Sveučilište u Splitu
Prirodoslovno-matematički fakultet

Uvod u umjetnu inteligenciju

# Prilagođena bidirekcijska pretraga labirinta

**Andrea Mikelić i Paula Brčić**

Split, veljača 2020.

# Sadržaj

# 1 Uvod

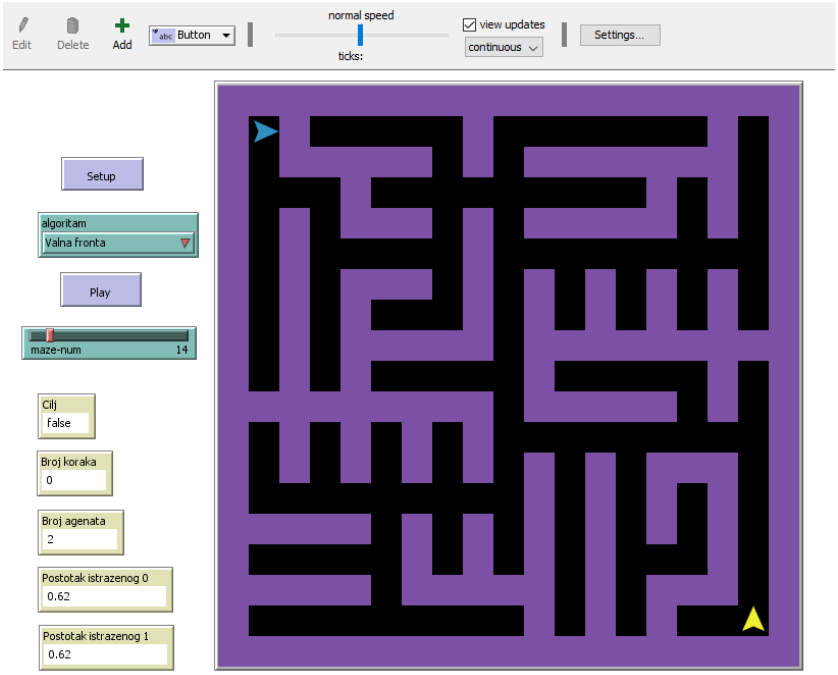
Problem s kojim smo započeli ovaj projekt je međusobno pronalaženje dvaju agenata u labirintu. Ideja je pokazati da algoritme kojima je primarna upotreba u grafovima/labirintima s fiksnim ciljem prilagodimo na dvostranu pretragu s pomičnim ciljem. Pritom ćemo koristiti ideje bidirekcijske pretrage.

Koristit ćemo NetLogo okruženje za implementaciju odabranih algoritama, a pomoću BehaviourSpace-a testirati na 100 labirinata. Tablicu dobivenih rezultata analizirat ćemo kroz alate za vizualizaciju u programu Orange. Pritom ćemo usporedbom parametara dati zaključak o učinkovitosti pojedinog algoritma.

# 2 Bidirekcijska pretraga

U matematičkim modelima problema, česta je upotreba grafova. Stoga ne čudi da se toliko truda ulaže u njihovo pretraživanje. Jedan od pristupa je bidirekcijska pretraga. Ona se temelji na istovremenoj pretrazi od početka do cilja i od cilja do početka. Bidirekcijska pretraga zamjenjuje jednostranu, podjelom grafa na dva podgrafa. U mnogo slučajeva to smanjuje potrebno vrijeme pretrage. Pritom je važno da su početna i krajnja točka jednoznačno određene.

Problem kojim smo se odlučili baviti u ovom projektu je međusobno traženje dvaju agenata u labirintu. Pritom je važno napomenuti da su početne pozicije agenata fiksne. Također, obzirom da se u svakom koraku agenti pomiču, efektivno imamo dva pomična cilja. Algoritmi koje smo koristiti u ovom projektu nisu originalno bidirekcijski, nego smo ih u implementaciji prilagodili konceptima bidirekcijske pretrage. Neki od algoritama koriste heuristiku za ciljano traženje, dok drugi pretražuju prostor bez poznavanja položaja svog cilja.




Slika 1: Korisničko sučelje

# 3 Implementacija

## 3.1 NetLogo

Za rješavanje spomenutog problema odabrali smo NetLogo okruženje (Slika 1) jer nam se svidio vizualni prikaz pomicanja agenata. Implementirali smo različite algoritme i testirali ih na 100 učitanih labirinata. Kako bi osigurali da su labirinti slične složenosti, koristili smo labirinte s predavanja. Također je važno da su početne pozicije agenata uvijek iste (lijevi gornji i desni donji kut). Korisnik putem sučelja ima mogućnost izbora algoritma pretrage i labirinta. Pritom pri odabiru algoritma korisnik ima dvije opcije – pomicati oba agenta po istom algoritmu ili u nekoj od ponuđenih kombinacija.

## 3.2 Algoritmi

Pri odabiru algoritama važno nam je bilo da se glavni agenti kontinuirano pomiču. Tako na primjer standardnu pretragu po širini nismo uzeli u obzir jer podrazumijeva vraćanje agenta na početnu poziciju. Još manje prikladan algoritam bila je i „hill climbing“ pretraga u kojoj agent u nekom trenutku nestane i pojavi se na nekom prethodnom čvoru. Također, višeagentska pretraga ovdje nam podrazumijeva korištenje „hatch“ agenata isključivo za računanje duljine dijela puta.

