Planiranje putanje mobilnog robota kombinacijom A* algoritma i neuronskih mreža

Katić, Rebeka

Master's thesis / Diplomski rad

2025

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:448390

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 713

PLANIRANJE PUTANJE MOBILNOG ROBOTA KOMBINACIJOM A* ALGORITMA I NEURONSKIH MREŽA

Rebeka Katić

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 713

PLANIRANJE PUTANJE MOBILNOG ROBOTA KOMBINACIJOM A* ALGORITMA I NEURONSKIH MREŽA

Rebeka Katić

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 30. rujna 2024.

DIPLOMSKI ZADATAK br. 713

Pristupnica:	**Rebeka Katić (0036523508)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	prof. dr. sc. Marija Seder

Zadatak: **Planiranje putanje mobilnog robota kombinacijom A* algoritma i neuronskih mreža**

Opis zadatka:

Klasični pristupi planiranja putanje mobilnog robota pretraživanjem prostora oslanjaju se na matematičke algoritme i unaprijed definiran radni prostor robota. Iako pouzdani, ovakvi pristupi mogu biti spori na velikim i skučenim prostorima. Kombinacijom neuronskih mreža i klasičnih algoritama, planiranje putanje moglo bi biti mnogo brže i učinkovitije u situacijama kada je potrebno napraviti replaniranje putanje, a da se pritom ne naruši kvaliteta putanje. U ovom diplomskom radu potrebno je implementirati prošireni A* algoritam pretraživanja prostora koristeći neuronske mreže kako bi se pretraživanje ubrzalo i fokusiralo na dijelove prostora oko optimalne putanje. Razvijeni algoritam potrebno je testirati na različitim simulacijskim scenarijima i usporediti s putanjom dobivenom klasičnim A* pretraživanjem.

Rok za predaju rada: 14. veljače 2025.

Sadržaj

1. Uvod

S obzirom na sve raširenu upotrebu mobilnih autonomnih vozila (AV), problem planiranja putanje postao je fokus u području autonomnog upravljanja. Planiranje putanje podrazumijeva određivanje putanje bez kolizija u zadanom okruženju, koje u stvarnom svijetu često može biti kaotično i prepuno prepreka. Kako bi se pojednostavio problem planiranja putanje i osigurao nesmetan rad robota, konfiguracijski prostor mora biti usklađen s algoritmom koji se koristi. Postoji niz algoritama za planiranje putanje i pronalaženje putanje koji se razlikuju u primjenjivosti, ovisno o kinematici sustava, dinamici okruženja, računalnim mogućnostima te dostupnosti podataka sa senzora i drugih izvora.

U ovom radu fokusirat ćemo se na A* algoritam te njegovu primjenu u neuronskoj mreži. A* algoritam se temelji na planiranju putanje pretraživanjem grafa. To je uobičajen pristup rješavanja problema planiranja putanje te ima širok spektar primjene kao što je navigacija autonomnih vozila, manipulacija robotskim rukama i AI u igrama. Za razliku od drugih pristupa kao što su planiranje temeljeno na uzorkovanju i reaktivno planiranje, planiranje temeljeno na pretraživanju jamči pronalaženje rješenja putanje, ako ono postoji, postupnim i opsežnim istraživanjem karte.

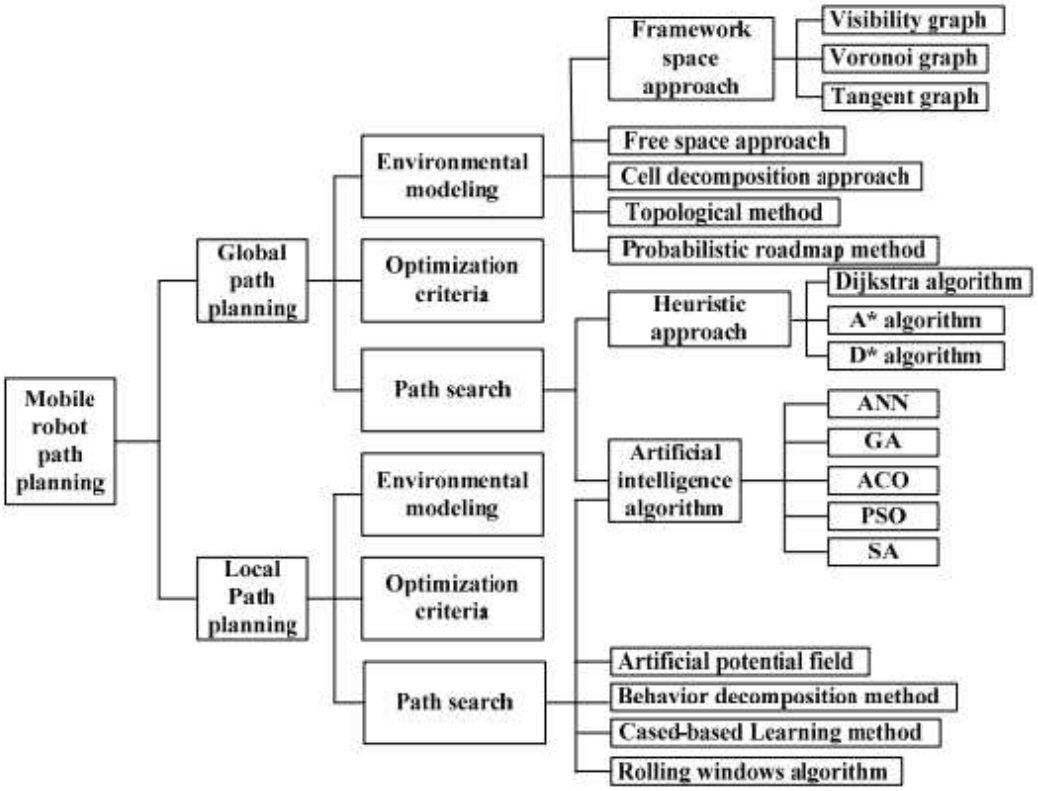
2. Opis problema

Problem planiranja putanje može se opisati na sljedeći način: s obzirom na robota i njegovo radno okruženje, mobilni robot traži optimalni ili suboptimalni putanja od početnog stanja do ciljnog stanja prema određenim kriterijima performansi. Dobra tehnologija planiranja putanje za mobilne robote ne samo da može uštedjeti mnogo vremena, već također smanjuje trošenje i kapitalna ulaganja mobilnog robota. [1]

U primjenama napredne robotike, problem planiranja putanje je definitivno vrlo izazovan, osobito za robote koji se odlikuju visokim stupnjem autonomije ili za robote koji moraju raditi u izazovnim okruženjima (svemir, podvodni svijet, nuklearna, vojna itd.). Najjednostavnija situacija je kada se put treba planirati u statičnom i poznatom okruženju; međutim, općenito, problem planiranja putanje može se formulirati za bilo koji robotski sustav podložan kinematičkim ograničenjima, u dinamičnom i nepoznatom okruženju. [2]

Iako postoji mnogo klasifikacija metoda planiranja putanje, ove metode se okvirno mogu svrstati u dvije kategorije: globalno i lokalno planiranje. Globalno planiranje putanje znači da robot potpuno razumije sve informacije o okolišu, a zatim koristi algoritam za pretraživanje putanje kako bi pronašao optimalni put od početne točke do ciljne točke, nakon čega robot ide tim putem do odredišta. Tijekom tog vremena robot više ne vrši planiranje putanje, pa se ova metoda naziva i offline ili statičnom metodom planiranja putanje. Globalna metoda planiranja putanje uglavnom uključuje metodu pogleda i A* algoritam. Lokalno planiranje putanje znači da je dio informacija o okolišu poznat. Robot se oslanja na ograničene vanjske informacije dobivene od senzora u stvarnom vremenu kako bi izvršio odgovarajući zadatak planiranja. Ova metoda može ostvariti izbjegavanje prepreka u stvarnom vremenu. Metode lokalnog planiranja putanje uglavnom uključuju APF i genetski algoritam (GA). Trenutačno, bez obzira radi li se o A* algoritmu, APF-u

ili drugim algoritmima, iako oni mogu dobro planirati putanju kretanja robota, postoji problem. Budući da različite metode planiranja imaju svoje karakteristike, učinkovitost planiranja putanje nije idealna. [3] Slika 2.1. prikazuje spomenute primjene.



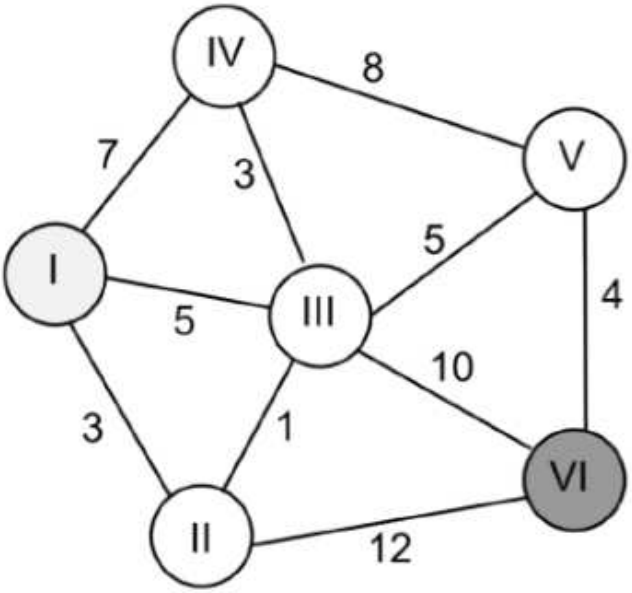
Slika 2.1. Klasifikacija planiranja putanje.

U ovom radu posvetit ćemo se metodi globalnog pretraživanja putanje, konkretnije A* algoritmu. Kao što je vidljivo na Slici 2.1. globalno planiranje putanje dijeli se na Dijkstrin algoritam, A* algoritam i D* Algoritam. U nastavku ćemo se osvrnuti na principe rada Dijkstrinog algoritma i A* algoritma koji kombinira prednosti Dijkstra algoritma i heuristike kako bi poboljšao učinkovitost.

