i maksimalni broj generacija, a njihove vrijednosti u ovom radu su postavljene na 0.5, 0.2 i 500. Broj elitističnih jedinki ograničen je na 20.

Toolbox koji se također predaje kao parametar je instanca klase *Toolbox* koja služi kao spremnik za sve alate evolucijskog algoritma. Nakon što se stvori instanca klase u nju se registriraju razni atributi i funkcije koji se onda nigdje ne moraju ponovno pozivati nego ih algoritam sam pronalazi u već napravljenom spremniku. U kodu 2.3 je prikazan dio programskog koda iz

datoteke *GA_DEAP.py* gdje se definira jedan *toolbox.* U njemu se prvo definira tip podataka rješenja *Integer* iz intervala [0, 1], a zatim se stvara pojedinac/rješenje gdje se taj tip podatka ponavlja onoliko puta koliko rješenje ima gena, tj. varijabli. Zatim se definira populacija kao lista od 300 jedinki, a nakon nje definira se funkcija za ocjenjivanje rješenja, tj. funkcija dobrote, kojoj se predaje referenca na funkciju *eval_function()*. Ta funkcija samo poziva već prethodno spomenutu funkciju *eval()* iz klase *Evaluator*. Na kraju se definiraju funkcija selekcije, kao turnirska selekcija s veličinom turnira tri, i funkcija mutacije kao funkcija slučajnog okretanja vrijednosti bita s vjerojatnošću 2%.

```
self.toolbox = base.Toolbox()
        self.toolbox.register("attr_bool", random.randint, 0, 1)
        self.toolbox.register("individual", tools.initRepeat, creator.Individual,
self.toolbox.attr_bool, n=self.num_genes)
        self.toolbox.register("population", tools.initRepeat, list,
self.toolbox.individual, n=300)
        self.toolbox.register("evaluate", eval_function)
        self.toolbox.register("mate", tools.cxTwoPoint)
        self.toolbox.register("select", tools.selTournament, tournsize=3)
        self.toolbox.register("mutate", tools.mutFlipBit, indpb=0.02)
```

Kôd 2.3 – Pseudokod algoritma roja čestica

Kako bi se usporedile različite knjižnice, genetski algoritam implementiran je i s pomoću knjižnice PyGAD. U njoj se nalazi klasa *GA*, a primjer njene instance koja je korištena u radu prikazan je u kodu 2.4.

```
ga_instance = pygad.GA(
            gene_type=int,
            num_genes=self.num_genes,
            sol_per_pop=self.sol_per_pop,
            initial_population=initial_population,
            num_generations=self.num_generations,
            fitness_func=self.fitness_func,
            parent_selection_type="tournament",
            K_tournament=3,
            num_parents_mating=self.num_parents_mating,
            crossover_type="two_points",
            mutation_type="random",
            mutation_probability=0.01,
            keep_elitism=20,
            on_generation=self.callback_generation,
            suppress_warnings=True
        )
```

Kôd 2.4 – Pseudokod algoritma roja čestica

Slično kao i kod knjižnice DEAP, prvo se definira tip gena, broj gena u jedinci, broj jedinki u populaciji i početna populacija. Zatim se definira funkcija dobrote kojoj se također prosljeđuje funkcija *eval()* iz datoteke *Evaluator.py*, tip selekcije, broj roditelja za operaciju križanja i vrsta križanja, vrsta mutacije, vjerojatnost mutacije te broj jedinki za elitizam. Svi parametri postavljeni su na iste vrijednosti kao i oni u algoritmu *eaSimple* knjižnice DEAP.

Algoritam roja čestica implementiran je pomoću knjižnice PySwarms. Budući da je problem raspoređivanja medicinskih sestara u smjene diskretni binarni problem korišten je prikladni modul *pyswarms.discrete.binary*. Instanca klase i parametri *options* prikazani su u kodu 2.5.

Klasa *BinaryPSO* prima broj čestica koje čine populaciju, dimenzije rješenja koje su jednake broju gena u jedinci i atribut *options* koji je rječnik varijabli specifičnih za ovaj algoritam. Konstante *c1* i *c2* su socijalni i individualni faktor, *w* je faktor inercije, *k* je broj susjeda koji uspoređuju najbolje pronađena rješenja – lokalne optimume, a *p* je oznaka za red Minkowskijeve udaljenosti. Minkowskijeva udaljenost reda 2 jednaka je euklidskoj udaljenosti.

```
self.options = {
            'c1': 2.0,
            'c2': 2.0,
            'w': 0.9 ,
            'k': 35,
            'p': 2
       }
pso_instance = ps.discrete.BinaryPSO(
            n_particles=700,
            dimensions=self.dimensions,
            options=self.options
        )
```

Kôd 2.5 – Pseudokod algoritma roja čestica

Tabu-lista je s obzirom na svoju jednostavnost implementirana ručno i nisu korištene dodatne knjižnice za metaheuristike. Također koristi istu funkciju dobrote kao i ostali algoritmi, a to je funkcija *eval()* iz datoteke *Evaluator.py*. Tabu-lista ostvarena je kao struktura podataka reda (engl. deque) iz Pythonovog modula *collections*. Takva struktura podataka slijedi princip FIFO (engl. *First-In-First-Out*) koji je pogodan za tabu-listu koja ima ograničenu veličinu i koja mora maknuti stare podatke kada se lista napuni do kraja. Veličina tabu-liste u ovom problemu ograničena je na 50 jedinki.