### Valna fronta

Klasični algoritam na početku izvođenja pridružuje vrijednosti poljima od cilja prema početnoj točki i za idući korak odlučuje se za susjedno polje s najmanjim brojem. U našoj implementaciji pridruživanje vrijednosti poljima izvršava se u svakom koraku, pri čemu se korak sastoji od jednog pomaka svakog agenta. Naglasimo još da je pretraga na obje strane jednoagentska i informirana.

### Pohlepni algoritam

Kada se agent nalazi na čvoru, bira onaj smjer koji mu osigurava najmanje koraka do sljedećeg čvora. Ako nije na čvoru, pomiče se naprijed. Naglasimo da je agentu dopušteno naći se na istom polju više puta, ali ne direktnim pomakom unatrag. Jedina iznimka ovom pravilu je u slučaju slijepe ulice („deadend“). Ova pretraga je višeagentska i neinformirana, stoga očekujemo da neće nužno pronaći rješenje. Pritom „hatch“ agenti se koriste iskljucivo za racunanje duljine puta a ne za pretragu prostora. Odnosno, ako „hatch“ agent pronađe agenta koji se traži, to ne smatramo rješenjem dok se međusobno ne pronađu početni agenti.

### Random

U svrhu usporedbe s „pametnijim“ algoritmima, izabrali smo koristiti i random algoritam. Nasumičnost se odnosi na odluke o smjeru nastavka puta kad se agent nađe na čvoru. Koristili smo funkciju „one-of“ koja je iz liste smjerova odabirala nasumični smjer. Ova je pretraga očito jednoagentska i neinformirana.

#### A*

Ovaj algoritam temelji se na heurističkoj funkciji koja zbraja broj koraka od trenutne pozicije do sljedećeg čvora i udaljenost od tog čvora do trenutnog cilja. Smjer se odabire minimizacijom heurističke funkcije. Pri odabiru heuristike bilo nam je svejedno, pa smo odlučili koristiti „Manhattan“ udaljenost. A* algoritam je višeagentska i informirana pretraga.

#### Dijkstraish

Ovaj algoritam sami smo osmislili kao proširenje pohlepnog, a zatim primijetili sličnost s algoritmom „Dijkstra“. Kad se agent nalazi na čvoru, ponaša se kao pohlepni, ali njegovi „hatch“ agenti stvaraju drugu generaciju koja ide do sljedećih čvorova. Smjer kojim će nastaviti početni agent odabire se minimizacijom zbroja puta „hatch“ agenata prve i druge generacije. Za razliku od pravog „Dijkstra“ algoritma, naša verzija uzima u obzir broj koraka samo do čvorova na „dubini“ 2. Također, naš algoritam je primjenjen na labirintu koji je manje povezanosti nego oni na kojima bi se češće primjenjivala „Dijkstra“. Ova je pretraga također višeagentska, ali neinformirana.

## 3.3 Kombinacije

Obzirom da očekujemo da se neće u svim slučajevima početni agenti pronaći, odlučili smo kombinirati algoritme. Točnije, jedan agent pomicat će se po pravilima prvog, a drugi po pravilima drugog algoritma. Time ćemo istražiti našu pretpostavku da će se postotak pronalaska agenata povećati. Pri kombiniranju smo izostavili algoritam valne fronte jer je, kao što se može vidjeti u analizi pojedinih algoritama, daleko najbolja u pronalaženju rješenja. Naglasimo da nema smisla provjeravati obrnute kombinacije algoritama (npr. Pohlepni-A* i A*-Pohlepni) jer je očito da ćemo dobiti analogne rezultate.

Kombinacije koje su testirane:

- Pohlepni-random
- Pohlepni-A*
- Pohlepni-Dijkstraish
- Random-A*
- Random-Dijkstraish
- A*-Dijkstraish

# 4 Analiza

## 4.1 Behaviour space

Koristan alat za analizu ponašanja implementiranih algoritama je BehaviourSpace. Njime imamo mogućnost pokrenuti naš model zadani broj puta i prikupiti željene podatke.

U našem slučaju, kod se izvrši za sve moguće kombinacije algoritma i labirinta. Obzirom da imamo 5 različitih algoritama i 100 korištenih labirinata, to čini ukupno 500 pokretanja. Analogno, za 6 kombinacija algoritama, 600 pokretanja.

Parametri koje smo htjeli analizirati su:

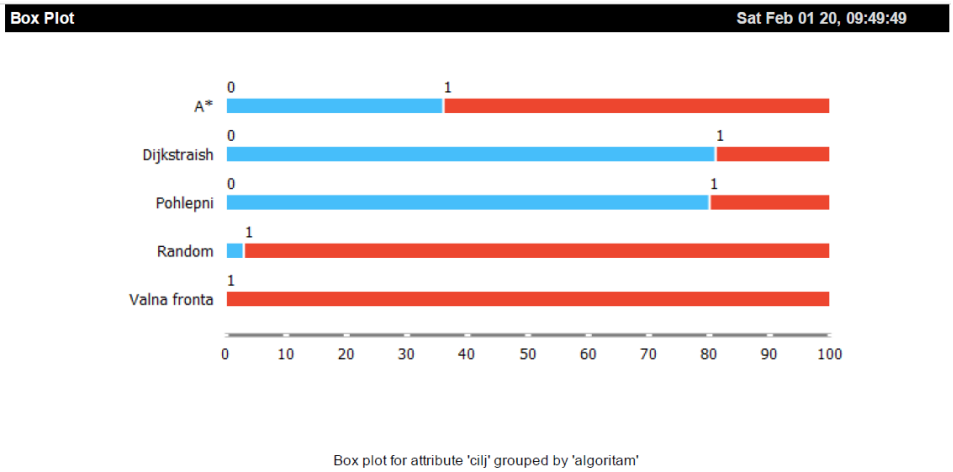
- cilj (jesu li se agenti pronašli)
- broj koraka (isključivo od glavnih agenata)
- broj agenata (2 početna + „hatch“ agenti)
- postotak istrazenog prostora (pojedinačno za glavne agente)