Dijkstrin algoritam

Jedan od najranijih i najjednostavnijih algoritama (Dijkstra, 1959). Počevši od početnog vrha algoritam označava sve izravne susjede početnog vrha s troškom dolaska do njih. Zatim od skupa čvorova uzima čvor s najnižim troškom prema svim njegovim susjednim čvorovima i označava ih troškom dolaska do njih, ako je taj trošak manji. Nakon što su svi susjedi nekog čvora provjereni, algoritam prelazi na čvor sa sljedećim najnižim troškom. Kada algoritam dosegne cilj, završava i robot može slijediti bridove koji vode

prema najnižem trošku brida.



Slika 2.2. Problem planiranja putanje od točke I do točke VI.[4]

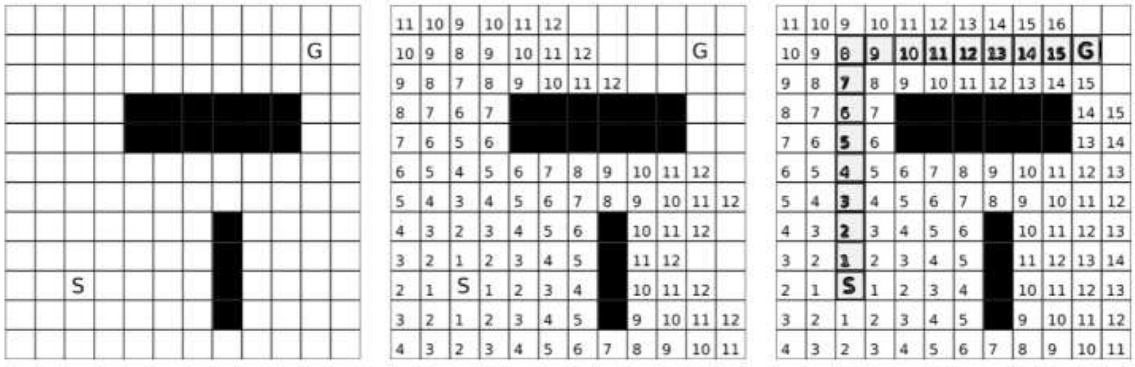
Na slici 2.2., Dijkstra bi prvo označio čvorove II, III i IV s troškom 3, 5 i 7, redom. Zatim bi nastavio istraživati sve bridove čvora II, koji do sada ima najniži trošak. To bi dovelo do otkrića da se čvor III zapravo može doći s $3 + 1 < 5$ koraka, pa bi čvor III bio ponovo označen troškom od 4.Kako bi potpuno procijenio čvor II, Dijkstra prije nego što prijeđe na sljedeći čvor mora procijeniti preostali brid i označi čvor VI s $3 + 12 = 15$.

Čvor s najnižim troškom sada je čvor III(4). Sada možemo ponovo označiti čvor VI s 14, što je manji trošak od 15, i označiti čvor V s $4 + 5 = 9$, dok čvor IV ostaje s $4 + 3 = 7$. Iako smo već pronašli dva puta do cilja, od kojih je jedan bolji od drugog, ne možemo stati jer još uvijek postoje čvorovi s neistraženim rubovima i ukupnim troškom manjim od 14. Zapravo, nastavak istraživanja od čvora V vodi do najkraćeg puta I-II-III-V-VI s troškom 13, bez preostalih čvorova za istražiti.

Kako Dijkstra ne bi stao dok ne postoji čvor s nižim troškom od trenutačnog troška do cilja, možemo biti sigurni da će najkraći put biti pronađen, ako on postoji. Možemo reći da je algoritam potpun.

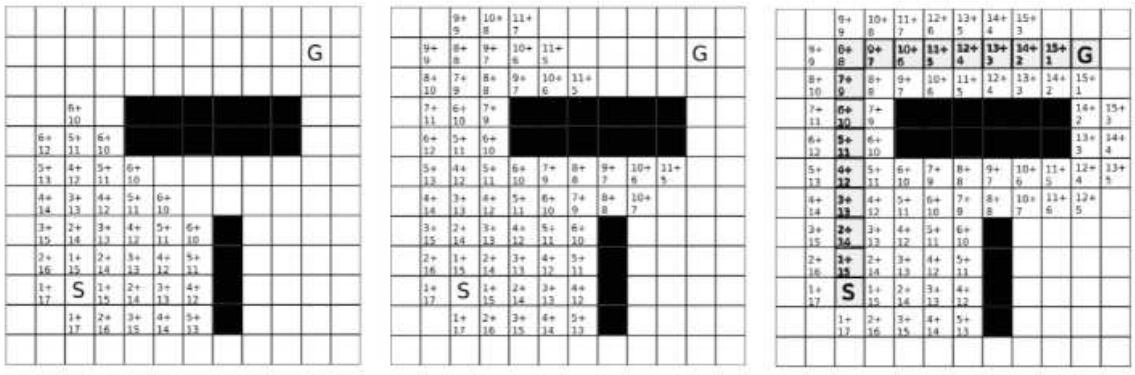
Budući da Dijkstra uvijek prvo istražuje čvorove s najmanjim ukupnim troškom, okoliš se istražuje usporedno s valnim frontama koje potiču od početnog čvora, na kraju dolazeći do cilja. To je, naravno, vrlo neučinkovito, osobito ako Dijkstra istražuje čvorove

daleko od cilja. To se može vizualizirati dodavanjem nekoliko čvorova lijevo od čvora I na slici 2.2. Dijkstra će istraživati sve te čvorove dok njihov trošak ne premaši najniži trošak pronađen za cilj. To se također može vidjeti prateći Dijkstrin algoritam na mreži, kao što je prikazano na slici 2.4.



Slika 2.3. Dijkstrin algoritam pronalazi najkraći put od 'S' do 'G', pri čemu je trošak jednak jedan po svakoj ćeliji. Obratite pažnju na mali broj ćelija koje ostaju neistražene nakon što je najkraći put (sivi) pronađen, jer Dijkstra uvijek prvo razmatra ćeliju s najnižim troškom puta.[4]

A* algoritam kombinira prednosti Dijkstra algoritma i heuristiku za poboljšanje učinkovitosti. Koristi pretraživanje temeljen na strategiji pretrage najboljeg prvog (*eng .best-first search, BFS*) i pronalazi put s najmanjim troškom od početnog čvora do ciljnog čvora koristeći heuristiku za procjenu troška dolaska do cilja.[5] Slika 2.3. za heuristiku koristi Manhattanova udaljenost.



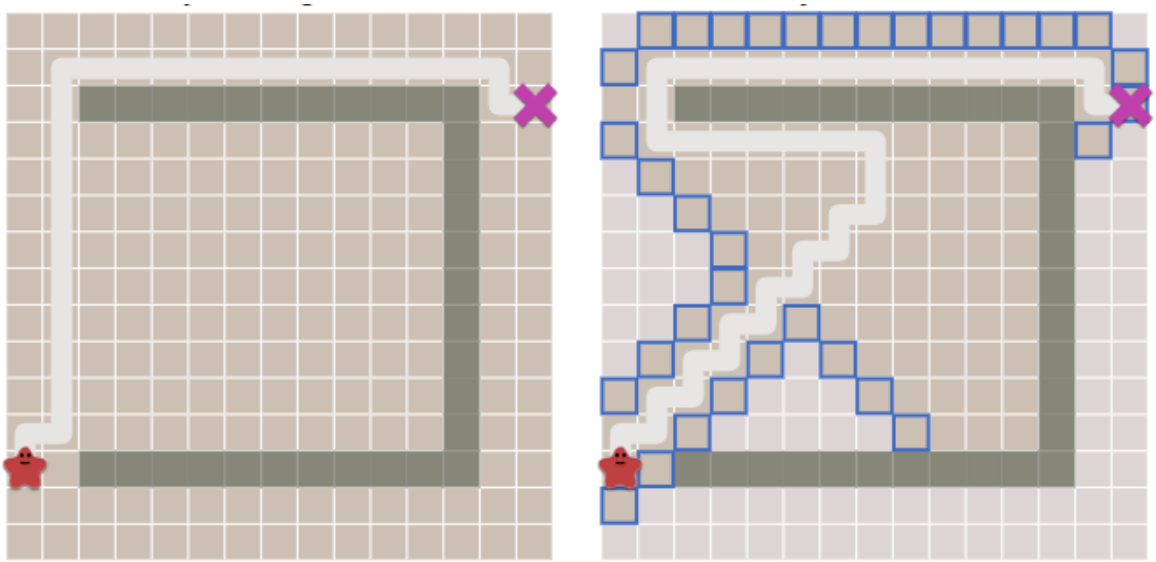
Slika 2.4. Pronalaženje najkraćeg puta od 'S' do 'G' s troškom od jedan po mrežnoj ćeliji koristeći A* algoritam. Slično Dijkstrinom algoritmu, A* ocjenjuje samo ćeliju s najnižim troškom, ali uzima u obzir procjenu preostale udaljenosti.[4]