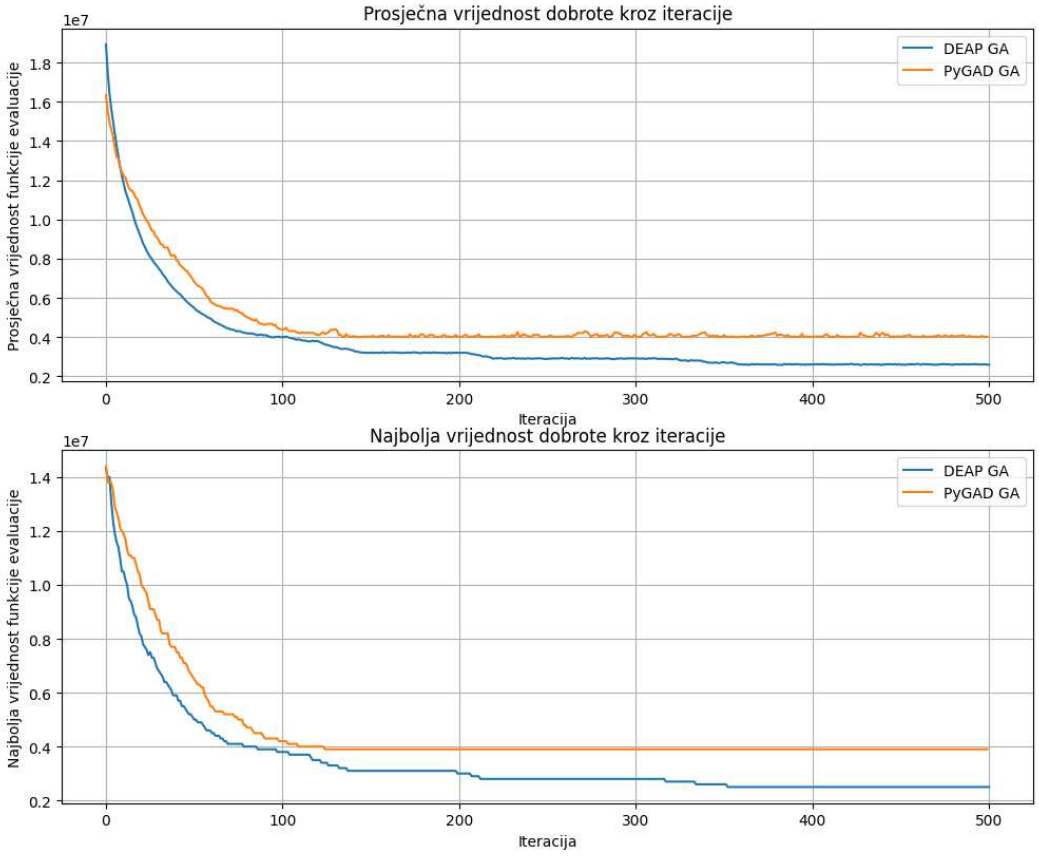
Susjedstvo se generira pomoću funkcije prikazane u kodu 2.6. Funkcija prihvaća trenutno rješenje, kopira ga te slučajnim odabirom uzima jedan indeks na čijem će se mjestu promijeniti bit. Takvo promijenjeno rješenje čini jednog susjeda, a broj susjeda u susjedstvu ograničen je na 15.

```
def generate_neighbor(self, solution):
      neighbor = solution.copy()
      idx = np.random.randint(len(solution))
      neighbor[idx] = 1 - neighbor[idx]
      return neighbour
```

Kôd 2.6 – Funkcija za generiranje susjeda tabu-pretrage

4. Usporedba rješenja

Iako knjižnice DEAP i PyGAD koriste iste parametre i ostvaruju isti algoritam, one dobivaju različite rezultate. Na Slika 4.1 prikazana su dva grafa koja vizualiziraju ponašanje algoritama kroz srednje i najbolje vrijednosti rješenja tijekom petsto iteracija. Genetski algoritam koji je implementiran pomoću knjižnice DEAP dobiva bolja rješenja, a njegovo najbolje konačno rješenja s vrijednošću 2.500.576 je znatno bolje od najboljeg rješenja dobivenog uz pomoć knjižnice PyGAD koje ima vrijednost funkcije dobrote 4.600.924. Iako se obje te vrijednosti mogu učiniti velike, one svoju veličinu duguju množenjem tvrdih ograničenja velikom konstantom. Za usporedbu, najbolje DEAP rješenje s funkcijom dobrote 2.500.576 ima 25 prekršaja tvrdih ograničenja i 576 prekršaja preferencija. Od tih 25 tvrdih prekršaja nijedan nije prekršaj minimalnog potrebnog broja medicinskih sestara u smjeni, nego svi dolaze od kršenja ograničenja specifičnih za slučaj (minimalni i maksimalni broj ukupno dodijeljenih smjena kao i broj uzastopnih smjena).