## 4.2 Orange

### 4.2.1 Analiza algoritama

#### Cilj

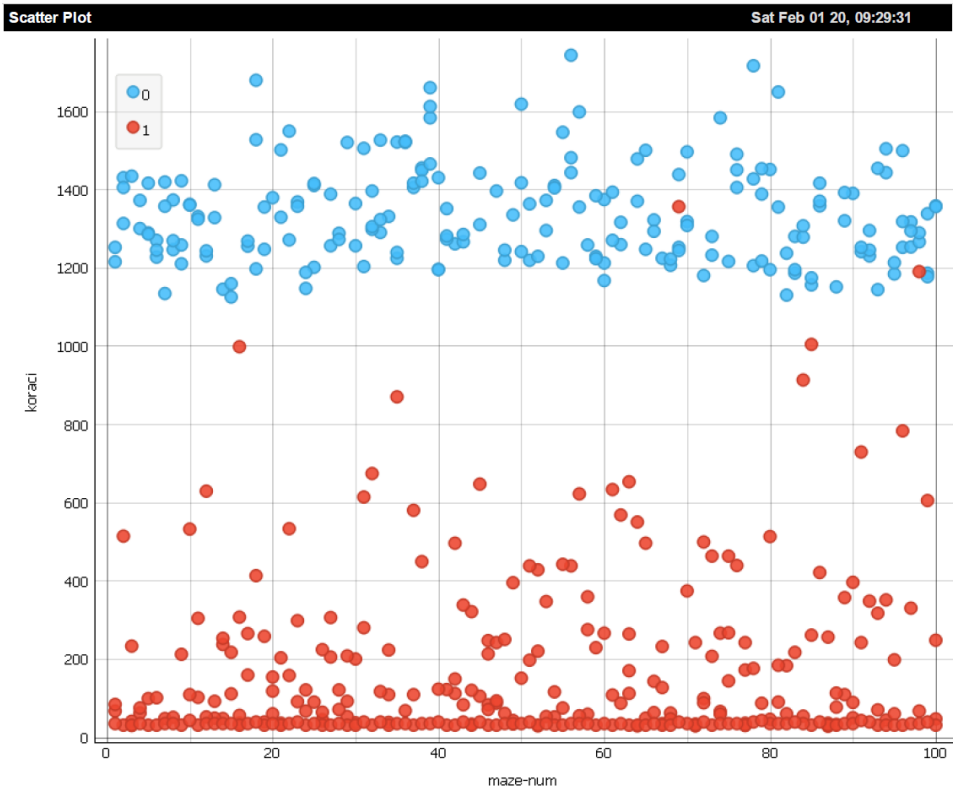
Promotrimo prvo donju sliku (Slika 2). Valna fronta jedina uvijek pronalazi rješenje, a random u iznenađujućoj većini slučajeva također (pretpostavljamo zbog povezanosti labirinata). Pohlepni i dijkstraish su u manje od 20% slučajeva pronašli rješenje. A* algoritam je puno bolji od njih, ali u skoro 40% slučajeva nije pronašao rješenje.



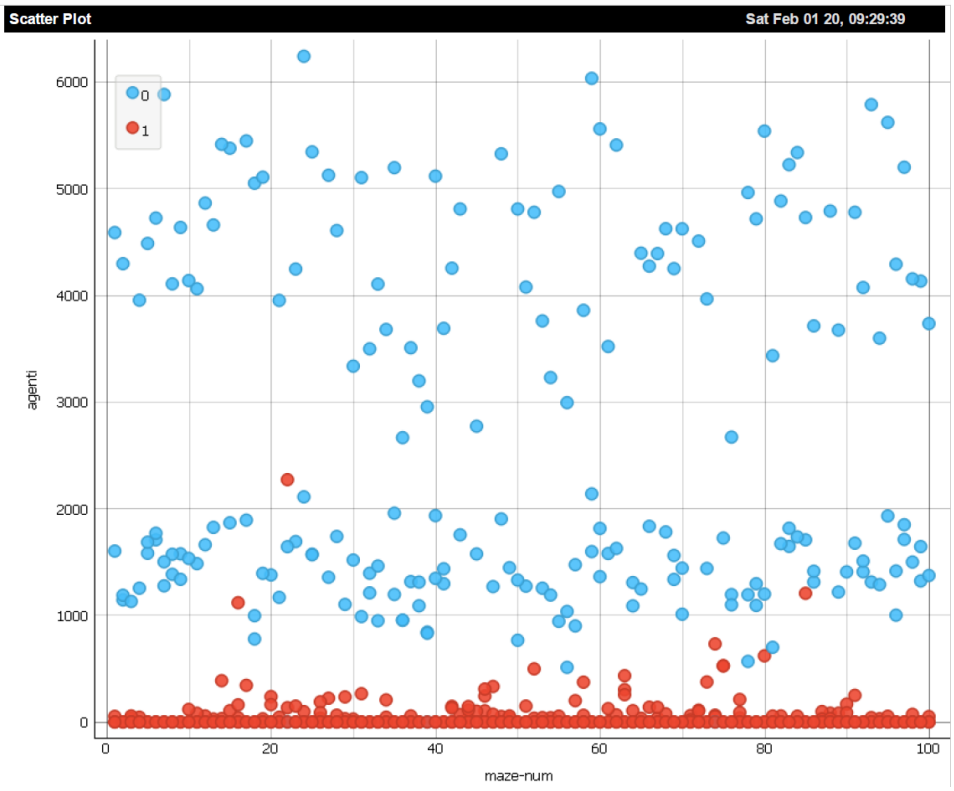

Slika 2: Boxplot s postotcima uspješnih i neuspješnih slučajeva po algoritmima

Na apscisama donjih grafova (Slike 3 i 4) su redni brojevi korištenih labirinata, a na ordinatama ukupan broj koraka/agenata. Crvena boja označava one slučajeve u kojima su se agenti našli, a plava neuspjele pokušaje. Vidimo da oni koji se nisu našli, iskoristili

