SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4374

Vizualizacija ponašanja algoritma rojeva čestica prilikom rješavanja problema labirinta

Viran Ribić

Zagreb, lipanj 2016.

SADRŽAJ

1. Uvod

Ubrzani napredak tehnologije neminovno donosi sve veća očekivanja od nadolazećih naraštaja. Zahtijeva se velika adaptivnost u novim okolinama i mogućnost prilagodbe novim problemima kako se domene različitih znanosti i usluga počinju preklapati u međusobnoj interakciji. Ta interdisciplinarnost i potreba za stručnjacima širokog spektra znanja najbolje je vidljiva na području računarnog i elektrotehničkog inženjerstva kako nove tehnologije pronalaze primjenu u područjima ekonomije, menadžmenta, biologije, medicine i otvaraju nove mogućnosti pri suočavanju s njihovim izazovima.

Takva potražnja iziskuje veća očekivanja od zaposlenika koji u raspoloživom vremenu moraju savladati veću količinu informacija i znanja. Jedna od tih vještina je razumijevanje područja umjetne inteligencije, njezinih prednosti i ograničenja. Računalni vid, ekspertni sustavi, stojno učenje, samo su neki od primjera grana umjetne inteligencije koji sve više postaju integrirani dio naše svakodnevice, od mobilnih aplikacija do enterprise sustava.

Inženjeri sposobni u modeliranju problema i implementaciji tehnika umjetne inteligencije proširuju svoje vidike novim znanjima koja se mogu primijeniti na široki spektar problema. Iako možda rješenje bazirano na umjetnoj inteligenciji nije uvijek primjereno i nužno, uvid u te metode potiče inovativnost pri formiranju drugih rješenja i daje širi spektar djelovanja onome tko tim znanjem raspolaže.

No, područje umjetne inteligencije je jako široko i svestrano. Bez dobre matematičke podloge nije ju moguće savladati s razumijevanjem, budući da se većina njezinih koncepata temelji na matematičkim metodama. Za nekog tko se prvi put susreće s ovim područjem potrebno je puno proučavanja literature i rješavanja praktičnih zadataka kako bi se izgradila potrebna intuicija.

Kako bi se ubrzao proces usvajanja tih vještina, ovaj rad fokusirao se na izgradnju okruženja za vizualizaciju jednog od pristupa evolucijskog računanja. Takvo okruženje trebalo bi pružiti sučelje za interkaciju s nekim od algoritama evolucijskog računanja i potencijalno olakšati razumijevanje osnovnih koncepata. Evolucijsko računanje je podgrana umjetne inteligencije koja potragu za rješenjem provodi imitirajući prirodne procese. Simulacija ponašanja takvog algoritma provest će se na problemu potrage za izlazom iz labirinta.

Nakon uvoda, dan je opis razmatranog problema, njegovih inačica i poznatih tehnika rješavanja.cSlijedi detaljniji opis algoritama evolucijskog računanja uz osvrt na karakteristične elemente takvih algoritama. Zatim je potrebno razmotriti algoritme rojeva čestica kao ciljane skupine za rješavanje i vizualizaciju problema labirinta. Provedena je diskusija integracije algoritma i metoda vizualizacije. Nakon toga slijedi poglavlje o samoj implementaciji, karakterističnim elementima i primjeri izvođenja. Za kraj dan je osvrt na postignute rezultate uz prijedloge vezane za budući rad na ovu temu.

2. Potraga za izlazom iz labirinta

Koncept labirinta pojavljuje se na području sredozemlja prije vise od dvije tisuće godina. Ovaj već dugo poznati problem opisan je s ulaznom i izlaznom točkom, te nizom hodnika od kojeg je samo manji broj ponuđenih putanja uistinu onaj koji vodi do rješenja. Kroz povijest predložen je i definiran veliki broj labirinata različitih karakteristika.

Prema načinu zapisivanja razlikujemo:

- labirinte zadan težinskim grafom,
- blok labirinte,
- numerički labirint,
- klasično definirani labirint.