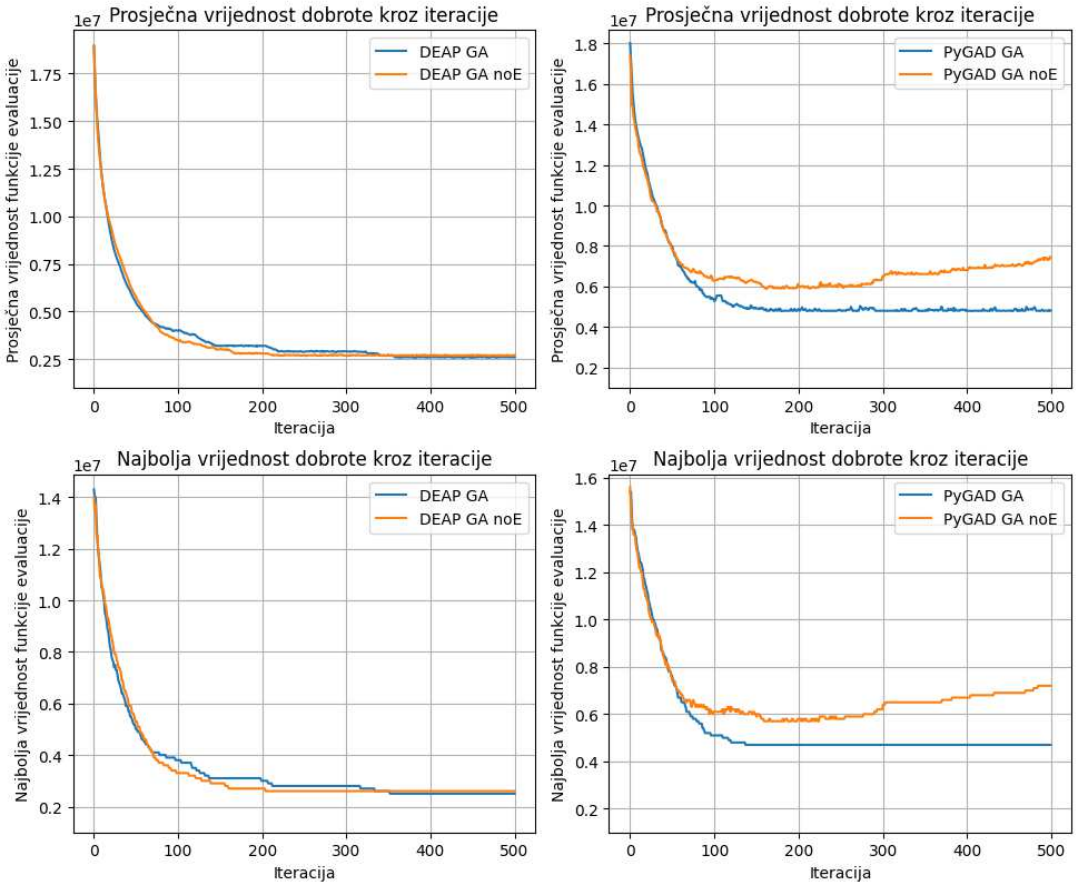


Slika 4.1 Usporedba knjižnca DEAP i PyGAD

Iz prikaza najboljih rješenja može se još primijetiti kako algoritam knjižnice DEAP i dalje nakon petsto iteracija ima mogućnosti pronaći bolje rješenje, dok algoritam knjižnice PyGAD nakon stote iteracije stagnira i više ne pronalazi bolja rješenja.