su puno agenata, a time im je broj koraka znatno narastao. Ti bi brojevi nastavili rasti, da nismo uveli ograničenje radi beskonačnih petlji (1000 „steps“ u BehaviourSpace-u). Ipak imamo nekoliko iznimki, odnosno slučajeva kad su se agenti u zadnjim trenutcima pronašli.



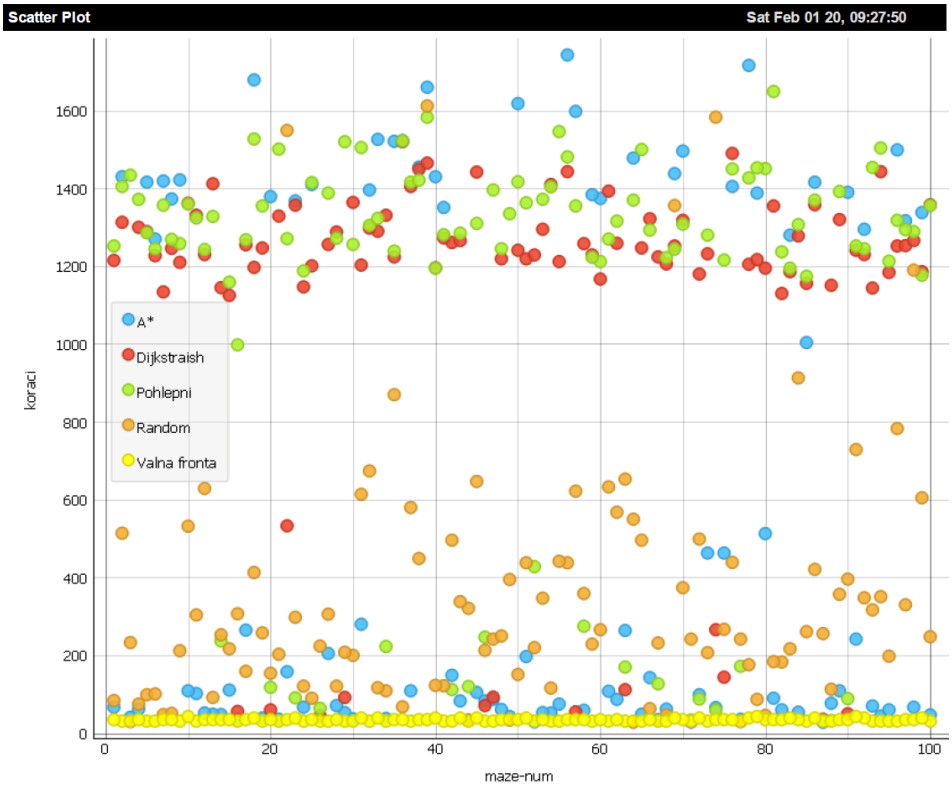

Slika 3: Scatterplot broja koraka, naglašeno je li pronađen cilj



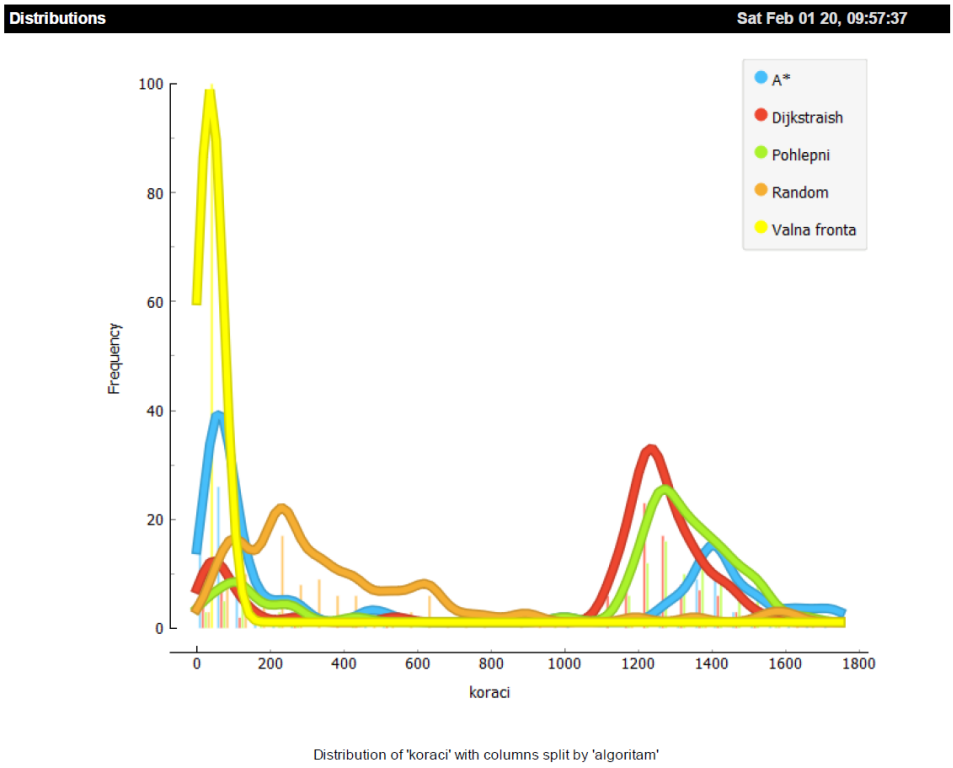

Slika 4: Scatterplot broja agenata, naglašeno je li pronađen cilj

### Broj koraka

Od ranije znamo da donji dio grafa (do oko 1000 koraka) prikazuju slučajeve kad su se agenti pronašli, pa ćemo to detaljnije analizirati (Slika 5). Prvo uočavamo da je valna fronta daleko najbolji algoritam, odnosno koristi najmanji broj koraka. Zato pogledajmo razliku ostalih prema ovom algoritmu. Algoritmi A*, pohlepni i dijkstraish koriste približno jednak broj koraka, i po tom pitanju su bolji od random algoritma. Primijetimo još da je random algoritam u određenom rasponu ravnomjerno raspoređen (Slika 6).



