**Lavirint**

Pronalazak izlaza iz lavirinta primenom genetskog algoritma

Duško Radić IN 39/2021

Fakultet tehničkih nauka

Operaciona istraživanja

**Opis problema**

Agent se nalazi u 2D (dvo dimenzionalnom lavirintu) koji ima ulaz i izlaz. Ulaz i izlaz su pozicije u lavirintu koje ne moraju biti na ivicima lavirnta. One mogu biti proizvoljne pozicije u lavirintu jedini kriterijum im je da postoji bar jedna validna putanja koja vodi od ulaza do izlaza iz lavirinta.

Cilj je pronalaženje putanje koja vodi od ulaza tj. početne pozicije u lavirintu do izlaza tj. završne pozicije u lavirintu.

**Okruženje**

Lavirint je podijeljen na ćelije gdje svaka ćelija ima svoju x i y poziciju i svoj tip.

Lavirint je definisan Cartesian koordinatnim sistemom. Ćelija sa pozicijom (0,0) se nalazi u gornjem lijevom uglu dok ćelija sa pozicijom (širina-1,visina-1) se nalazi u donjem desnu uglu lavirinta.

Postoje samo 2 tipa ćelija:

* **Prazna ćelija** – agent se može nalaziti u datoj ćeliji
* **Zid** – agent se ne može nalaziti u datoj ćeliji

Agent u svakom trenutku može da izabere jednu od 4 akcija. Pokret ka **gore, dole, lijevo i desno** tj. **UP, DOWN, LEFT, RIGHT.** Dijagonalno kretanje u lavirintu nije dozvoljeno. **Agentu nije dozvoljeno vraćanje na već posjećenu ćeliju.**

Za generisanje lavirnita je korišten primov algoritam.

**Genetski algoritam**

Genetski algoritam je optimizacioni algoritam baziran na principu evolucije i prirodne selekcije. Osnovna ideja je da se populacija jedinki unaprijedi (evoluira) kroz generacije dok se ne nađe optimalno riješenje ili se ne zadovolji kriterijum zaustavljanja algoritma.

**Opis koraka u korištenom genetskom algoritmu**

Koraci u osnovnom genetskom algoritmu su:

1. Inicijalizacija populacije
2. Evaluacija
3. Selekcija
4. Ukrštanje
5. Mutacija
6. Zamjena populacija i elitizam
7. Zaustavljanje

Ovaj algoritam koristi iste korake samo što smo njihov opis i objašnjenje rada drugačije rasporedili da bi bilo čitaocima lakše za razumjeti.

1. **Enkodiranje jedinki**

**Enkodiranje jedinki** možemo definisati kao proces pretvaranja stvarnog rješenja problema u neki odgovarajući format (npr: binarni niz, niz realnih brojeva) koji genetski algoritam može da obradi.

Naše jedinke predstavljaju putanju koja vodi od ulaza do izlaza iz lavirinta. Putanja je niz pozicija ćelija iz lavirinta koje je agent prešao da bi došao do izlaza. **Redoslijed ćelija u putanji je važan** pa iz toga razloga pamtimo i korake (instrukcije) koji su doveli do rješenja (putanje) jedinke.

**Jedinka pamti:**

* **Putanju -** dužine ‘L'
* **Niz instrukcija koja vodi ka rješenju (putanji) -** dužine ‘L-1’
* **Trenutnu evaluaciju (fitness) –** realan broj

1. **Ukrštanje jedinki**

**Ukrštanje jedinki (crossover) –** je proces kombinovanja genetskih informacija dva roditelja kako bi proizveli nove jedinke (potomke). Cilj je da korisne informacije roditelja prenesemo u njihove potomke.