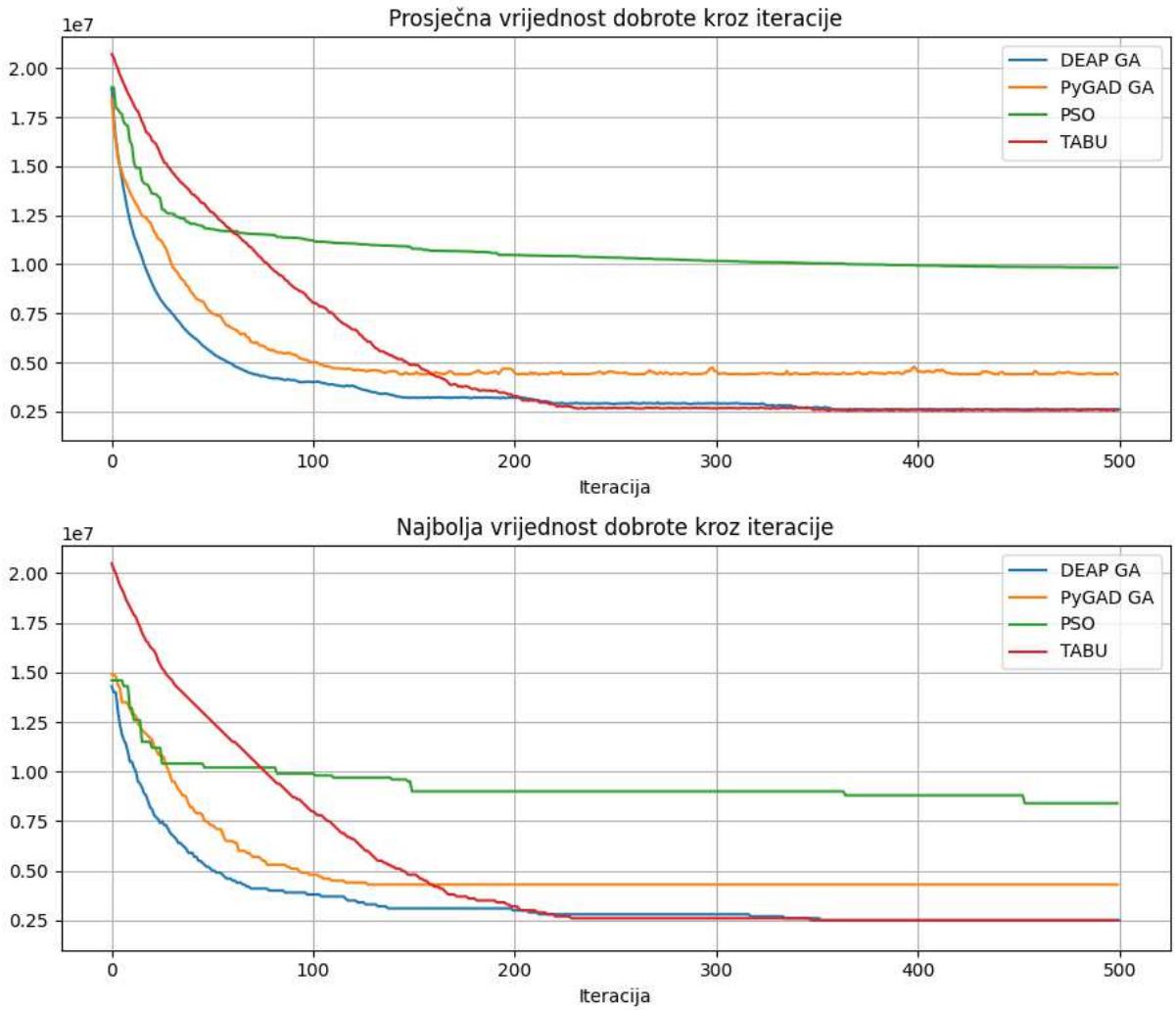
Kako bi se ispitao utjecaj elitizma, oba genetska algoritma dobila su svoju verziju bez implementiranog elitizma. Svi ostali parametri ostali su nepromijenjeni. Usporedba je vidljiva na Slika 4.2.

U slučaju isključenja elitizma razlika između knjižnica DEAP i PyGAD još više dolazi do izražaja. Kod algoritma iz DEAP-a nema značajnih razlika. Krajnja najbolja rješenja su skoro identična – jedan prekršaj tvrdih ograničenja više u slučaju bez elitizma. Algoritam iz PyGAD-a, s druge strane, bez elitizma nailazi na velike fluktuacije. U grafu se jasno vidi kako isključenje elitizma rezultira velikim amplitudama u konvergenciji. Vrijednosti rješenja se do dvjestote iteracije smanjuju nakon čega dolazi do preobrata i algoritam kreće u krivom smjeru te pronalazi sve gora i gora rješenja. Krajnje najbolje rješenje nakon 500 iteracija sada ima 71 prekršaj tvrdih ograničenja umjesto prijašnjih 43.



Slika 4.2 Utjecaj elitizma na uspješnost genetskog algoritma

Usporedba sva četiri implementirana algoritma prikazana je na Slika 4.3. Do najboljih rješenja dolaze tabu-pretraga i genetski algoritam implementiran pomoću knjižnice DEAP. Na kraju izvođenja petsto iteracija najbolje rješenje je ono dobiveno tabu-algoritmom koje ima 25 prekršaja tvrdih ograničenja i 524 prekršaja mekih ograničenja. Slijedi ga rješenje dobiveno DEAP-ovim genetskim algoritmom koje završava s istim brojem prekršaja tvrdih ograničenja i 50-ak prekršaja mekih ograničenja više. Detaljnije vrijednosti krajnjih rješenja vidljive su u Tablica 4.1. Iako genetski algoritam brže konvergira, tabu-lista postiže zavidne rezultate s obzirom na puno manju složenost algoritma. Genetski algoritam ostvaren knjižnicom PyGAD ima 43 prekršaja tvrdih ograničenja, dok je uvjerljivo najgore rezultate ostvario PSO-algoritam s 84 prekršaja tvrdih ograničenja. Zanimljivo je vidjeti kako graf funkcije evaluacije najboljih jedinki za algoritam PSO ima stepenastu krivulju budući da mu je već nakon tridesete iteracije potrebno puno vremena kako bi našao sljedeće bolje rješenje.