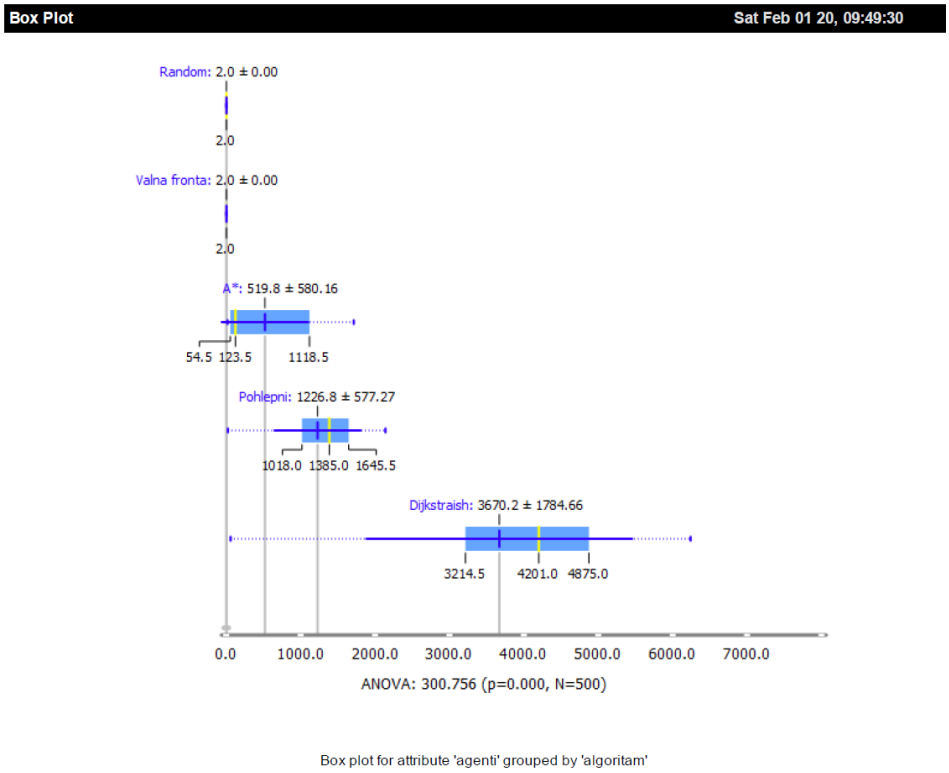

Slika 5: Scatterplot broja koraka, naglašeni algoritmi



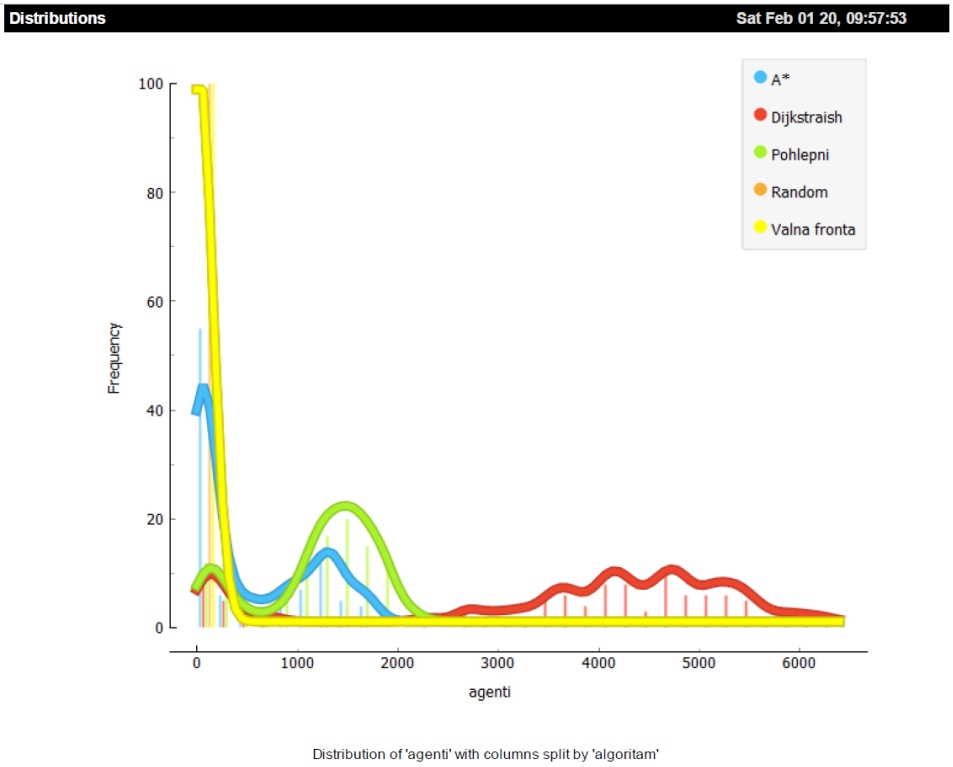

Slika 6: Distribucija koraka po algoritmu

### Broj agenata

Valna fronta i random su jednoagentski algoritmi, dakle koriste samo početne agente (Slika 7). Dijkstraish je po ovom kriteriju najlošiji algoritam jer koristi „hatch“ agente od druge, za razliku od A* i Pohlepnog koji koriste samo „hatch“ agente prve generacije. Također primijetimo da Dijkstraish ima najveći raspon broja korištenih agenata (Slika 8).