3. Opis algoritama

3.1. A* algoritam

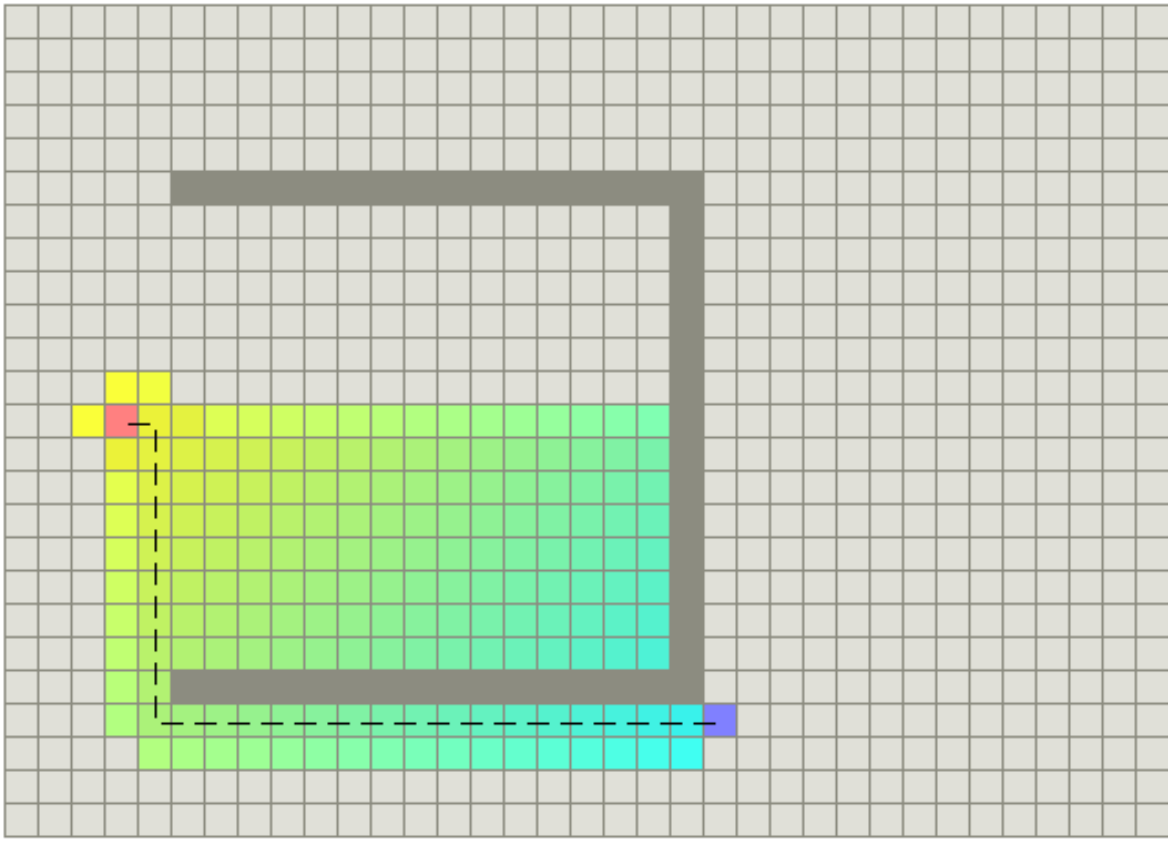
A* algoritam je jako popularan za rješavanje problema pronalaženje puta, jer je prilično fleksibilan i može se koristiti u širokom spektru konteksta. Tajna uspjeha algoritma je u tome što kombinira informacije koje koristi Dijkstrin algoritam (preferirajući čvorove koji su blizu početne točke) i informacije koje koristi Greedy Best-First-Search (preferirajući čvorove koji su blizu cilja).

Na slici 3.1. vidljiva je razlika između Dijkstrinog algoritma i Greedy Best First Search. Kao što smo i prije spomenuli, Dijkstrin algoritam je jako dobar u pronalaženju najkraćeg puta, ali je jako spor zato što pretražuje cijelu mapu za razliku od Greedy Best First Searcha. Na slici se može vidjeti kako put koji je pronašao BFS nije najkraći što da zaključiti da ovaj algoritam radi brže, ali ne i najbolje kada mape postanu kompleksnije.



Slika 3.1. Lijeva slika prikazuje izračun putanje s Dijkstrinim algoritmom, a desna slika s Greedy Best First Search.[6]

Kada se govori o A*, g(n) predstavlja trošak puta od početne točke do bilo kojeg čvora n, dok h(n) predstavlja heurističku procjenu troška od čvora n do cilja.



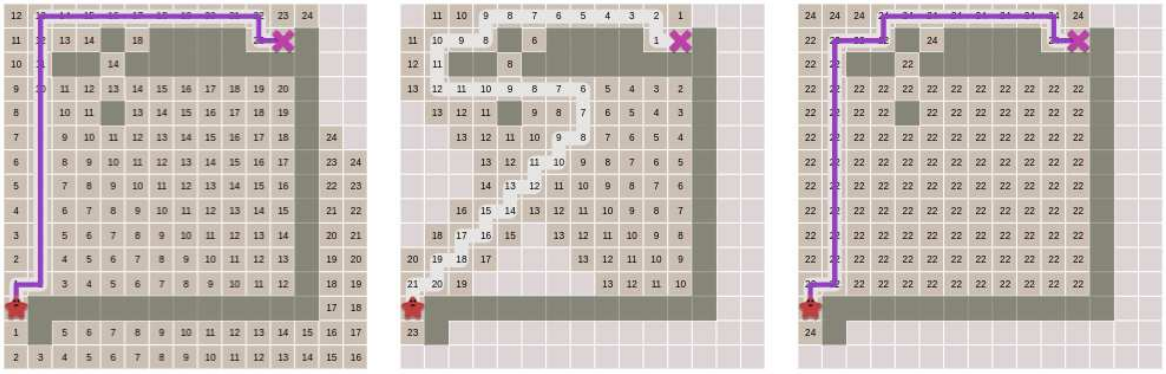
Slika 3.2. Pronalaženje putanje koristeći A* algoritam.[7]

Na slici 3.2., žuta (h) predstavlja čvorove koji su daleko od cilja, a tirkizna (g) predstavlja čvorove koji su daleko od početne točke. A* balansira ova dva faktora dok se kreće od početne točke ka cilju. Algoritam ispituje čvor n koji ima najniži:

$$f(n) = g(n) + h(n) \tag{3.1}$$

te najbolju šansu da dođe do cilja.

Slika 3.3. prikazuje razliku između sva tri algoritma: Dijkstrinog, BFS i A*. Kada Greedy Best-First Search pronađe pogrešan odgovor (duži put), A* pronalazi pravi put (najkraći) uz manje potrošenog vremena u istraživanju mape nego Dijkstrin algoritam.



Slika 3.3. Prvi kvadrat prikazuje Dijkstrin algotiam, drugi BFS i treći A* .[7]

Heuristika

Uspješnost algoritma jako ovisi o heuristici koja se upotrebljava. Izbor heuristike može drastično utjecati na performanse i učinkovitost algoritma. Dobra heuristika je ona koja pomaže algoritmu da pronađe najkraći put istražujući što manji broj čvorova. Svojstva heuristike uključuju:

- Prihvatljivost: Heuristika je prihvatljiva ako nikada ne precjenjuje trošak dosezanja cilja. Klasičan primjer prihvatljive heuristike je udaljenost u pravoj liniji na prostornoj karti. Precjenjivanje će dovesti do toga da A* pretražuje čvorove koji možda nisu 'najbolji' u smislu f vrijednosti.

- Dosljednost (ili Monotonost): Heuristika je dosljedna ako je procijenjeni trošak od trenutačnog čvora do cilja uvijek manji ili jednak procijenjenom trošku od bilo kojeg susjednog čvora plus trošak koraka od trenutačnog čvora do susjednog čvora.[8]

Heuristika se koristi kako bi "kontrolirala" A* algoritam

- Ako je h(n) jednak 0 tada samo g(n) ima ulogu, a A* postaje Dijkstrin algoritam, koji će zajamčeno pronaći najkraći put, ali se time gube prednosti algoritma.

- Ako je h(n) uvijek manji (ili jednak) trošku pomicanja od n do cilja, tada A* će zajamčeno pronaći najkraći put. Što je h(n) niži, to više čvorova A* proširuje, čineći ga sporijim.

- Ako je h(n) točno jednak trošku pomicanja od n do cilja, tada će A* slijediti samo najbolji put i nikada se neće proširiti negdje drugo, čineći ga vrlo brzim. Iako to

nije moguće u svim slučajevima, postoji mogućnost uspjeha u nekim posebnim okolnostima.

- Ako je h(n) ponekad veći od troška pomicanja od n do cilja, tada A* neće zajamčeno pronaći najkraći put, ali može biti brži u njegovom pronalasku.
- Na drugom ekstremu, ako je h(n) vrlo visok u odnosu na g(n), tada samo h(n) ima utjecaj, a A* postaje Greedy Best-First-Search.

Odabir heuristike ovisi o mapi prostora koji se pretražuje.

- Na kvadratnoj mreži koja omogućuje četiri smjera kretanja, koristite Manhattanova udaljenost (L1).
- Na kvadratnoj mreži koja omogućuje osam smjerova kretanja, koristite dijagonalnu udaljenost (L∞).
- Na kvadratnoj mreži koja omogućuje bilo koji smjer kretanja, možda ćete htjeti koristiti Euklidsku udaljenost (L2), no to nije nužno. Ako A* algoritam pronalazi putove na mreži, ali dopuštate kretanje izvan mreže, možda biste trebali razmotriti druge prikaze mape.
- Na heksagonalnoj mreži koja omogućuje 6 smjerova kretanja, koristite Manhattanova udaljenost prilagođenu heksagonalnim mrežama.

U ovom radu koristili smo Čebiševljeva udaljenost kao heurističku funkciju koja omogućuje kretanje u osam smjerova. Dana je formulom:

$$d(a, b) = \max |x_i - y_i| \tag{3.2}$$

Slika 3.4. prikazuje pseudokod A* algoritma. Prvo se inicijaliziraju dvije liste. "openList" lista koja sadrži čvorove kandidate za ispitivanje i "closedList" sadrži zatvorene (posjećene) čvorove. Na početku početni čvor se stavlja u "openList". Svaki čvor također čuva pokazivač na svoj roditeljski čvor kako bismo mogli odrediti kako je pronađen.