Labirint zadan težinskim grafom limitira moguće poteze u pojedinom čvoru labirinta i u suštini predstavlja samo drugačiji zapis klasičnog labirinta. Blok labirint rješava se mijenjanjem predefiniranog broja pozicija kako bi se postigla povezanost ulaza i izlaza. Ovaj je zadatak kombinatorne prirode, i nešto je složeniji od klasičnog labirinta. Numerički labirint proširenje je klasičnog u kojem pozicije labirinta sadrže i cjelobrojnu vrijednost. Taj broj označava za koliko se koraka subjekt mora pomaknuti u proizvoljnom smjeru. Posljednji u ovom nizu jest klasično definirani labirint . To je labirint neoznačenih hodnika koji ima barem jedan povezani put od ulaza do izlaza [1].

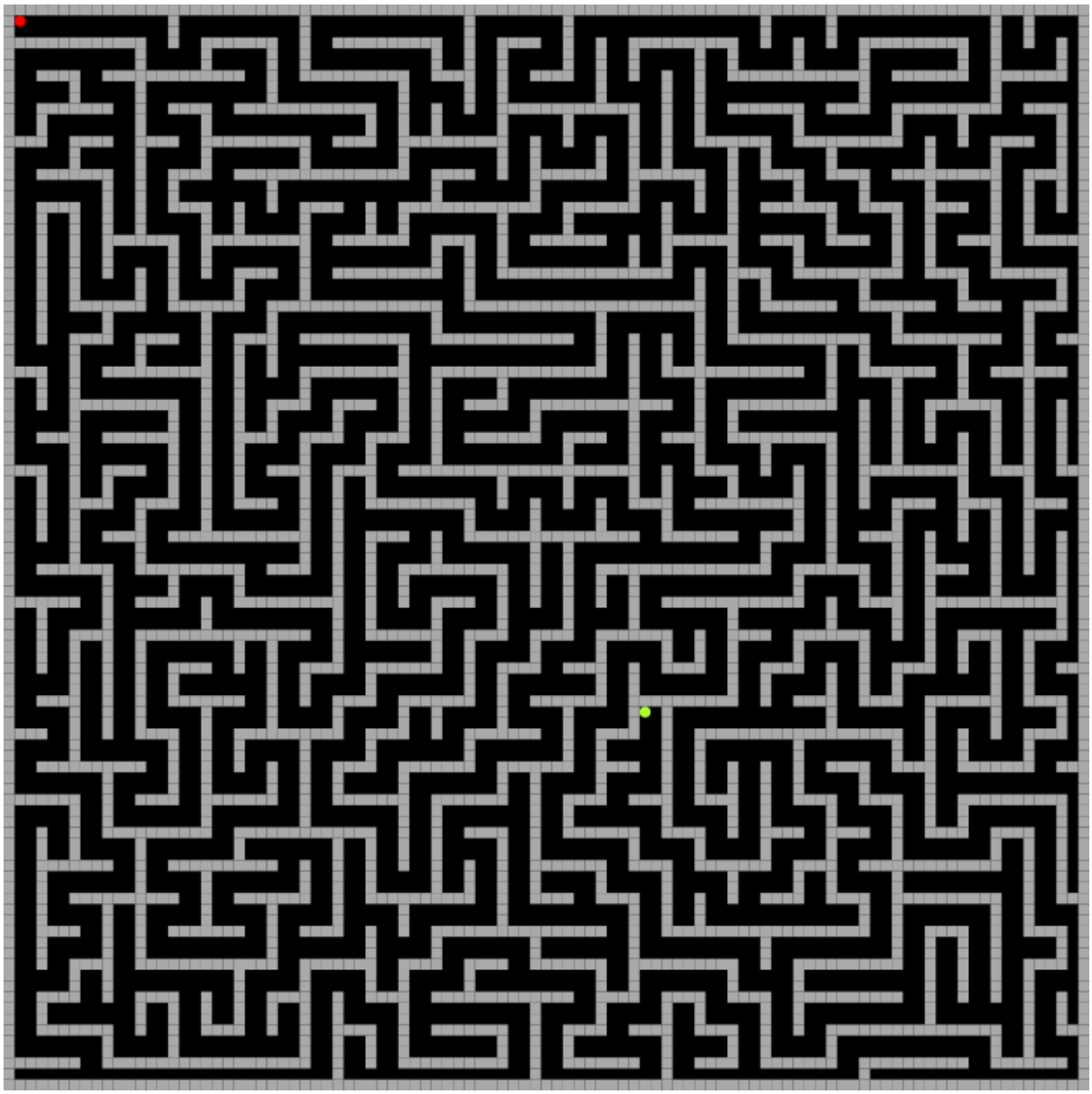
Iduća kategorizacija labirinata može se provesti po dimenziji problema. Labirint se može definirati u n-dimenzija ukoliko za takvu dimenzionalnost možemo napisati funkciju prijelaza iz jednog stanja u drugo. Više dimenzije donose i dodatne stupnjeve slobode prilikom definiranja problema, pa tako 3D labirint možemo tretirati kao hiperlabirint gdje koraci nisu točke već površine koje subjekt ostavlja za sobom dok se kreće. Za labirint dimenzije n koraci su koraci dimenzije n-1. Osvrnimo se samo na činjenicu da se različitim metodama najviše 4D labirinti mogu uspješno vizualizirati tako što bi posljednju dimenziju tretirali kao vremensku koordinatu. Iz tog razloga