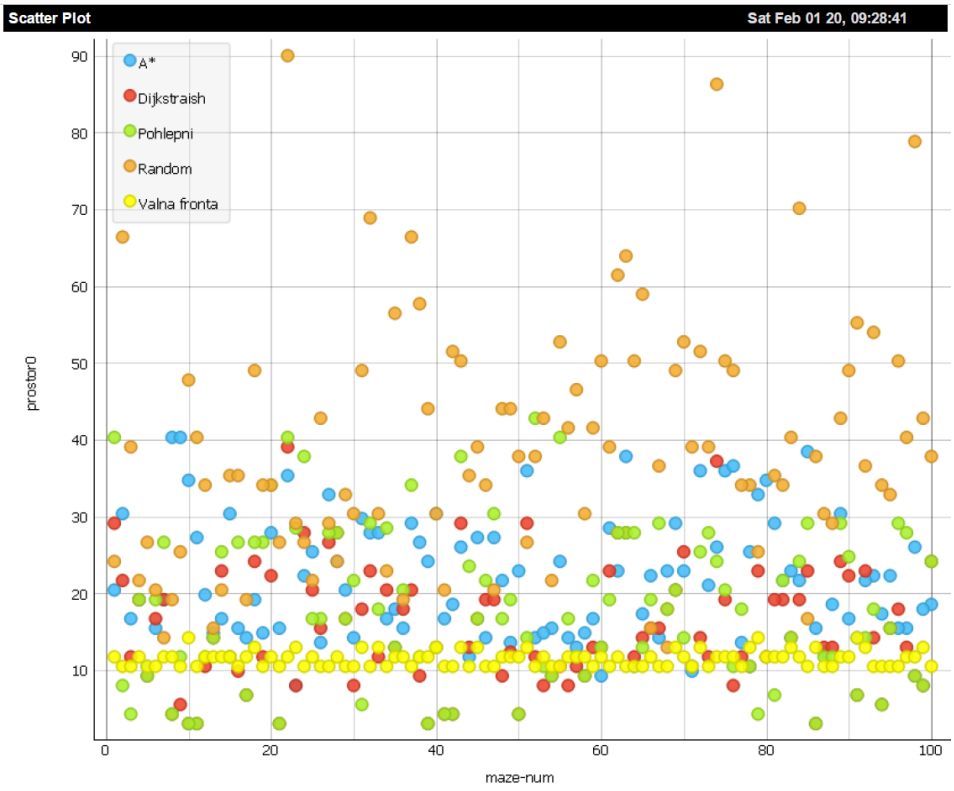

Slika 7: Boxplot broja agenata po algoritmu



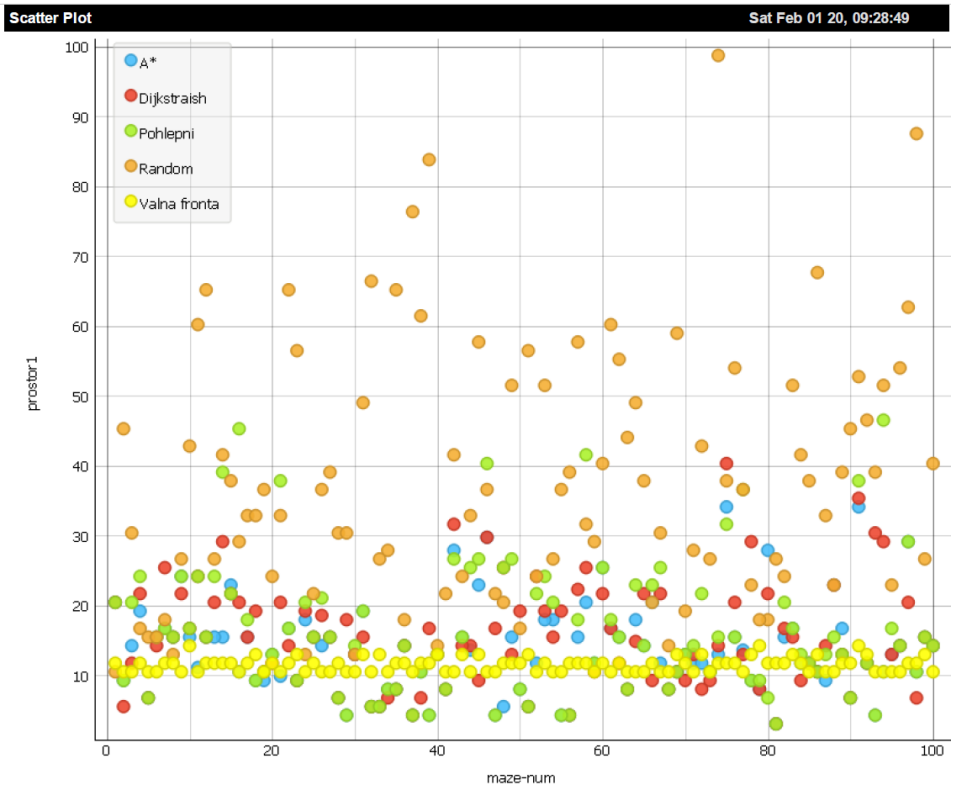

Slika 8: Distribucija broja agenata po algoritmu

### Postotak istraženog prostora

Valna fronta i ovdje će nam služiti kao referenca. Primjetimo da je postotak istraženog prostora podjednak za sve labirinte, što nam je potvrda da su labirinti približno jednake složenosti (Slike 9 i 10). Točke ispod ovog područja predstavljaju slučajeve koji se sigurno nisu pronašli jer su pretražili manji dio prostora nego valna fronta (slučajevi kad agenti jako brzo zaglave u malom prostoru). U