Postoji glavni ciklus koji ponavlja izvlačenje najboljeg čvora n (currentNode) iz skupa

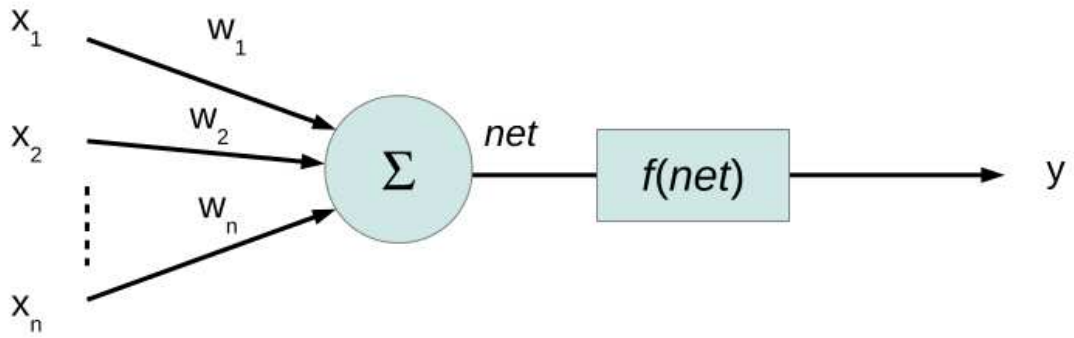
"openList" (čvor s najnižom f vrijednošću) i obrađuje ga. Ako je n cilj, tada je proces završen. Ako n nije cilj, n se uklanja iz "openList" i dodaje u "closedList". Zatim se ispituju njegovi susjedi k. Susjed koji je u "closedList" već je viđen, pa ga ne moramo ponovno gledati. Susjed koji je u "openList" već je planiran za pregled, pa ga sada ne moramo gledati. Inače, dodajemo ga u "openList", a njegov roditelj postavlja se na n. Trošak puta do k, g(k), bit će postavljen na g(n) + heuristika(n, k).

```
// A* (star) Pathfinding

// Initialize both open and closed list
let the openList equal empty list of nodes
let the closedList equal empty list of nodes

// Add the start node
put the startNode on the openList (leave it's f at zero)

// Loop until you find the end
while the openList is not empty

    // Get the current node
    let the currentNode equal the node with the least f value
    remove the currentNode from the openList
    add the currentNode to the closedList

    // Found the goal
    if currentNode is the goal
        Congratz! You've found the end! Backtrack to get path

    // Generate children
    let the children of the currentNode equal the adjacent nodes

    for each child in the children

        // Child is on the closedList
        if child is in the closedList
            continue to beginning of for loop

        // Create the f, g, and h values
        child.g = currentNode.g + distance between child and current
        child.h = distance from child to end
        child.f = child.g + child.h

        // Child is already in openList
        if child.position is in the openList's nodes positions
            if the child.g is higher than the openList node's g
                continue to beginning of for loop

        // Add the child to the openList
        add the child to the openList
```

Slika 3.4. Pseudokod A* algoritma.[9]

3.2. Neuronska meža koja koristi A* algoritam

Umjetna neuronska mreža (*eng.Artificial Neural Network; ANN*) je u osnovi masivni paralelni računalni modeli koji imitiraju funkciju ljudskog mozga. Sastoji se od velikog broja jednostavnih procesora povezanih težinskim vezama. Procesni čvorovi nazivaju se neuronima. Izlaz svakog čvora ovisi samo o informacijama koje su lokalno dostupne na čvoru, bilo da su pohranjene interno ili dolaze putem težinskih veza. Svaka jedinica prima ulaze od mnogih drugih čvorova i prenosi svoj izlaz na druge čvorove.[10]



Slika 3.5. Osnovni model neurona.[11]

Slika 3.5. predstavlja osnovni model neurona. Model se sastoji od ulaza(dendrita) x1 do xn, težina w1 do w2, tijela neurona koje računa ukupnu pobudu(označeno s *net*) te funkcije f(*net*). Težine w1 i w2 određuju u kojoj se mjeri svaki ulaz neurona pobuđuje dok prijenosna funkcija obrađuje pobudu i prosljeđuje ju na izlaz neurona. Jednadžba 3.3 računa ukupnu pobudu *net*, gdje θ predstavlja prag paljenja neurona.

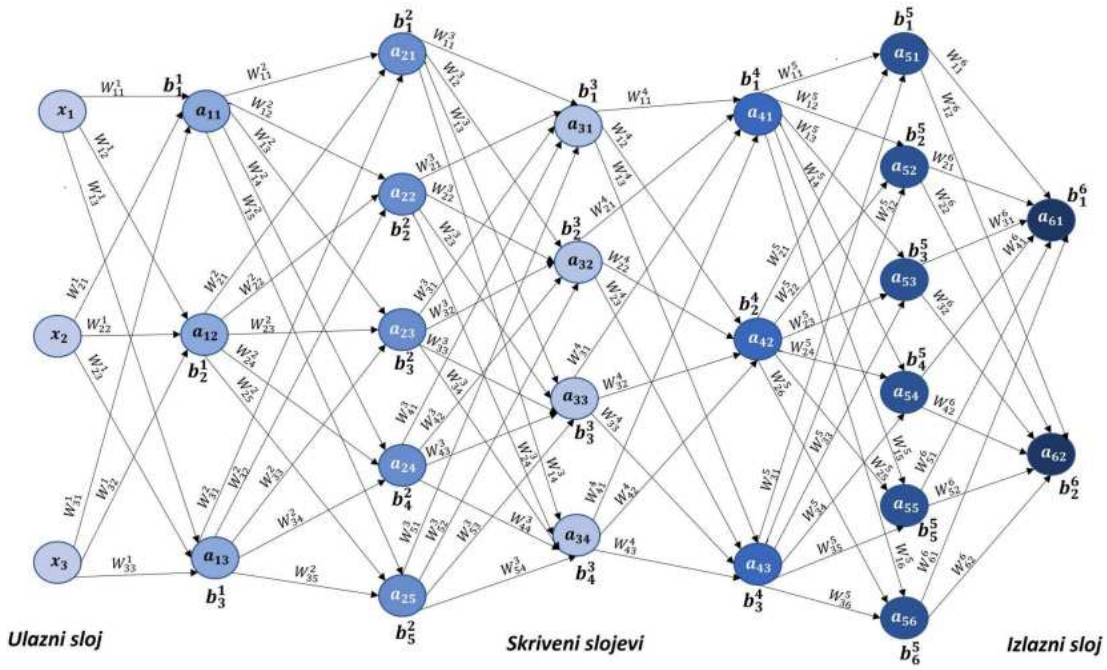
$$\text{net} = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n - \theta \tag{3.3}$$

Primjer jednostavne prijenosne funkcije je funkcija skoka prikazana jednadžbom 3.4

$$f(\text{net}) = \begin{cases} 0, & \text{net} \leq 0 \\ 1, & \text{inače} \end{cases} \tag{3.4}$$

Kada se neuronska mreža trenira, svi njezini težinski koeficijenti i pragovi inicijalno

se postavljaju na slučajne vrijednosti. Podatci za treniranje unose se u donji sloj (ulazni sloj) i prolaze kroz sljedeće slojeve, prolaze kroz razne operacije - poput skalarnog produkta nakon kojeg slijedi nelinearna funkcija, sve dok na kraju ne stignu radikalno transformirani, do izlaznog sloja. Tijekom treninga, težinski koeficijenti i pragovi stalno se prilagođavaju dok podaci za trening s istim oznakama dosljedno ne daju slične izlaze.[11]



Slika 3.6. Grafički prikaz umjetne neuronske mreže s pet slojeva.[12]

Broj parametarat potrebnih za treniranje računa se prema formuli 3.5 . Za neuronsku mrežu sa slike 3.6. potrebno je 109 parametara.

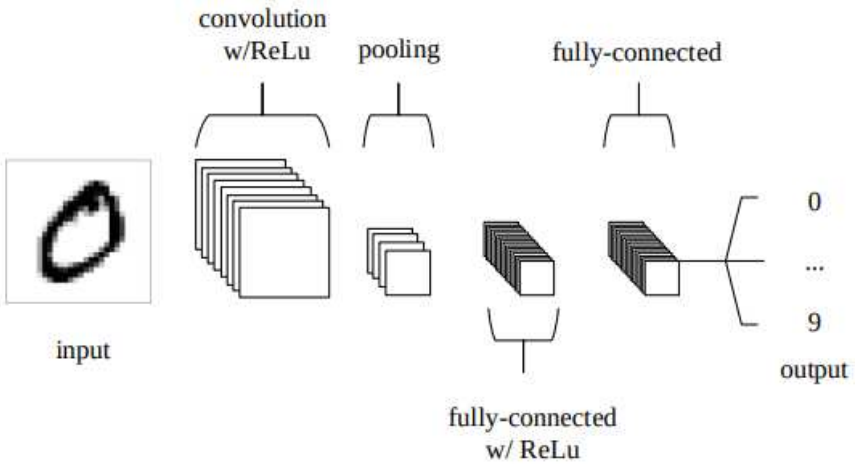
$$N_{\text{param}} = \sum_{k=1}^{L} N_k N_{k-1} + N_k, \tag{3.5}$$

Postoji mnogo vrsta neuronskih mreža koje se koriste za različite primjene, u ovom radu fokusirat ćemo se na konvolucijsku neuronsku mrežu. [12]

Konvolucijske neuronske mreže (CNN-ovi) - analogne su tradicionalnim umjetnim neuronskim mrežama (ANN-ovima) po tome što se sastoje od neurona koji se samostalno optimiziraju kroz učenje. Svaki neuron i dalje prima ulaz i provodi operaciju (poput skalarnog produkta nakon kojeg slijedi nelinearna funkcija) što je osnova brojnih

ANN-ova. Od početnih vektora slike do konačnog izlaza, cijela mreža i dalje izražava jedinstvenu perceptivnu funkcijsku vrijednost (težine). Posljednji sloj sadrži funkcije gubitka povezane s klasama, a svi uobičajeni trikovi i tehnike razvijeni za tradicionalne ANN-ove i dalje su primjenjivi.