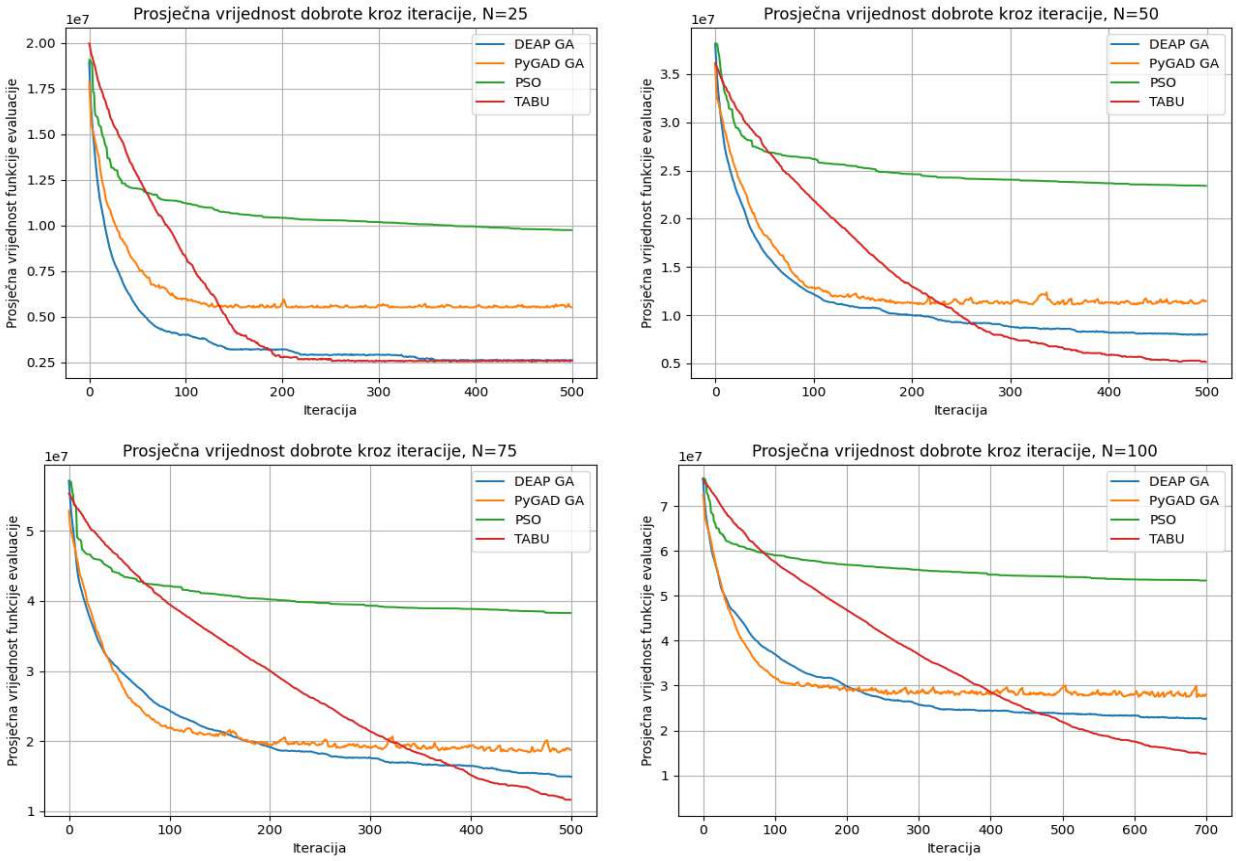
Slika 4.3 Usporedba četiri metaheuristike nad NSP

	Najbolje rješenje NSPLib/N25		
Algoritam	Broj prekršaja tvrdih ograničenja	Broj prekršaja mekih ograničenja	Vrijednost funkcije dobrote
TABU	25	524	2.500.524
DEAP GA	25	576	2.500.576
PyGAD GA	43	904	4.300.904
PSO	84	636	8.400.636

Tablica 4.1 – Najbolja rješenja dobivena podacima iz NSPLib/N25

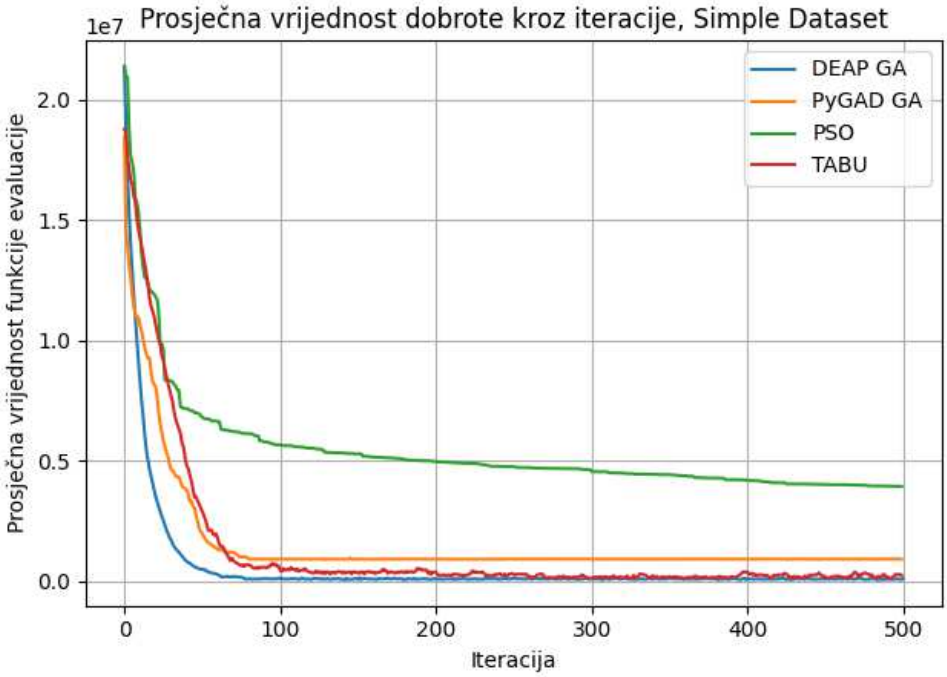
Iako se iz tablice vidi da nijedno dobiveno najbolje rješenje nije valjano, važno je spomenuti kako je bez obzira na to zadovoljeno glavno tvrdo ograničenje, a to je minimalan broj medicinskih sestara po smjenama. Ostali tvrdi prekršaji većinom dolaze od toga što raspored i dalje sadrži prevelik broj uzastopnih smjena.