gornja granica na dimenzionalnost labirinta razmatrana u okviru ovog rada iznosi 4.

Strategije usmjeravanja najvažniji su aspekt pri generiranju labirinta, a dva najinteresantnija predstavnika su savršeni i isprepleteni labirinti. Savršeni labirint ne dozvoljava petlje niti izolirana polja te su svojom strukturom ekvivalentni stablima. Listovi su slijepe ulice od kojih točno jedan sadrži i izlaz iz labirinta. Isprepleteni labirinti oslabljuju zahtjeve savršenih labirinata utoliko što dozvoljavaju pojavu petlji, zatvorenih putanja koje dovode igrača do početne pozicije.

Tu bi valjalo napomenuti da je do danas osmišljen popriličan broj algoritama koji se znaju nositi s problemom labirinta. Dio njih specijalizirao se za savršene algoritme poput Wall followera i Maze-Routing algortima. Ukoliko labirint ima više različitih načina kako pronaći izlaz, ili ima više izlaza od kojih želimo naći najbliži, mogu se uzeti u razmatranje algoritmi breadth-first search, depth-first searchili A* ako imamo definiranu prikladnu heuristiku [2].

Odlučeno je da će se simulacija provoditi za klasični labirint u 2D prostoru dok će se za strategije usmjeravanja iskoristiti oba pristupa. U okviru simulacije korisnik će moći promatrati ponašanje algoritma kroz oba pristupa te sam izvoditi zaključke o efikasnosti na pojedini problem. Odabrana je i zetta geometrija ćelija, koja definira da se na pravokutnoj matrici labirinta iz jednog stanja subjekt može kretati u 8 smjerova, svaki pod kutom od 45 stupnjeva u odnosu na prethodni. Ovako definirani labirint realizirat će se matricom pozicija, ali će radi prilagodbe problemu, informacije o pozicijama algoritmu prezentirati kao omeđeni kontinuirani prostor. Slika 2.1 prikazuje jednu inačicu našeg labirinta.



Slika 2.1: Primjer labirinta

3. Evolucijsko računanje

3.1. Motivacija

Evolucijsko računanje je grana umjetne inteligencije zasnovana na Darwinovim principima evolucije. Ono što u računalima nalazimo kao optimizaciju, u našem je svijetu prirodna selekcija. Ono što su u prirodi organizmi, to su u računalu interpretacije rješenja, nekakve kombinacije brojeva koje čine odgovor na neki problem. Već u ovoj uvodnoj motivaciji pronalazimo sličnosti prirodne evolucije i pristupu optimizacijskom problemu. Kroz naredno poglavlje pogledat ćemo sve sličnosti i predstaviti argumente zašto je evolucijom inspiriran pristup optimizacijskom problemu izvediv i valjan.