ovom području gotovo i nema algoritma A*. Po ovom kriteriju Random algoritam je najlošiji.




Slika 9: Scatterplot postotka istraženog prostora prvog agenta

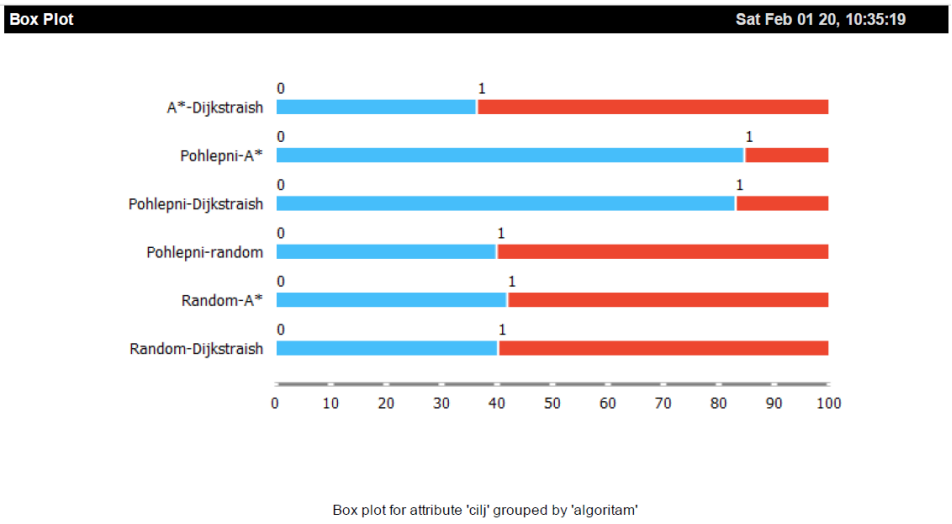



Slika 10: Scatterplot postotka istraženog prostora prvog agenta

### 4.2.2 Analiza kombinacija algoritama

#### Cilj

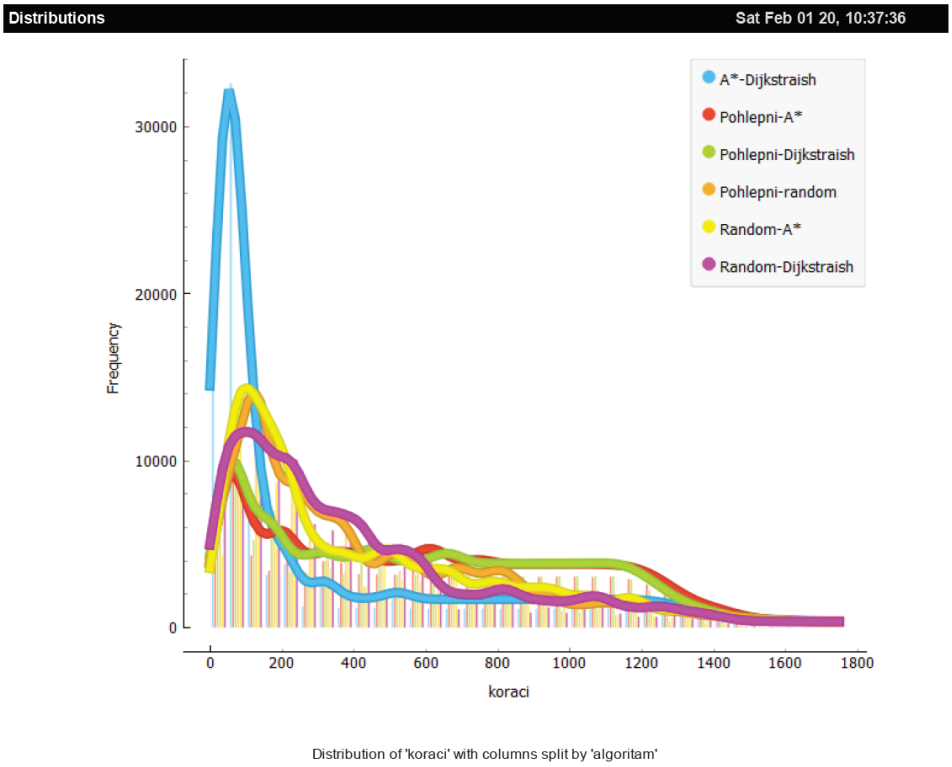
Možemo primjetiti da se uspješnost pronalaska mijenja logično obzirom na algoritme koji su kombinirani (Slika 11). Pritom naglasimo da kombinacija s Pohlepnim daje lošije rezultate, a one s Random algoritmom bolje u usporedbi sa samostalnim algoritmom.