Svi prethodni rezultati su dobiveni korištenjem ulazne datoteke NSPLib\N25\1.nsp koja sadrži podatke za 25 medicinskih sestara. Kako se povećava broj medicinskih sestara tako rezultati algoritama postaju sve lošiji. Razlog leži u tome što se s povećanjem broja osoba za koje se optimira raspored proporcionalno povećava broj varijabli, pa tako prelaskom s 25 medicinskih sestara na 50 broj varijabli koje treba optimirati skače sa 700 na 1400. Izvođenje algoritama također traje osjetno duže. Na Slika 4.4 prikazana je usporedba srednjih vrijednosti funkcije dobrote za rješenja koja se dobiju korištenjem podskupova podataka iz NSPLib: N25, N50, N75 i N100 koji sadrže podatke za 25, 50, 75 i 100 medicinskih sestara.



Slika 4.4 Utjecaj broja medicinskih sestara na uspješnost pronalaska rješenja

Kako bi se ipak pokazala sposobnost svih algoritama da brzo i učinkovito pronađu rješenje koje je valjano i nema prekršaja tvrdih ograničenja, svi algoritmi su pokrenuti i nad jednostavnijim skupom podataka [14]. U tom skupu podataka nalazi se 13 medicinskih sestara i također je potrebno napraviti tjedni raspored u tri smjene koji zadovoljava ograničenja minimalnog i maksimalnog broja dodijeljenih smjena kao i ograničen broj uzastopnih smjena. Osim tvrdih ograničenja u skupu podataka nalaze se i preferencije medicinskih sestara za željene smjene. Razlika između jednostavnijih preferencija i onih iz NSPLiba je u tome što je sada i preferenca binarna – medicinska sestra samo preferira određenu smjenu ili ne, nema podataka o jačini preference. Broj varijabli koji se u ovom slučaju optimira je 13 (medicinskih sestara) * 3 (smjene) * 7 (dana), tj. 273. Rezultati su vidljivi na Slika 4.5.



Slika 4.5 Primjena algoritama na jednostavnijem skupu podataka

Genetski algoritam iz knjižnice DEAP i tabu pretraga sada ostvaruju potpuno valjana rješenja koja nemaju prekršaje tvrdih ograničenja. Njihovi rezultati se minimalno razlikuju – u dva prekršaja mekih ograničenja. Iako je u ovom slučaju tabu-pretraga za nijansu bolja, ipak se prilikom više pokretanja algoritama uspostavlja da ona ne pronalazi valjano rješenje svaki put, dok DEAP-ov genetski algoritam u svakom pokretanju nalazi valjano rješenje. S druge strane PyGAD-ov genetski algoritam i algoritam roja čestica i dalje ne uspijevaju dobiti valjana rješenja.

	Najbolje rješenje Simple Dataset		
Algoritam	**Broj prekršaja tvrdih ograničenja**	**Broj prekršaja mekih ograničenja**	**Vrijednost funkcije dobrote**
TABU	0	73	73
DEAP GA	0	75	75
PyGAD GA	9	89	900.089
PSO	23	82	2.300.082

Tablica 4.2 Najbolja rješenja na skupu Simple Dataset

Iako algoritmi ostvareni knjižnicama DEAP, PyGAD i PySwarms nisu uspjeli dobiti valjana rješenja, kao što to nije uspio ni algoritam tabu-pretrage, na službenoj stranici NSPLiba [15] objavljeni su rezultati algoritama koji dolaze do odličnih rješenja.

Autori NSPLiba, Broos Maenhout i Mario Vanhoucke, 2006. godine objavili su članak *An electromagnetic meta-heuristic for the nurse scheduling problem* gdje su predstavili rješenje problema raspoređivanja medicinskih sestara u smjene korištenjem algoritma elektromagnetske metaheuristike (engl. *electromagnetic meta-heuristic*). Ova metaheuristika spada pod populacijske metaheuristike, a njen se pristup temelji na principima elektromagnetizma gdje se rješenja međusobno mogu privlačiti ili odbijati. Bolja rješenja privlače ostatak populacije, dok lošija rješenja odbijaju druga rješenja. Korištenjem takvog algoritma ostvareni su odlični rezultati. U mjerenjima rezultata korišten je kriterij zaustavljanja od 5000 evaluacija rješenja, a elektromagnetska metaheuristika je nad svim skupovima podataka iz NSPLiba i nad svim specifičnim slučajevima ostvarila minimalno 80% valjanih rješenja u populaciji.