Procesi evolucije u prirodi odvijaju se među kompetitivnim jedinkama koje se razlikuju po nekim svojim obilježjima. Tako će obilježja poput brzine, veličine, snage ili mogućnosti za kamuflažu kod nekih jedinki rezultirati boljom prilagodbom na prirodu koja ju okružuje. Isto tako, neka su rješenja koja razmatramo u našem računalnom sustavu bolja od drugih, po funkciji cilja kojoj teže, a koju zadovoljavaju u većoj ili manjoj mjeri. Jedinke koje su se bolje prilagodile na uvjete prirode opstaju kroz vrijeme, a rješenja koja su se pokazala najboljim među dostupnim mogu biti prihvaćena kao ona koja želimo uzeti kao konačna. Ta rješenja mogu se grupirati i najbolja rješenja možemo pokušati kombinirati s njima sličnim, kako bi potencijalno ostvarili još i bolje rezultate. Upravo je to vidljivo i u prirodi: križanjem genetskog materijala organizama nastaju potomci koji imaju najbolje karakteristike svojih roditelja.

Priroda je puna pojava koje se čine slučajne, nepredvidljive, i koja su čak i za današnja shvaćanja još uvijek nerazumljive. Ti nasumični događaji čine da dominantni organizmi kroz vrijeme propadnu i bivaju zamijenjeni novim, boljim oblicima. Jedna zanimljiva manifestacija sigurno je mutacija koja unosi neizvjesnost u sustav, ponekad na štetu, a ponekad na korist. Takav mehanizam osigurava nam "kretanje" rješenja u računalnom sustavu, gdje se iz konstantnog sukoba križanja dobrog i poznatog materi-

jala s neotkrivenim i nasumičnim mutacijama postižu sve bolji i bolji rezultati.

Iz ove usporedbe vidi se da između prirodnih i umjetnih evolucijskih procesa postoji puno dodirnih točaka i sličnosti. Problem koji rješavamo mora biti moguće opisati funkcijom dobrote ili kazne. Kada se govori o funkciji dobrote, razmatramo problem pronalaženja maksimuma, odnosno najveće moguće dobrote. Pronalaženja minimuma kod funkcije kazne svodi se na postupak favoriziranja onih rješenja koja su bliže nuli. To preslikavanje mora obuhvatiti sve bitne aspekte problema kako bi rješenje bilo što vjerodostojnije. Ukoliko je funkcija dovoljno bogata informacijama postupak usporedbe rješenja rezultirat će kvalitetnijim odabirom boljih jedinki. Ako je naša funkcija glatka i monotono rastuća ili padajuća, ovisno o problemu, smanjuju se vrijeme izvođenja programa. S druge strane jako oscilirajuće funkcije i funkcije s platoima, područjima u kojoj vrijednost funkcije poprima konstantnu vrijednost, mogu uzrokovati zastoj ako algoritam nema mehanizam kojim bi se zaštitio od njih. Posljedica je prisilna konvergencija k suboptimalnom rješenju, koja se još naziva i lokani optimum.

Lokalni optimum samo je prividno najbolje rješenje i čest je uzrok slabim rezultatima izvršavanja algoritma koji sprječava pronalazak globalnog optimuma. Globalni optimum naziv je za točku n-dimenzijskog prostora problema u kojem je funkcija evaluacije najbolja. Svaki je globalni optimum ujedno i lokalni i iz te činjenice proizlazi problem pravilnog razlikovanja ta dva konstrukta. Idući primjer osmišljen je radi zornijeg prikaza problematike.