Jedina značajna razlika između CNN-ova i tradicionalnih ANN-ova je ta što se CNN-ovi primarno koriste u području prepoznavanja uzoraka unutar slika. To nam omogućuje kodiranje značajki specifičnih za slike unutar same arhitekture, čineći mrežu prikladnijom za zadatke fokusirane na slike uz dodatno smanjenje broja parametara potrebnih za postavljanje modela. [13]



Slika 3.7. Jednostavna konvolucijska arhitektura koja se sastoji od pet slojeva. [13]

CNN-ovi se sastoje od tri sloja, a to su konvolucijski sloj, slojevi uzorkovanja (pooling) i potpuno povezani slojevi. Slika 3.7. predstavlja pojednostavljenu CNN arhitekturu za MNIST (velika baza podataka brojeva koja se često koristi za treniranje različitih sustava za obradu slika) klasifikaciju.

Osnovna funkcionalnost CNN-a predstavljenog na slici 3.7. može se rastaviti na četiri dijela:

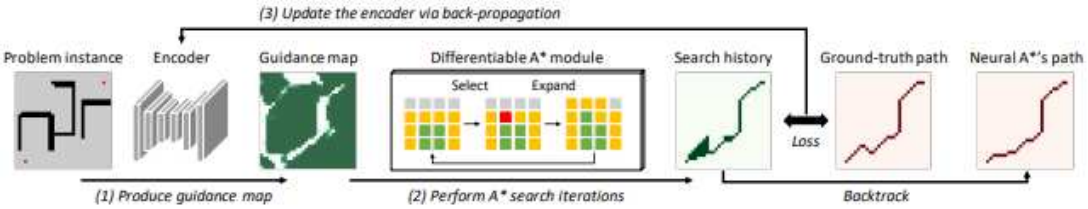
1. Kao i u drugim oblicima ANN-a, ulazni sloj će sadržavati vrijednosti piksela slike.

2. Konvolucijski sloj će odrediti izlaz neurona koji su povezani s lokalnim područjima ulaza kroz izračun skalarnih produkata između njihovih težina i regije povezane s ulaznim volumenom. Ispravljena linearna jedinica (*eng. rectified linear unit,*

ReLU) se koristi za primjenu aktivacijske funkcije 'elementwise', poput sigmoid funkcije, na izlaz aktivacije proizvedene od prethodnog sloja.

3. Sloj uzorkovanja (*eng.pooling layer*) će zatim jednostavno izvršiti smanjenje uzorka (downsampling) duž prostornih dimenzija danog ulaza, dodatno smanjujući broj parametara unutar te aktivacije.

4. Potpuno povezani slojevi (*eng.fully-connected layers*) zatim će obavljati iste zadatke kao u standardnim ANN-ima i pokušati generirati ocjene klasa iz aktivacija, koje će se koristiti za klasifikaciju. Također se sugerira da se ReLU može koristiti između tih slojeva, kako bi se poboljšala učinkovitost.[13]

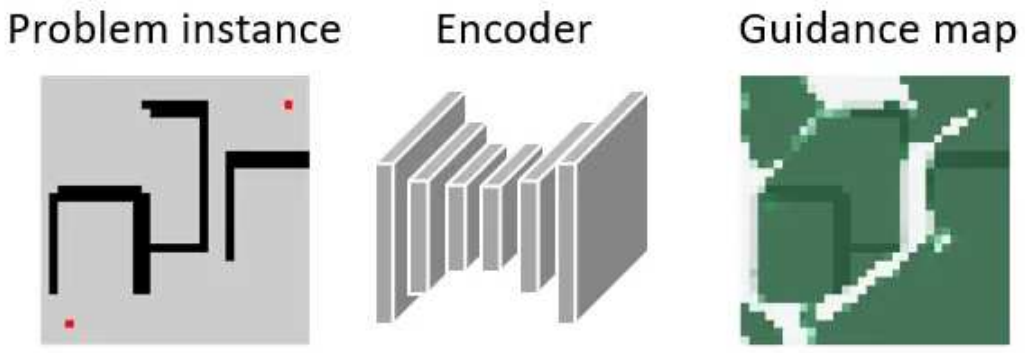


Slika 3.8. Trening od početka do kraja za planiranje putanje koristeći naš diferencijabilni A* modul.[14]

A* algoritam za pretraživanje se preformulira u diferencijabilan te ga se povezuje s konvolucijskim enkoderom kako bismo formirali neuronsku mrežu koja se može trenirati od početka do kraja. Neuronski A* rješava problem planiranja puta tako da kodira instancu problema u mapu smjernicu, a zatim izvodi diferencijabilni A* koristeći mapu smjernicu. Učeći kako uskladiti rezultate pretraživanja sa stvarnim najboljim putem. Kako bi reformulirali A* algoritam u diferencijabilni varijable iz algoritma na slici 3.4. moramo predstaviti kao matrice veličine zadane mape tako da svaka linija može biti izvršena s pomoću matričnih operacija.

Na slici 3.8. možemo vidjeti kako se neuronski A* planer sastoji od kombinacije potpuno konvolucijskog enkodera i diferencijabilni A* modula.

Ključna ideja neuronskog A* je izvođenje A* pretraživanja na mapama smjernicama (*eng.guidance maps*), koje su transformirane iz ulaznih instanci problema s pomoću enkodera, te pronalaženje najjeftinijih puteva s obzirom na smjernice troškova (*eng.guidance cost*). Slika 3.9. prikazuje izgled vodičke mape s obzirom na zadanu mapu. Vodičke mape



Slika 3.9. Svaku instancu problema (npr. sliku mape s početnim i ciljnim lokacijama) transformiramo u smjernicu mapu. Zelena i bijela boja na smjernicama mapama označavaju visoke i niske troškove. [14]

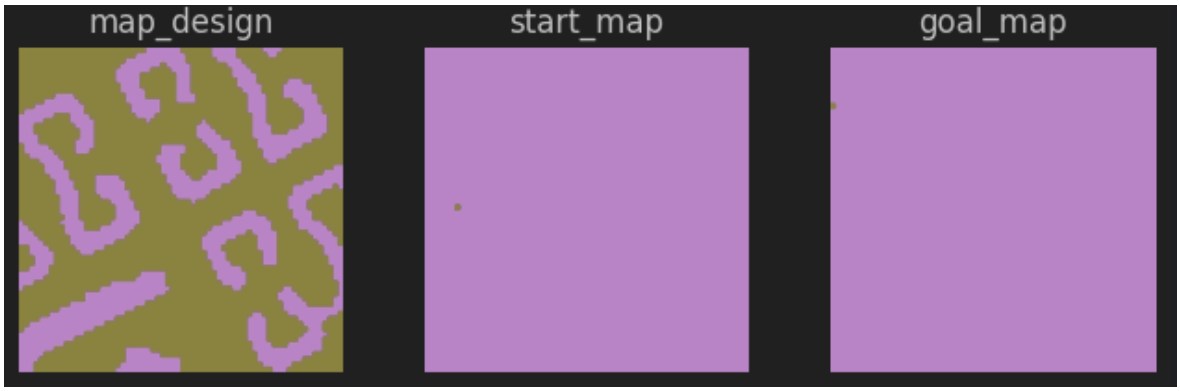
i njihove izračunati troškovi smjernica predstavljaju kako planer razumije problem. Treniranje se izvodi na sljedeći način:

1. Dano je stanje problema (npr. karta okoline označena početnim i ciljnim točkama), enkoder ga transformira u mapu sa skalarnim vrijednostima koja se naziva mapa smjernica.

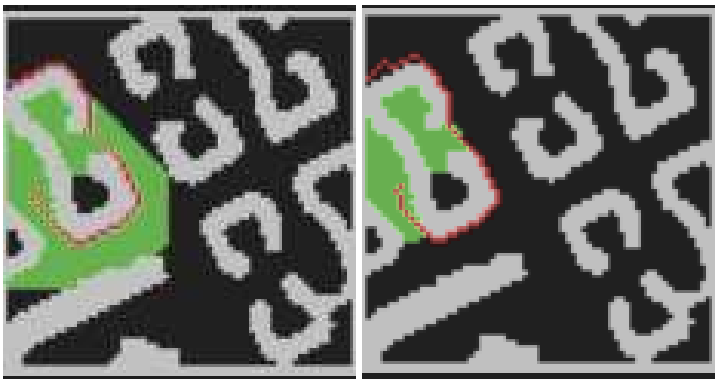
2. Diferencijabilni A* zatim izvodi pretraživanje s mapom smjernicom kako bi generirao povijest pretraživanja i rezultantni put.

3. Povijest pretraživanja se uspoređuje sa stvarnim putem ulazne instance kako bi se izračunao gubitak, koji se propagira unazad za treniranje enkodera.

Uloga diferencijabilnoj A* za trening je: podučiti enkoder kako proizvesti vodičke mape koje minimiziraju razliku između rezultirajućih povijesti pretraživanja (*eng.search history*) i referentnih (*eng.ground-truth*) putova. Enkoder tada uči prepoznavati vizualne naznake u ulazima koje su učinkovite za reproduciranje referentnih putanja. Konkretno, u scenariju pretraživanja najkraćeg puta na slikama 3.10. i 3.11., gdje su referentne putanje određene optimalnim planerima, enkoder se trenira kako bi učinkovito pronalazio gotovo optimalne putanje koristeći vizualne znakove, poput oblika slijepih ulica. U tom kontekstu, vodične mape proširuju ulazne mape kako bi se dalo prednost određenim čvorovima za istraživanje ili izbjegavanje, poboljšavajući kompromis između optimalnosti i učinkovitosti pretrage.[14]



Slika 3.10. Prikaz mape prostora, početne točke i ciljane točke. [14]



Slika 3.11. Prikaz izračuna putanje za A* (lijevo) i neuronskog A* (desno).