Isti istraživači kasnije objavljuju radove gdje proučavaju genetske algoritme [16] nad problemom raspoređivanja medicinskih sestara u smjene. Koriste evolucijske operacije prilagođene ovom problemu, kao što je redoslijedno križanje (engl. *order crossover*) gdje se u novoj jedinki zadržava relativan poredak gena kao kod roditelja. Također se implementiraju hibridni pristupi, kao što je npr. lokalna pretraga nakon križanja, koji dorađuju rješenja. Izmijenjeni genetski algoritmi na ovaj način ostvaraju vrlo dobra rješenja.

Zaključak

Problem raspoređivanja medicinskih sestara u smjene potvrdio je svoje obilježje NP-teških problema. Nemoguće je unaprijed na temelju ulaznog skupa podataka i odabrane metaheuristike predvidjeti hoće li se uspjeti pronaći valjano rješenje.

Na uspješan pronalazak rješenja najviše utječe veličina ulaznog skupa podataka s obzirom na to da ona izravno određuje broj varijabli koje algoritam optimizira. To znači da će na izradu rasporeda za raspoređivanje medicinskih sestara u smjene najviše utjecati broj medicinskih sestara i duljina vremenskog perioda za koje se raspored izrađuje. Složenost ograničenja također ima velik utjecaj na pronalazak valjanog rješenja.

Rješenja metaheuristika gube na kvaliteti prilikom povećavanja složenosti problema, a broj prekršaja tvrdih ograničenja se pritom također povećava. Iako nijedna metaheuristika nije uspjela dobiti valjano rješenje nad podacima iz NSPLiba, tabu pretraga i genetski algoritam najbolje su uspjeli minimizirati prekršaje tvrdih ograničenja. Do valjanih rezultata, tj. onih rasporeda gdje se ne krši nijedno tvrdo ograničenje, se dolazi korištenjem jednostavnijeg skupa podataka gdje se treba optimirati manji broj varijabli. U tom slučaju valjana rješenja pronalaze tabu-pretraga i genetski algoritam knjižnice DEAP koji prilikom svakog pokretanja uspijeva pronaći dobro rješenje.

Iako je pseudokod genetskog algoritma na prvi pogled jednostavan za implementirati svejedno se mogu javiti velike razlike u radu između dviju implementacija. To se vidi na primjeru korištenja genetskog algoritma iz dviju knjižnica – DEAP i PyGAD. Iako se radi o istom algoritmu i istim parametrima svejedno dolazi do velike razlike u rezultatima gdje prednost uvijek uzima onaj algoritam iz knjižnice DEAP.

Iznenađujuće dobre rezultate ostvaraje tabu-pretraga s obzirom na to da slijedi najjednostavnije principe. Princip tabu-liste ovog algoritma štiti ga od pojave ciklusa i ponavljanja pretrage istih prostora, dok generiranje susjeda na način promjene samo jednog bita u rješenju omogućava fino podešavanje.

Potvrđena je i činjenica da ne postoji najbolja metaheuristika koja će nad svim problemima uspjeti pronalaziti valjana rješenja. Iako se ovdje algoritam roja čestica pokazao kao najgora opcija za rješavanje problema raspoređivanja medicinskih sestara u smjene, u slučaju da se radilo o problemu pronalaženja minimuma plohe s više lokalnih optimuma tada bi PSO

vjerojatno pokazao puno bolje rezultate. Algoritam roja čestica češće se primjenjuje na kontinuirane optimizacijske probleme gdje matematičke formule za brzinu i pomak preciznije pomiču česticu [17].

Svaka implementacija spomenutih algoritama ima još prostora za napredak. Najveći potencijal za poboljšanje leži u dinamičkom programiranju, tj. mijenjaju parametara i funkcija samog algoritma prilikom njegovog izvođenja. Kod genetskog algoritma to bi značilo npr. smanjiti faktor mutacije pri zadnjim iteracijama kako bi se potaknula eksploatacija, a kod tabu-liste istraživanje već dobrih rješenja može se poboljšati tako da se za susjedstvo ne uzimaju više bilo koja mutirana rješenja nego samo ona koja su bolja od trenutnog. Algoritam roja čestica može smanjiti faktor inercije pred kraj izvođenja kako bi izbjegavao velike pomake i na taj način omogućio fino podešavanje rješenja.

Iako su vremenski zahtjevne i ne postoje točna pravila i upute koje se mogu slijediti za dobivanje dobrih rezultata, vjerujem kako je metaheuristike moguće primijeniti na optimizacijske probleme iz realnog svijeta kao što je izrada rasporeda za smjene medicinskih sestara. Najveća njihova prednost leži u tome što kada se jednom algoritam dobro podesi i postane robustan, male promjene u ulaznim podacima ili ograničenjima neće utjecati na njegov rad. To je svakako korisno u problemima kao što je problem raspoređivanja medicinskih sestara u smjene s obzirom na to da se često mogu dogoditi situacije kada je raspored potrebno izmijeniti zbog npr. promjene zaposlenika pa posljedično i njegovih preferencija. U svakom slučaju prednost se treba dati korištenju metaheuristika u odnosu na ručno rješavanje ovakvih problema.