Analizirajmo ponašanje planinara koji je odlučio osvojiti najviši vrh nekog područja, a pritom nije uzeo nikakva tehnološka pomagala koja bi mu mogla pomoći u snalaženju. Sve se odluke zasnivaju na onome što vidi oko sebe. Ukoliko se za vrijeme njegovog uspona spustila magla, smanjuje mu se broj odluka koje može donijeti jer su mu uskraćene informacije o njegovoj neposrednoj okolini (susjednim vrhovima). U tom trenutku može se odlučiti proglasiti trenutnu poziciju najvišom i vratiti se nazad ili pokušati istraživati okolo u potrazi za boljom, uz opasnost da se na taj isti proplanak neće moći vratiti i potencijalno završiti svoju ekspediciju s još slabijim rezultatom. U konceptualno istim uvjetima djeluju elementi algoritma koji nose informaciju o domeni kojom tragaju, primjerice nadmorskom visinom planine. Vidimo da je ovaj pristup poprilično neefikasan: slučajnim odabirom donosimo odluke i nadamo se najboljem, što u velikim prostorima daje malu vjerojatnost uspjeha. Iz ovog primjera je također zorno predočena potreba za kolaboracijom više jedinki.

Uz pretpostavku scenarija u kojem se na planinu penje više penjača, izgledi za uspjehom značajnije rastu. Penjačima je sada omogućena i međusobna kolaboracija, kojom mogu djelovati kao tim kako bi postigli zajednički cilj. U slučaju magle mogli bi se raspršiti tek dovoljno da čuju jedan drugoga, te dovikivanjem signalizirati gdje se tko u kojem trenutku nalazi. Robusnost takvog pristupa kompenzira grešku pojedinačnih procjena i korak je bliže k onom ponašanju koje želimo modelirati. Dodajmo još opcionalno, ali poželjno svojstvo očuvanja najboljeg rješenja u svakom trenutku. Kod evolucijskih algoritama ono je sadržano u elitizmu, odnosno mehanizmu pohrane najboljeg rješenja u svakom trenutku traganja kako bi se omogućilo sigurno traganje za boljim rješenjima bez opasnosti gubitka do sad pronađenog optimuma.

Navedeni primjeri sadrže temeljne koncepte evolucijskog računanja:

- problem i pripadna funkcijom ocjene,
- populaciju rješenja i njezine ocjene,
- strategije manipuliranja rješenjima kroz iteracije algoritma.

3.2. Programksko ostvarenje

Iako je u prirodi proces evolucije konstantni proces koji nema jasno definiran kraj, izračuni u računalu moraju završiti u konačnom broju koraka kako bi se izlaz programa mogao iskoristiti. Ishod optimizacijskog procesa algoritama evolucijskog računanja ne garantira pronalazak najboljeg rješenja sa svakim pokretanjem. Kako bi se zaštitili od toga i prihvatili suboptimalna rješenja, proces optimizacije ne izvršava se u beskonačnoj petlji. Najčešći mehanizmi zaustavljanja uključuju brojanje iteracija u kojima je algoritam izvršio potpunu evaluaciju populacije, vremenski ograničenu optimizaciju te niz različitih i specifičnih mehanizama za otkrivanje stagnacije kvalitete rješenja za koje želimo prijaviti kraj izvršavanja.

Raznolikost algoritama evolucijskog računanja proizlazi iz slobode manipuliranja operatorima izvršivi nad populacijom. Od nekolicine pristupa izdvojit ćemo dvije podgrane: evolucijski algoritmi i algoritmi rojeva čestica. Kako će naglasak ovog rada biti na algoritmima, ponašanju i inteligenciji rojeva čestica, najprije ćemo dati manji osvrt na evolucijske algoritme, točnije najpoznatijeg predstavnika - genetski algoritam.

Ono što nam je kod njega zanimljivo jest njegova izravna povezanost s prirodnim procesima opisanim u uvodu poglavlja. Rješenje u računalu modelirano je nizom bitova interpretiranih kao računalni primitivi ili strukture koje su ulazni parametri u funkciju evaluacije. Jednom kad je njihov položaj u domeni problema određen iznosom odgovarajuće funkcije, sljedno se primjenjuju tri genetska operatora:

- selekcija,
- križanje,
- mutacija.