Treniranje neuronskog A*

Dizajn gubitka Diferencijabilni A* povezuje ulaz vodične mape Φ s izlazom pretrage tako da se gubitak na izlazu propagira unatrag prema Φ, a time i prema enkoderu. U ovom kontekstu, izlaz je zatvoreni popis C, binarna matrica koja akumulira sve pretražene čvorove V*.

Za procjenu točnosti koristimo srednji L1 gubitak između C i referentne (ground-truth) mape putanja $\bar{P}$, definiran kao:

$$\mathcal{L} = ||C - \bar{P}||_{1/|\mathcal{V}|}. \tag{3.6}$$

Ovaj gubitak nadzire proces odabira čvorova penalizirajući:

- Lažno negativne odabire, tj. čvorove koji su trebali biti uključeni u C kako bi se pronašla referentna putanja $\bar{P}$.
- Lažno pozitivne odabire, tj. čvorove koji su nepotrebno dodani u C, a nisu dio. $\bar{P}$.

Drugim riječima, ovaj gubitak potiče neuronski A* da:

- Pronalazi putanju koja je što bliža referentnoj.
- Smanji broj nepotrebno istraženih čvorova, čime povećava učinkovitost pretrage.

[14]

Dizajn enkodera Gubitak prikazan gore propagira se unatrag (*eng back-propagated*) kroz svaki korak pretrage u diferencijabilnoj A* modulu prema enkoderu. Ovdje očekujemo da enkoder nauči vizualne znakove u zadanim primjerima problema koji omogućuju precizno i učinkovito planiranje. Ti znakovi uključuju, na primjer, oblike slijepih ulica i obilaznica u binarnim mapama troškova ili boje i teksturalne uzorke prohodnih cesta u sirovim prirodnim slikama. Za ovu svrhu koristimo potpuno konvolucijsku mrežnu arhitekturu, koja može naučiti lokalne vizualne reprezentacije na izvornoj rezoluciji. Ulaz u enkoder dan je kao konkatenacija X i Vs (početni čvor) + Vg (čvor cilj). Na ovaj način, ekstrakcija tih vizualnih znakova pravilno je uvjetovana početnim i ciljanim pozicijama.[14]

Omogućavanje treniranja u mini-serijama Potpuni algoritam neuronskog A* prikazan je na slici 3.12. Kako bi se ubrzalo treniranje, potrebno je obraditi više primjera problema odjednom u mini-seriji.

Dataset Skup podataka za planiranje kretanja (MP): Kolekcija osam vrsta okruženja u mreži s karakterističnim oblicima prepreka. Svaka grupa okruženja sastoji se od 800 mapa za trening, 100 mapa za validaciju i 100 mapa za testiranje, s istim vrstama prepreka postavljenim u različitim rasporedima.

Algorithm 2 Neural A* Search

Input: Problem instances $\{Q^{(i)} = (X^{(i)}, v_{\mathrm{s}}^{(i)}, v_{\mathrm{g}}^{(i)}) \mid i = 1, \ldots, b\}$ in a mini-batch of size b.

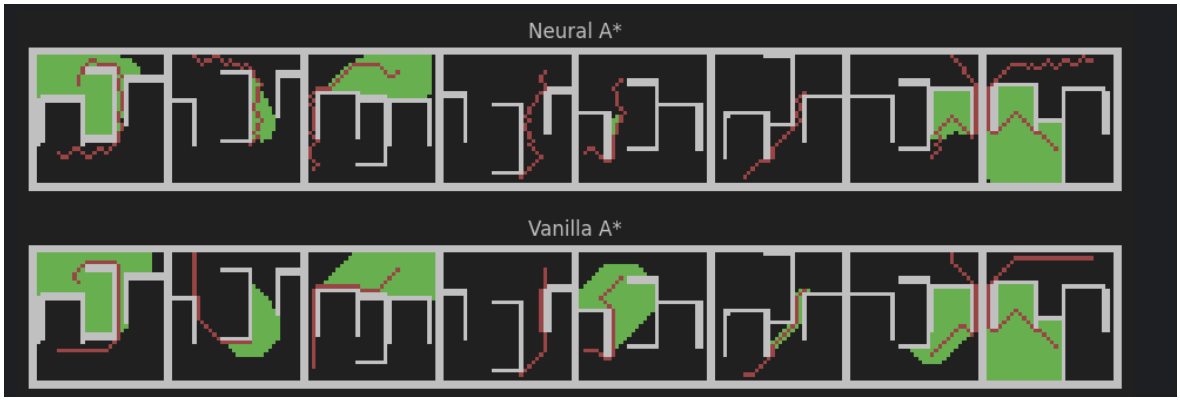
Output: Closed-list matrices $\{C^{(i)} \mid i = 1, \ldots, b\}$ and solution paths $\{P^{(i)} \mid i = 1, \ldots, b\}$.

for all $i = 1, \ldots, b$ **do in parallel**
  Compute $V_{\mathrm{s}}^{(i)}, V_{\mathrm{g}}^{(i)}$ from $v_{\mathrm{s}}^{(i)}, v_{\mathrm{g}}^{(i)}$.
  Compute $\Phi^{(i)}$ from $X^{(i)}, V_{\mathrm{s}}^{(i)}, V_{\mathrm{g}}^{(i)}$ by the encoder.
  Initialize $O^{(i)} \leftarrow V_{\mathrm{s}}^{(i)}$, $C^{(i)} \leftarrow \mathbf{0}$, $G^{(i)} \leftarrow \mathbf{0}$.
  Initialize $\texttt{Parent}^{(i)}(v_{\mathrm{s}}^{(i)}) \leftarrow \emptyset$.
end for
repeat
  for all $i = 1, \ldots, b$ **do in parallel**
    Select $V^{*(i)}$ based on Eq. (3).
    Compute $\eta^{(i)} = 1 - \langle V_{\mathrm{g}}^{(i)}, V^{*(i)} \rangle$.
    Update $O^{(i)}$ and $C^{(i)}$ based on Eq. (8).
    Compute $V_{\mathrm{nbr}}^{(i)}$ based on Eq. (4).
    Update $O^{(i)} \leftarrow O^{(i)} + V_{\mathrm{nbr}}^{(i)}$.
    Update $G^{(i)}$ based on Eq. (5) and Eq. (6).
    Update $\texttt{Parent}^{(i)}$ based on Algorithm 1-L6,7.
  end for
until $\eta^{(i)} = 0$ for $i = 1, \ldots, b$
for all $i = 1, \ldots, b$ **do in parallel**
  $P^{(i)} \leftarrow \texttt{Backtrack}(\texttt{Parent}^{(i)}, v_{\mathrm{g}}^{(i)})$.
end for

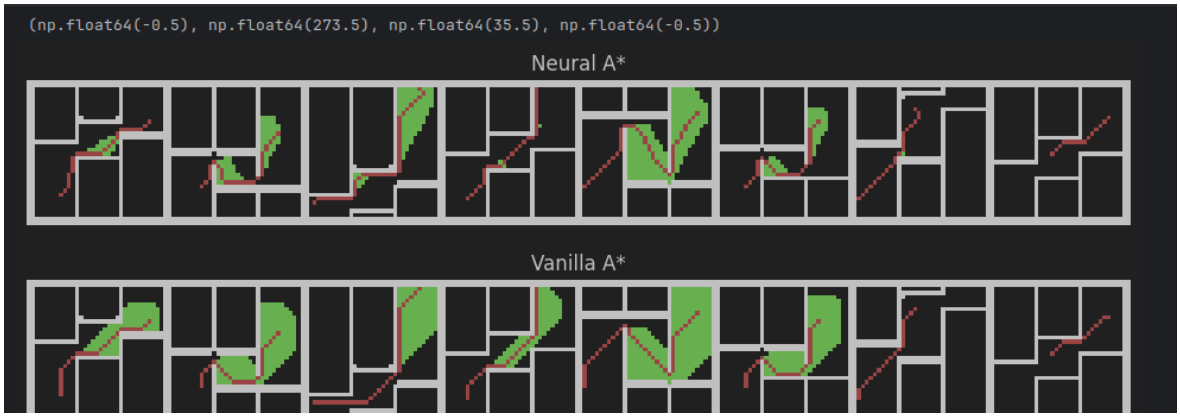
Slika 3.12. Neuronski A* algoritam. [14]

4. Rezultati i rasprava

Zadatak ovog rada je bio napraviti usporedbu A* algoritma opisanog u poglavlju 3.1. te neuronske mreže koja koristi A* opisane u poglavlju 3.2. U prethodnim poglavljima opisane su karakteristike pojedinog algoritma te su dani pseudo kodovi. Na slikama zelena boja prikazuje prostor mape koji je algoritam istražio, dok crvena označava odabranu putanju.



Slika 4.1. Broj epoha - 50



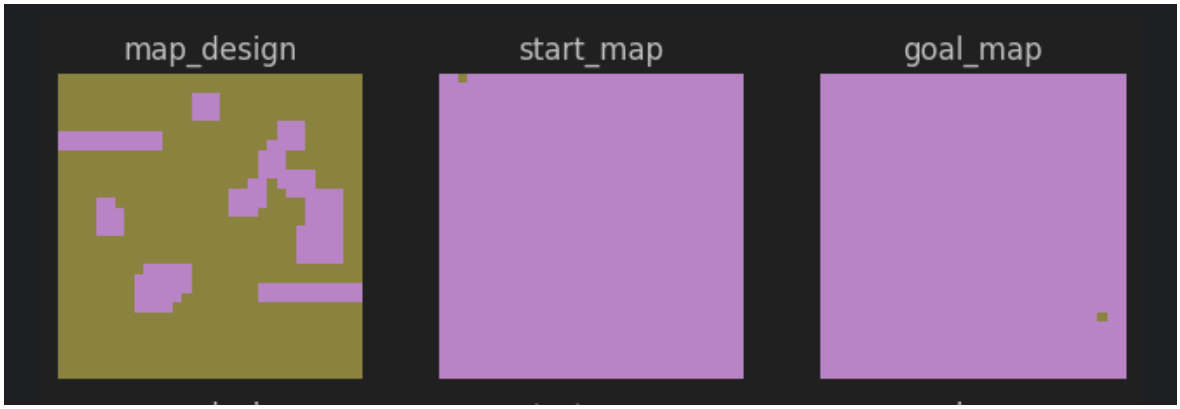
Slika 4.2. Broj epoha - 100

Slika 4.3. Usporedba rezultata nakon različitog broja epoha.