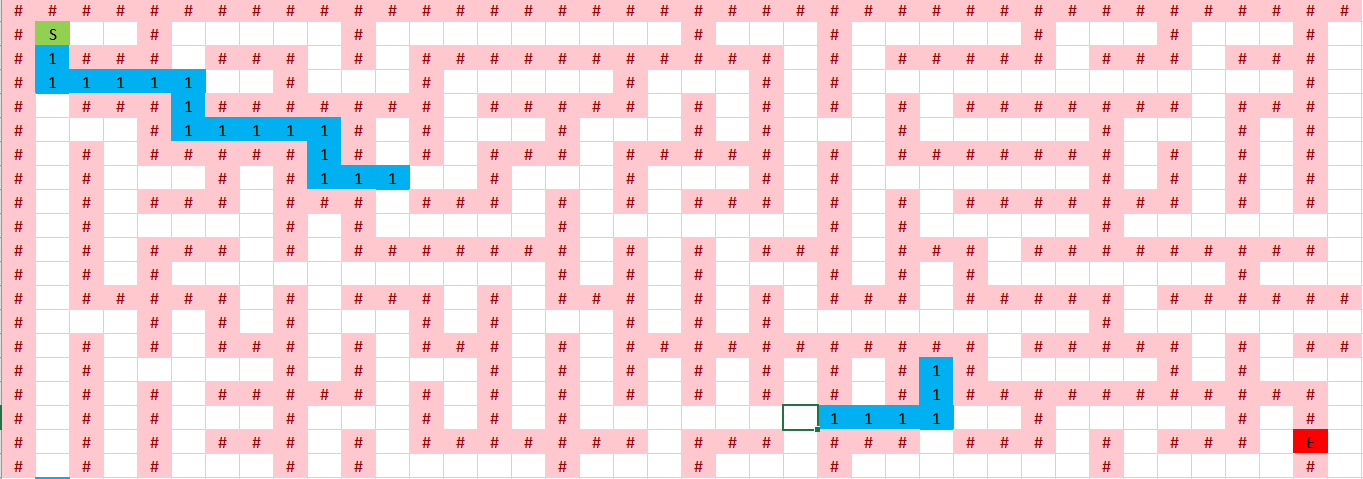
Ukrštanje naših jedinku u olbiku putanja nije moguće jer je redoslijed ćelija u putanji bitan.

Pogledajmo primjer ukrštanja putanja u dvije tačke (two point crossover):

Imamo roditelje:

1. (1,1),(1,2),...,(15,5) dužine 18
2. (1,1),(1,2) …(27,15) dužine 45

Biramo dvije nasumične pozicije a) 16 i za b) 39 gdje će mo da izvršimo ukrštanje. U ovom pristupu imamo više problema:

1. Jedinke nam nisu iste dužine.
2. Ne možemo da garantujemo da će se spajanjem dvije putanje dobiti validna putanja 

Da bi riješili ove probleme možemo da napravimo BFS algoritam koji će spajati dvije parcijalne ne validne putanje u jednu cijelu validnu putanju ako je „moguće“ jer *agent* nema pravo da se vraća unazad tj. on ne smije da posjeti ćeliju koju je već posjetio.

Drugi način je da umjesto ukrštanja putanje uradimo ukrštanje niza koraka koji vode ka rješenju (putanji) jedinke. To radimo tako što

1. Biramo *glavnog* roditelja i *donor* roditelja
2. Pronalazimo sve pozicije u *glavnom* roditelju u kojima možemo da izaberemo neku intrukciju (korak) koji nismo izabrali u trenutnom rješenju. To nam je lista mogućih pozicija za ukrštanje.
3. Biramo nasumičnu poziciju u datoj listi i biramo jednu od instrukcija koja nije bila na toj poziciji.
4. Biramo nasumičnu poziciju u nizu instrukcija *donor* roditelja
5. Od pozicije ukrštanja *glavnog* roditelja i instrukcija *donor* roditelja kreiramo novog potomka.
6. Ponavljamo algoritam ali sada sa zamjenom uloga roditelja

Ovaj algoritam nam generiše dva nova potomka koja imaju validanu putanju i niz instrukcija sa kojima je napravljena putanja.

1. **Evaluacija (fitness) jedinki**

**Evaluacija (fitness) jedinki** – se može definisati kao izračunavanje mjere vrijednosti jedinke kao rješenje problema. Cilj rješavanja problema je ili maksimizacija ili minimizacija ove funkcije.

Za ovaj problem kao fuknciju evaluacije (fitness method) je korišteno **euklidsko** rastojanje zadnje pozicije ćelije u putanji od izlaza iz lavirinita.

Koja daje bolje rezultate od **mannhattan** rastojanja:

1. **Izbor (selekcija) jedinki i elitizam**

**Izbor (selekcija) jedinki** – se može definistai kao biranje vrijednosti jedinki koje će učestvovati u stvaranju nove generacije zavisno od od njihove evaulacije (fitness). One jedinke sa većom evaluacijom imaju veću šansu da budu izabrane od onih sa manjom.

Za selekciju jedinki je izabrana ruletska selekcija. Razlog zašto je izabrana je u tome što nam je praktično dala najviše *eksperimentisanja* sa jedinkama tj. ona je davala veću šansu jedinkama koje imaju veću evaluaciju ali opet im nije dala prevelike šanse da samo one budu izabrane što je dovelo do smanjenja natprilagođavanja (overfitting) algoritma.

1. **Mutacija**

**Mutacija** se može definisati kao povremena izmjena nekih gena u jedinci kako bi se sačuvala raznolikost populacije. Raznolikost populacije nam pomaže da izbjegnemo natprilagođenje algoritma.

Kao stopu mutacije je izabrano 0.1 (10% da se jedinki se mutira). Razlog zašto je izabrana ovoliko velika stopa mutacije je baš zato da bi se izbjeglo natprilagođenje algoritma. Stopa >0.1 je davala previše nasumične rezultate koji nekada nisu dopustili algoritmu ni da konvergira ka lokalnom optimumu.