Selekcija iz populacije odabire jedinku za daljnju obradu, pa tako neke od različitih strategija selekcije uljučuju: nasumičnu selekciju, k-turnirsku selekciju i proporcionalnu selekciju. Križenje je proces kombiniranja segmenata rješenja dvaju ili više razmatranih roditelja. Tu na primjer nalazimo križanje s t točaka prekida i uniformno križanje. Mutacija se pak izvodi kao izmjena pojedinog podatka ili još atomarnije, nekolicine bitova, uz neku malenu vjerojatnost te tako simulirati nepredvidivost [3].

Primjer psudokoda koji opisuje zajedničko ponašanje svih evolucijskih algoritama dan je na slici 3.1 .

Pseudokod 1 Genetski algoritam

Ulaz: $problem$ – definicija problema.

Izlaz: $rezultat$ –najbolje pronađeno rješenje.

$pop := generirajPop(problem)$

for ($p \in pop$) **do**

$racunajDobrotu(p)$

end for

repeat

for ($i \in prebroji(pop)$) **do**

$p1, p2 := birajIzGeneracije(pop)$

$p' = spoji(p1, p2)$

$izmjeni(p')$

$evaluiraj(p')$

$pop'.dodaj(p')$

end for

$pop = pop'$

until $\neg uvjetPostignut()$

return $vratiNajbolju(pop)$

Slika 3.1: Pseudokod genetskog algoritma

4. Inteligencija rojeva čestica

4.1. Inspiracija

Kao i kod ostalih algoritama evolucijskog računanja, inspiracija za algoritme rojeva čestica također dolazi iz prirode, ali tu ne promatramo proces evolucije već njen krajnji rezultat. U velikim skupinama jako povezanih entiteta javlja se zanimljiv fenomen nazvan izranjajuća inteligencija (emerging intelligence). Vođeni mišlju "The whole is larger than the sum of its parts" prirodni sustavi navedene strukture mogu postići nevjerojatne stvari. Ono što je posebno zanimljivo jest da su subjekti takve zajednice često veoma jednostavni organizmi koji samostalnim djelovanjem ne bi uspjeli postići jednake rezultate. Informacije koje svaki subjekt prikupi akumuliraju se u svijest roja (hive mind), apstraktnu tvorevinu kojom opisujemo centralno tijelo zaduženo za provedbu odluke grupe. Na taj se način direktnom komunikacijom subjekata sa svojim susjedima u prostoru postiže propagacija informacije čitavim rojem. Direktno i indirektno, svaki entitet populacije ima na raspolaganju svake informacije o njegovoj okolini. On se tada može lakše osloniti na tuđe odluke, jer je njegov cilj zajednički za cijelu populaciju. Ovakav način djelovanja ima i još jedno interesantno svojstvo a to je otpornost na pogreške.

Pošto se takva konglomeracija organizama formira, kreće i djeluje s ciljem postizanja osobnog i općeg interesa, greške u takvom sustavu gube se u masi boljih informacija dok se u isto vrijeme taj mehanizam brine da dobre informacije propagiraju do svih jedinki. Za takve sustave kažemo da su robusna, i to je svojstvo samoispravljanja i otpornosti itekako poželjno pri izgradnji bilo kojeg računalnog sustava. U ovom ponašanju vidimo uzorke koji se ponavljaju, a zajednički su prethodno opisanom prirodnom evolucijom a time i računalnom optimizacijom. Formiranje i održavanje roja temelji se na dijeljenoj funkciji cilja koja je prirodno definirana i jasna za implementaciju. U većini prirodnih sustava ona je potraga za najbogatijim izvorom hrane ili najsigurnijim skloništem. Vidljiv je čak i mehanizam napredovanja za koji možemo zamisliti da sva

uzrokovana rješenja “slijede” ono najbolje sve dok ona uistinu vodi najboljim putem, a kada to više nije slučaj, lako će se zamijeniti novim favoritom. U nastavku ćemo opisati kako se takav sustav može simulirati u računalu.