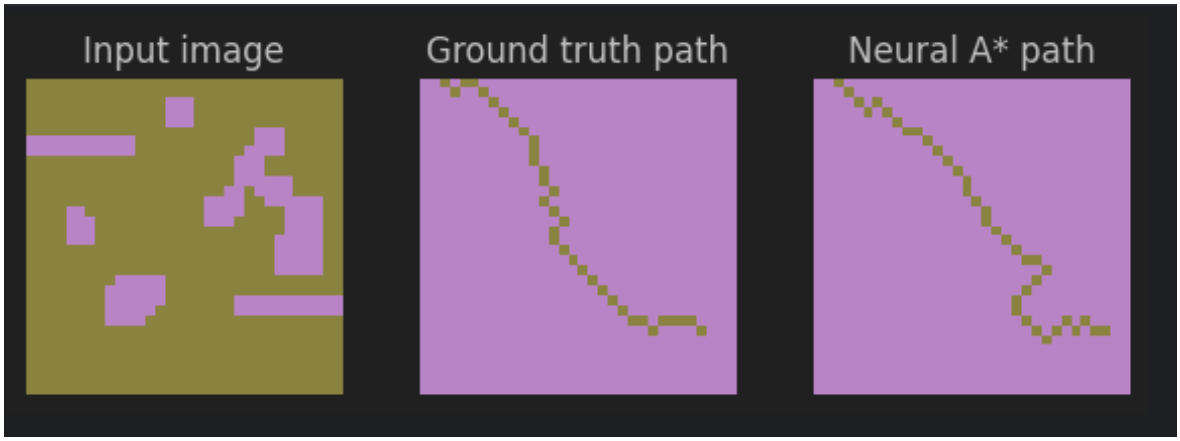
Slike 4.1., 4.2. prikazuju kako broj epoha utječe na točnost algoritma neuronskog A*.

Kao što se može vidjeti sa slika, broj epoha nije značajno utjecao na ishod algoritma.

Slika 4.4. uz izgled mape prikazuje zadanu početnu točku i ciljanju točku, dok slika 4.5. prikazuje izgled referentne putanje te putanje dobivene algoritmom neuronskog A*. Vidljivo je da putanja dobivena algoritmom ne odstupa previše od referentne putanje.



Slika 4.4. Prva slika prikazuje mapu, druge dvije redom početak i cilj.



Slika 4.5. Prikaz mape, referentne putanje te putanje dobivene neuronskim A*-om.

Sa slike 4.6. i iz tablica Tablice 4.1. i 4.2. možemo vidjeti usporedbu neuronskog A* i A* algoritma. Vidljivo je da neuronski A* uz manje pretraživanje prostora (područje označeno zelenom bojom) pronađe točnu putanju (označeno crvenom bojom), no zbog toga što se u neuronskom A* pokušava optimizirati potrošnja i vrijeme pretraživanja ponekad dobivamo malo manje "idealne" putanje nego u samom A*.

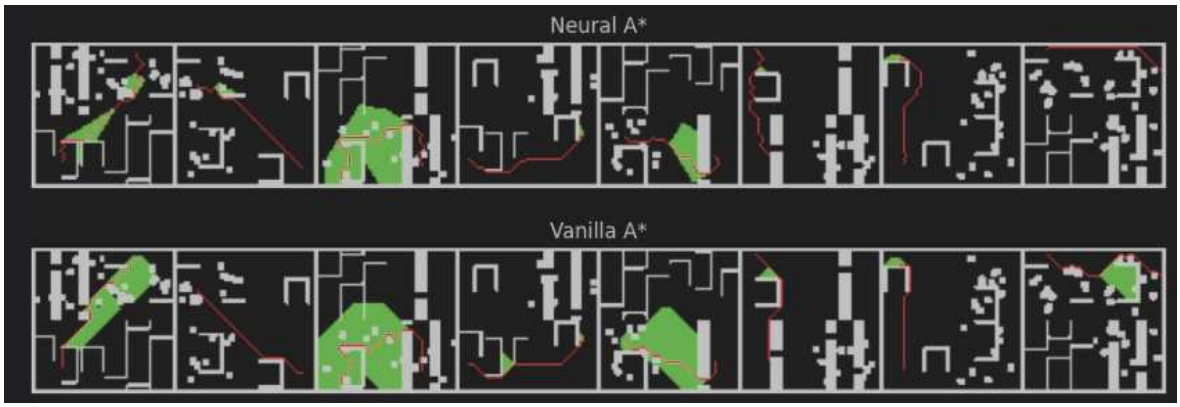
Najveća razliku između ova dva algoritma je u tome što neuronski A* ima mogućnost pretraživanja putanja na slikama igrica 4.7., slikama iz stvarnog svijeta. Za razliku od prethodnog eksperimenta, gdje su lokacije prepreka bile eksplicitno zadane, model sada mora naučiti vizualne reprezentacije prepreka kako bi ih izbjegao prilikom odabira čvorova.

Slika 4.6.	Neuronski A* duljina putanje	Neuronski A* pretraženo područje
0	65	233.0
1	51	69.0
2	72	909.0
3	56	62.0
4	51	264.0
5	50	56.0
6	55	74.0
7	54	54.0

Tablica 4.1. Neuronski A*, podaci o duljini putanja i pretraženom području izraženo u pikselima na slici.

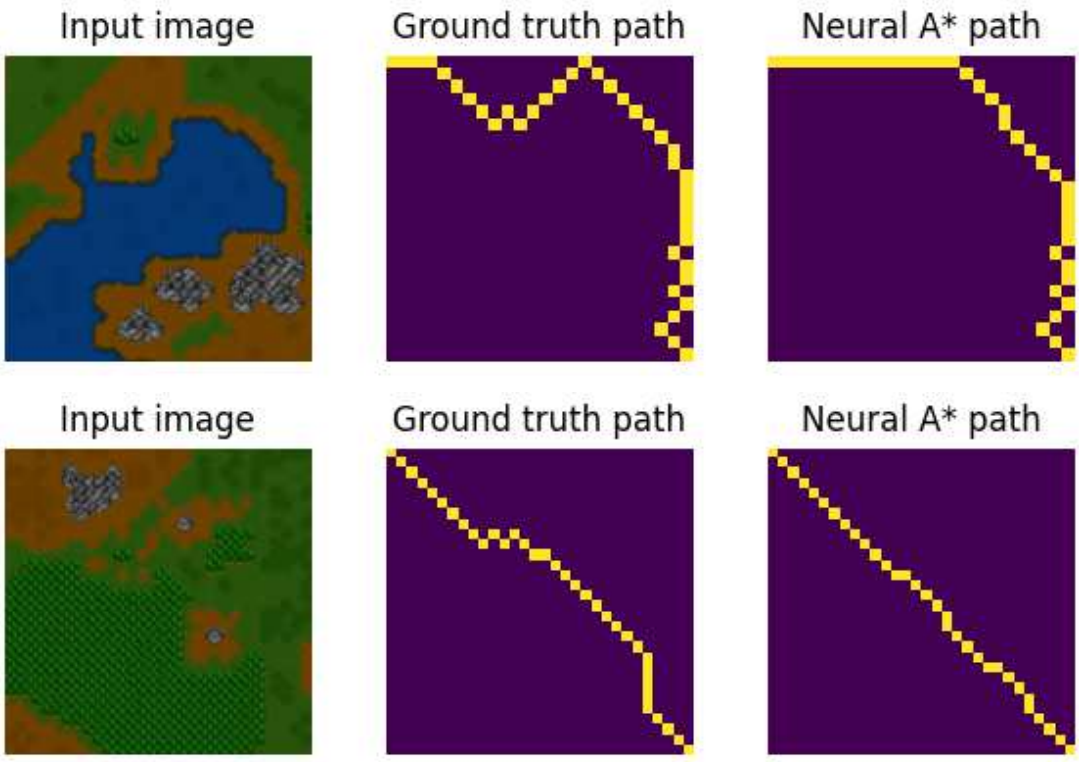
Slika 4.6.	A* duljina putanje	A* pretraženo područje
0	55	535.0
1	50	50.0
2	71	1219.0
3	56	91.0
4	48	766.0
5	50	70.0
6	55	81.0
7	54	231.0

Tablica 4.2. A*, podaci o duljini putanja i pretraženom području izraženo u pikselima na slici.

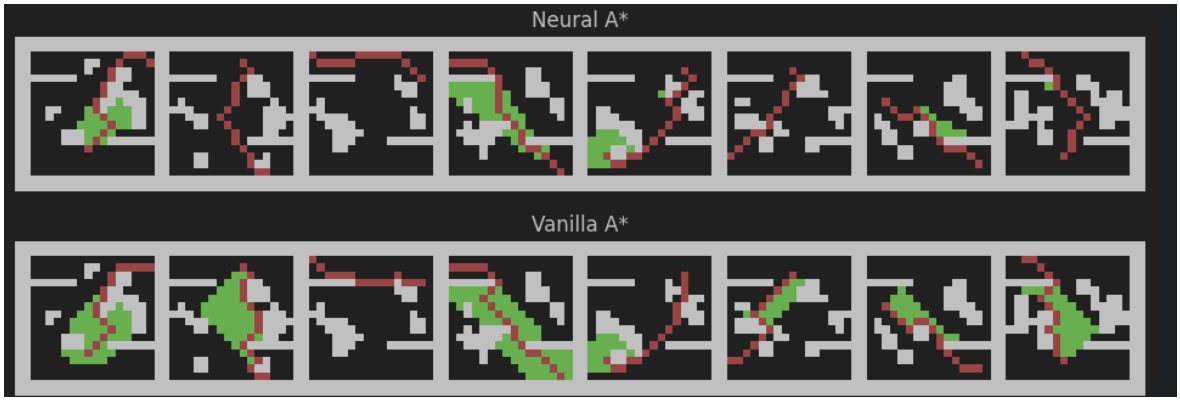


Slika 4.6. Usporedba neuronskog A* i A*.