Slika 11: Boxplot s postotcima uspješnih i neuspješnih slučajeva po algoritmima

### Broj koraka

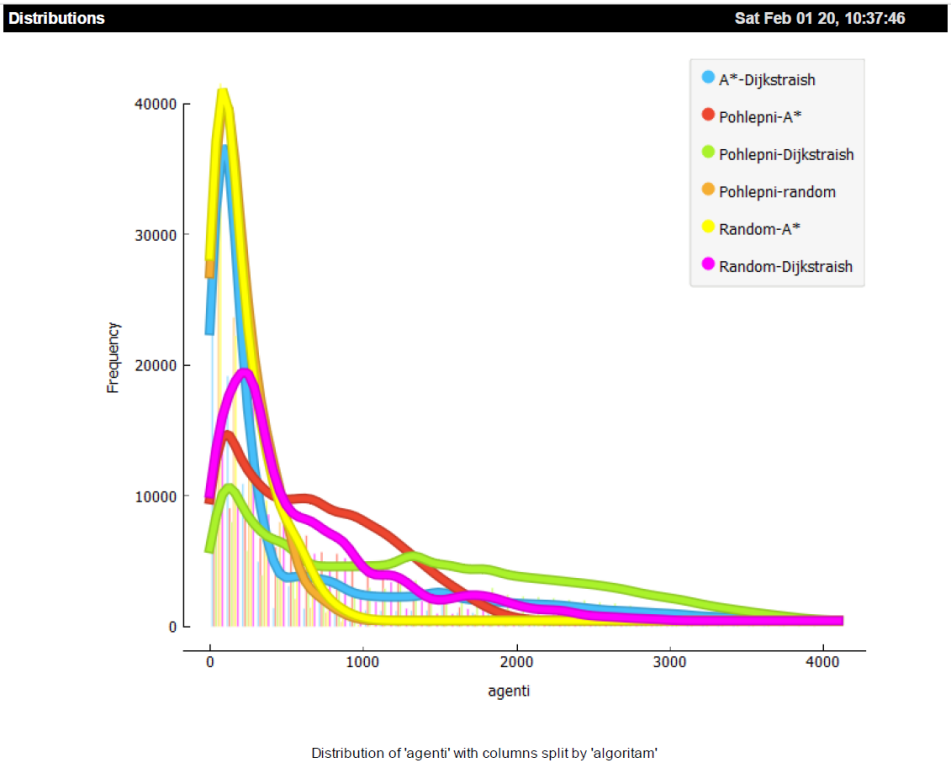
Prvo uočavamo da kombinacija algoritama A*-Dijkstraish u najviše slučajeva koristi najmanji broj koraka (Slika 12). Ostali su podjednaki, ali odvojimo crvenu i zelenu liniju (kombinacije s pohlepnim) koji imaju jednaku zastupljenost za veći raspon broja koraka.




Slika 12: Distribucija broja koraka po algoritmu

### Broj agenata

Izdvojimo prvo kombinacije Pohlepni-random i Random-A* koji očito zbog algoritma Random (jednoagentski) imaju mali broj agenata (Slika 13). Sličan njima je A*-Dijkstraish jer, iako su obe višeagentske, zbog velike uspješnosti pronalaska rješenja algoritma A* koriste mali broj agenata. Također izdvojimo Random-Dijsktraish kao jedinu kombinaciju s Random koja koristi prilično velik broj agenata, što pripisujemo algoritmu Dijkstraish.




Slika 13: Distribucija broja agenata po algoritmu

# 5 Zaključak

Kroz ovaj projekt htjeli smo implementirati različite algoritme u cilju međusobnog pronalaska dvaju agenata u labirintu. Time smo pokazali prilagodljivost algoritama pretrage na blago drugačiji problem nego za koji su ti algoritmi osmišljeni.

NetLogo se pokazao efikasnim zbog vizualizacije logike odabranih algoritama. Njegova opcija korištenja BehaviousSpace-a omogućila nam je testiranje algoritama na velikom broju labirinata. Dobivene podatke smo analizirali u programu Orange. Tu smo na raznolike načine mogli vizualizirati podatke koje smo sakupili, i lako detaljnije analizirati sve rezultate.

Pokazali smo da je, od odabranih algoritama, valna fronta najbolja po svim kriterijima koje smo promatrali. Suprotno pretpostavkama, Pohlepni i Dijkstraish algoritam su znatno manje uspješni u pronalasku rješenja iako primjenjuju „pametnu“ logiku u pretrazi. Zato je za usporedbu bilo koristno imati i Random algoritam. Problem pronalaska dvaju agenata relativno je jednostavan, ali naša implementacija može se lako prilagoditi za primjenu na kompleksnijim problemima.

# Literatura


[1] Materijali s vježbi "Uvod u umjetnu inteligenciju". (2020)

[2] Bidirectional Search. (February 2020)
Retrieved from https://www.geeksforgeeks.org/bidirectional-search/

[3] Kumar, Navin. (2019)
Bidirectional Graph Search Techniques for Finding Shortest Path in Image Based Maze Problem.

[4] A* Search Algorithm. (February 2020)
Retrieved from https://en.wikipedia.org/wiki/A*_search_algorithm

[5] Dijkstra's Shortest Path Algorithm. (February 2020)
Retrieved from https://brilliant.org/wiki/dijkstras-short-path-finder/

[6] Wilensky, U. (1999)
NetLogo.

[7] NetLogo Dictionary. (February 2020)
Retrieved from https://ccl.northwestern.edu/netlogo/docs/dictionary.html