Literatura

[1] Planday, *The evolution of employee scheduling: A look at past, present and future* (2023, svibanj). Poveznica: https://www.planday.com/resources/articles/the-evolution-of-employee-scheduling-a-look-at-past-present-and-future/; pristupljeno 2. svibnja 2024.

[2] Gass, S. I. *Model World: In the Beginning There Was Linear Programming*, Interfaces, 20,4 (1990), str. 128–132.

[3] Miller, Holmes E.; Pierskalla, William P.; Rath, Gustave J. *Nurse Scheduling Using Mathematical Programming*, Operations Research, 24,5 (1976), str. 857–870.

[4] Warner, D. Michael *Scheduling Nursing Personnel According to Nursing Preference: A Mathematical Programming Approach*, Operations Research. 24,5 (1976), str. 842–856.

[5] Sörensen, K., Sevaux, M., Glover, F. *Handbook of Heuristics*. Cham: Springer, 2018.

[6] Helmenstine, Anne M. *How Many Atoms Exist in the Universe?,* ThoughtCo., (2019, kolovoz). Poveznica: https://www.thoughtco.com/number-of-atoms-in-the-universe-603795, pristupljeno: 1. lipnja 2024.

[7] Vanhoucke M., Maenhout B. *NSPLib - A Nurse Scheduling Problem Library: A tool to evaluate (meta-)heuristic procedures.* Faculty of Economics and Business Administration, Ghent University. 2005.

[8] Čupić M., *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.* 2013.

[9] M. Dorigo, T. Stützle *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[10] Zervoudakis, Konstantinos; Tsafarakis, Stelios *A global optimizer inspired from the survival strategies of flying foxes.* Engineering with Computers. 39,2 (2023), str. 1583–1616.

[11] Fathollahi-Fard, Amir Mohammad; Hajiaghaei-Keshteli, Mostafa; Tavakkoli-Moghaddam, Reza *Red deer algorithm (RDA): a new nature-inspired meta-heuristic*. Soft Computing. 24,19 (2020) str. 14637–14665.

[12] Dhiman, Gaurav; Kumar, Vijay *Emperor penguin optimizer: a bio-inspired algorithm for engineering problems*. Knowledge-Based Systems. 159 (2018), str. 20–50.

[13] Chen Q., Wong N. *New Simulation Methodology of 3D Surface Roughness Loss for Interconnects Modeling.* Department of Electrical and Electronic Engineering, The University of Hong Kong. 2009

[14] Muafira *Solving Nurse Scheduling/Rostering Problems in Python.* Medium (2022, listopad). Poveznica:https://medium.com/@muafirathasnikt/solving-nurse-scheduling-rostering-problems-in-python-d44acc3ed74f , pristupljeno: 11. lipnja 2024.

[15] Maenhout B. , Vanhoucke M. *An electromagnetic meta-heuristic for the nurse scheduling problem.* Journal of Heuristics. 13 (2007), str. 359-385

[16] Maenhout B. , Vanhoucke M. *Comparison and hybridization of crossover operators for the nurse scheduling problem.* Annals of Operations Research. 159 (2008), str. 333-353

[17] Seixas Gomes de Almeida, B., Coppo Leite V. *Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems*. IntechOpen, 2019.

Sažetak

U ovom radu predstavljen je diskretni binarni problem raspoređivanja medicinskih sestara u smjene. Problem uključuje izradu rasporeda smjena koji zadovoljava tvrda i meka ograničenja. Ulazni podaci uzeti su iz NSPLiba, a rješenja problema dobivena su pomoću implementacije triju metaheuristika – genetskog algoritma, tabu-pretrage i optimizacije roja čestica. Najbolji rezultati ostvareni su genetskim algoritmom i tabu-pretragom koji se i inače koriste za logističke probleme. Algoritam roja čestica nije se pokazao prikladnim za ovu vrstu problema s obzirom na to da se češće koristi nad kontinuiranim optimizacijskim problemima.

Summary

This paper presents a discrete binary nurse scheduling problem. The problem involves creating a shift schedule that satisfies hard and soft constraints. The input data is taken from NSPLib and the problem solutions are obtained using the implementation of three metaheuristics – genetic algorithm, tabu search, and particle swarm optimization. The best results were achieved using the genetic algorithm and tabu search, which are commonly used for logistic problems. The particle swarm optimization algorithm did not prove suitable for this type of problem, as it is more frequently used for continuous optimization problems.

Skraćenice

NSP – Nurse Scheduling Problem

NSPLib – Nurse Scheduling Problem Library

NP – Nondeterministic Polynomial time

TSP – Travelling Salesman Problem

KP – Knapsack Problem

GA – Genetic Algorithm

PSO – Particle Swarm Optimization