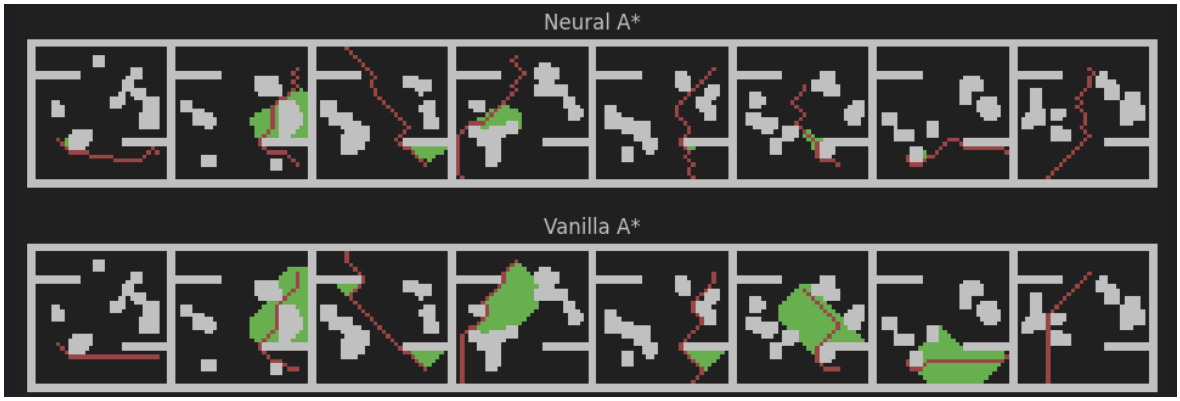
Slika 4.11. prikazuje izračun putanje u različitim rezolucijama. Kao što se vidi iz slika algoritam neuronskog A* bez problema pronalazi putanju uz manje istraživanje prostora nego A*. Mana algoritma je to što povećanjem rezolucije brzina treniranja se drastično povećava. Na primjer za treniranje neuronske mreže s rezolucijom slike od 16 piksela 4.8. bilo je potrebno svega par minuta, dok je za treniranje slike od 64 piksela 4.10. bilo potrebno par sati što iziskuje potrebu za znatno jačim računalom.



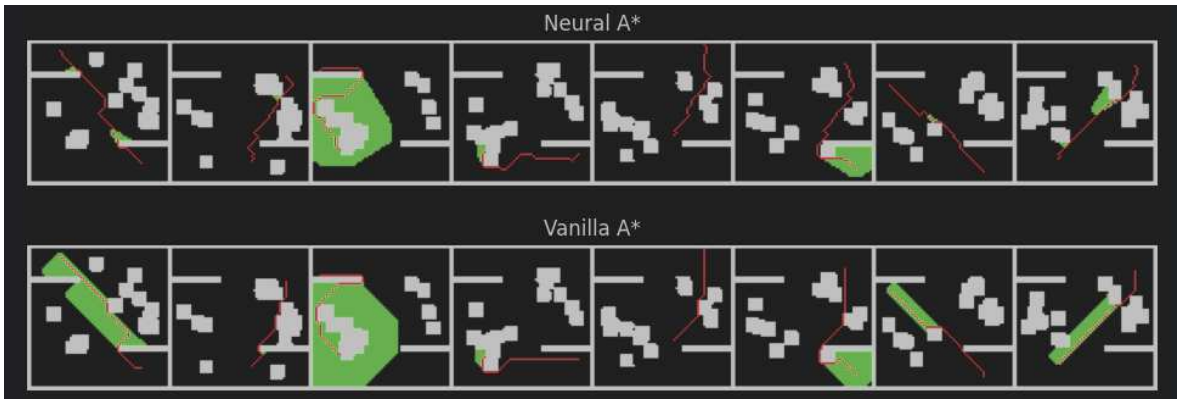
Slika 4.7. Pretraživanje putanje na mapi od igrice.



Slika 4.8. Slike s rezolucijom 16 piksela



Slika 4.9. Slike s rezolucijom 32 piksela



Slika 4.10. Slike s rezolucijom 50 piksela

Slika 4.11. Usporedba različitih rezolucija slika.

5. Zaključak

U ovom radu istražili smo i usporedili tradicionalni A* algoritam i neuronski A* algoritam za planiranje putanje. A* algoritam koji je temeljen na Dijkstrinom algoritmu i strategiji pretrage najboljeg prvog je jako učinkovit te uz puno manje pretraživanja nego Dijkstrin pronalazi put. Međutim, može biti računalno zahtjevan jer istražuje veliki broj čvorova, posebno u složenim okruženjima.

S druge strane, neuronski A* algoritam koristi snagu neuronskih mreža za pretraživanje, značajno smanjujući broj pretraženih čvorova, te mu je cilj poboljšao kompromis između optimalnosti i učinkovitosti, što rezultira bržem pronalaženju puta. Neuronski A* algoritam pokazuje svoju snagu u rješavanju složenijih okruženja, poput mapa igara i stvarnih slika, gdje tradicionalni A* može imati poteškoća.

Provedeni eksperimenti pokazuju da, iako neuronski A* algoritam možda neće uvijek pronaći apsolutno najkraću putanju, pruža ravnotežu između učinkovitosti i kvalitete putanje.

Sveukupno, integracija neuronskih mreža s tradicionalnim algoritmima za planiranje putanje poput diferencijabilni A* algoritam otvara nove mogućnosti za učinkovitu navigaciju u složenim okruženjima.

Literatura

[1] H.-y. Zhang, W.-m. Lin, i A.-x. Chen, "Path planning for the mobile robot: A review", *Symmetry*, sv. 10, br. 10, 2018. https://doi.org/10.3390/sym10100450

[2] A. Gasparetto, P. Boscariol, A. Lanzutti, i R. Vidoni, *Path Planning and Trajectory Planning Algorithms: A General Overview*. Cham: Springer International Publishing, 2015., str. 3–27. https://doi.org/10.1007/978-3-319-14705-5_1

[3] L. Qiong, C. Xudong, H. Jizhuang, i M. Ruihao, "Research on robot path planning method based on tangent intersection method", u *2020 International Conference on Information Science, Parallel and Distributed Systems (ISPDS)*, 2020., str. 272–276. https://doi.org/10.1109/ISPDS51347.2020.00063

[4] Nikolaus Correll, https://eng.libretexts.org/Bookshelves/Mechanical_Engineering/Introduction_to_Autonomous_Robots_(Correll)/04%3A_Path_Planning/4.02%3A_Path-Planning_Algorithms.

[5] geeksforgeeks, https://www.geeksforgeeks.org/difference-between-dijkstras-algorithm-and-a-search-algorithm/.

[6] Astar article, https://www.redblobgames.com/pathfinding/a-star/introduction.html.

[7] stanford, https://theory.stanford.edu/ amitp/GameProgramming/AStarComparison.html/.

[8] geeksforgeeks, https://www.geeksforgeeks.org/a-algorithm-and-its-heuristic-search-strategy-in-artificial-intelligence/.

[9] https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2.

[10] A. Dongare, R. Kharde, A. D. Kachare *et al.*, “Introduction to artificial neural network”, *International Journal of Engineering and Innovative Technology (IJEIT)*, sv. 2, br. 1, str. 189–194, 2012.

[11] M. Čupić, B. D. Bašić, i M. Golub, *Neizrazito, evolucijsko i neuroračunarstvo.* Marko Čupić, Bojana Dalbelo Bašić, Marko Golub, 2013.

[12] M. Hajba M, Pejović G, “Neuronske mreže - aproksimatori funkcija.” 2023.

[13] K. O’Shea, “An introduction to convolutional neural networks”, *arXiv preprint arXiv:1511.08458*, 2015.

[14] R. Yonetani, T. Taniai, M. Barekatain, M. Nishimura, i A. Kanezaki, “Path planning using neural a* search”, *CoRR*, sv. abs/2009.07476, 2020. [Mrežno]. Adresa: https://arxiv.org/abs/2009.07476

Sažetak

U ovom radu istražili smo i usporedili tradicionalni A* algoritam i neuronski A* algoritam za planiranje putanje. A* algoritam, poznat po svojoj potpunosti i optimalnosti, jamči pronalazak najkraće putanje ako ona postoji, ali može biti računalno zahtjevan zbog velikog broja istraženih čvorova. Neuronski A* algoritam koristi diferencijabilni A* algoritam i neuronsku mrežu za vođenje procesa pretraživanja, smanjujući broj istraženih čvorova i ubrzavajući pronalaženje putanje uz malu kompromisnost u optimalnosti. Eksperimenti pokazuju ksko neuronski A* algoritam pruža praktičnu ravnotežu između učinkovitosti i kvalitete putanje. Integracija neuronskih mreža s tradicionalnim algoritmima otvara nove mogućnosti za učinkovitu navigaciju u složenim okruženjima.

Ključne riječi: A*; neuronski A*; pretraživanje putanje

Abstract

In this paper, we explored and compared the traditional A* algorithm and the Neural A* algorithm for path planning. The A* algorithm, known for its completeness and optimality, guarantees finding the shortest path if one exists, but can be computationally demanding due to the large number of explored nodes. The neural A* algorithm leverages a differentiable A* algorithm and a neural network to guide the search process, reducing the number of explored nodes and accelerating path finding while making a minor trade-off in optimality. Experiments show that the neural A* algorithm provides a practical balance between efficiency and path quality. The integration of neural networks with traditional algorithms opens new possibilities for efficient navigation in complex environments.

Keywords: A*; Neural A*; path planning