Algoritam mutacije i isti kao i 2. i 3. korak u ukrštanju jedinki.

1. **Kriterijum zaustavljanja**

Algoritam se zaustavlja ako se prekoraci broj generacija populacije ili ako se nađe rješenje (ne mora nužno biti optimalno) problema.

**Rezultati**

Lavirint je 40x20 (WxH). 40 ćelija u širini i 20 ćelija u visini sa fiksnim pozicijama za ulaz i izlaz. Imamo maksimalno 50 generacija, što je ujedno i jedan od kriterijuma zaustavljanja. Svaka generacija ima 100 jedinki, broj jediniki je isti za svaku generaciju. Za svaku kombinaciju paramatera testa je izgenerisano 10 testnih primjera, svaki sa razlicitim seed-om.

Za algoritme koji su koristili **mannhattan** rastojanje imali smo najlošije performanse gdje je srednja evaluacija generacija bila 8.72 a broj riješenih problema 1/11 sa stopom mutacije od 0.1 i stopom elitizma od 0.4

Za algoritme koji su koristili **euklidsko** rastojanje smo imali bolje rezultate:

|  |  |  |  |
| --- | --- | --- | --- |
| Stopa mutacije | Stopa elitizma | Prosječna evaluacija | Broj riješenih problema od 11 |
| 0.1 | 0.0 | 19.6 | 0 |
| 0.1 | 0.2 | 5.65 | 3 |
| 0.01 | 0.2 | 9.04 | 2 |
| 0.1 | 0.4 | 7.16 | 2 |
| 0.01 | 0.4 | 7.89 | 3 |
|  |  |  |  |
|  |  |  |  |

Iz ovih navedenih primjera vidimo da dobijamo u prosjeku bolje rezultate za slučajeve gdje više eksperimentišemo nego u onima gdje manje. Najbolje rezultate smo imali kada smo imali visoku stopu mutacije i malu stopu elitizma (ali ne premalu) nego u drugim slučajevima.

Jedan način na koji je probano poboljšanje eksperimentisanja generacije je u ukrštanju jedinki. Umjesto da popunjavamo potomke jedinki sa instrukcijama *donor* roditelja možemo da pustimo potomak da pronađe svoj put ka cilju. Za ovaj slučaj ukrštanja smo imali sledeća rješenja

|  |  |  |  |
| --- | --- | --- | --- |
| Stopa mutacije | Stopa elitizma | Prosječna evaluacija | Broj riješenih problema od 11 |
| 0.1 | 0.0 | 19.46 | 1 |
| 0.1 | 0.2 | 6.98 | 4 |
| 0.01 | 0.2 | 3.61 | 5 |
| 0.1 | 0.4 | 7.73 | 3 |
| 0.01 | 0.4 | 4.61 | 4 |
| 0.1 | 0.6 | 8.01 | 2 |
|  |  |  |  |

**Moguća poboljšanja**

Funkcija evaluacije se može poboljšati tako da koristi rastojanje svake ćelije u svojoj putanje do izlaza. Problem do koga smo došli prilikom implementacije takve funkcije je što putanje jedinki nisu iste dužine pa ako koristimo neku suviše jednostavnu funkciju koja je NPR:

Gdje je ***dist*** funkcija rastojanja između dvije tačke u lavirintu, ***exit*** pozicija izlaza i pozicija u putanji

Ona neće kazniti putanje koje su previše duge više nego one koje su kratke i koje nemogu da stignu do izlaza. Pronalaženje bolje funkcije evaluacije bi znatno poboljšalo riješenja algoritma ali je teško naći funkciju koja je *glatka* i ravnomjerno kazni prekratke i preduge putanje.

**Zaključak**

U testnim rješenjima smo ustanovili da algoritmi koji više eksperimentišu ili imaju više nasumičnih aspekata (kao što je u ovom slučaju bilo ukrštanju jedinki) generalno imaju bolje performanse od onih modela koji ne eksperimentišu dovoljno. Kroz testne primjere smo primjetili da su najbolja rješenja davali oni algoritmi koji su koristili euklidsko rastojanje kao funkciju evaluacije jedinke sa stopom mutacije od 0.01 i stopom elitizma 0.2 sa ukrštanjem koje ne koristi gene *donor* roditelja.

Za relativno male lavirinte (pretpostavljamo do 50x50) je bolje koristiti neki od algoritama kao što su dijkstra, BFS,DFS ili A\* jer za male lavirinte prostorska kompleksnost nije toliko veliki problem, dodatno ovi algoritmi garantuju da će pronaći optimalno rješenje problema. Za velike lavirinte sa velikom prostorskom kompleksnostšću, koristiti genetski algoritam ima prednosti u odnos na obični greedy (pohlepni) algoritam.