4.2. Definicija

Inteligencija roja čestica (engl. Swarm Intelligence) bavi se računalnim sustavima inspiriranim „kolektivnom inteligencijom“ [4]. U prirodi, postoji mnogo primjera rojeva koji su inspirirali znanstvenike u dizajniranju svojih algoritama, a samo neki od njih su: jata ptica, rojevi pčela, kolonije mrava ili čopori vukova. Za subjekte populacije koristit ćemo termin inteligentni agenti i oni će predstavljati početno nasumično odabrani uzorak domene problema. Oni će biti vođeni kolektivnom inteligencijom u procesu optimizacije. Inteligentni agent nositelj je inteligentnog ponašanja u terminologiji umjetne inteligencije i opisuje entitet koji je u stanju donositi odluke i djelovati u skladu s njegovom pozicijom u prostoru problema. Prostor problema čini skup svih mogućih vrijednosti koje čine rješenje optimizacijskog problema. Te su točke mapirane na odgovarajuću funkciju cilja pomoću koje agente možemo uspoređivati. Ono što čini suštinsku razliku između algoritama ovog pristupa su strategije kako navigirati jedinkama kroz iteracije. Algoritam pčela šalje različite vrste radnika kako bi skupile informacije i iskoristile trenutno najbolje poznate pozicije, algoritmi mrava lutaju ostavljajući feromon iza sebe koji drugi mravi mogu osjetiti i prema kojim se mogu navoditi. Općenito, algoritam roja čestica, oslanja se na kretanje cijele populacije pri donošenju pojedinačnih odluka uzimajući u obzir informacije o prethodno najboljoj poziciji, globalno najboljoj poziciji i pozicijama na kojoj se agenti trenutno nalaze. U našem ćemo problemu koristiti upravo taj posljednji pristup, no prije detaljne analize algoritma, opisat ćemo još malo prednosti i nedostatke svih algoritama rojeva čestica.

4.3. Prednosti i nedostaci

Kako rojevi u prirodi dolaze u različitim formacijama i oblicima, a njihova je stabilnost i dalje zadržana, pokazano je da je mogućnost skaliranja takvih sustava zadržana i u njihovim simuliranim inačicama. Nadalje, adaptivnost je iduća ključna značajka gdje se zbog samoorganizirajućih i autokonfigurirajućih svojstava mogu dinamički prilagođavati promjenjivoj okolini, sve za vrijeme izvođenja. Skalabilnost i adaptivnost za posljedicu imaju otpornost na manje perturbacije na neispravne podatke i usješno

amortiziraju njihov negativni učinak. U kontekstu izračunljivosti, rojevi čestica najzanimljiviji su nam zbog načina na koji se nose sa NP teškim problemima, i tako pružajući nam rješenja za široki spektar novih problema. Pa ipak, njihova primjenjivost dolazi s cijenom. Naime, za razliku od klasičnih algoritama koji specificiraju ulazne podatke i niz koraka koji vode k cilju, algoritmi evolucijskog računanja ne mogu garantirati potpunu ispravnost njihovog ishoda. Kako su bazirani na stohastičkim procesima, ovi algoritmi rezultiraju aproksimacijom rješenja koja će u većini slučajeva biti zadovoljiva, ali ne i apsolutna. Ali ni tu zadovoljivost ponekad nije lako postići.

Općenito, algoritmi evolucijskog računanja predstavljaju recept kako formulirati problem i pristupiti rješavanju. Ono što čini razliku unutar samog algoritma, što se razlikuje od jednog do drugog pokretanja i što je ključno pri izvršavanju su ulazni parametri algoritma. To su često težinski koeficijenti intenziteta nekih svojstava, granice na korake koje agenti mogu poduzeti u pojedinoj iteraciji i slično, a vezani su za sam problem, njegovu veličini, zahtjev na brzinu izvođenja ili dostupnost memorijskih resursa itd. Proces odabira parametara često je baziran na principu pokušaja i pogreške, a dobri rezultati postižu se i adaptacijom tih parametara tijekom izvođenja, ukoliko je moguće definirati takav mehanizam.

5. Optimizacija rojem čestica

Optimizacija rojem čestica najopćenitiji je od svih algoritama rojeva čestica i čini bazu za mnoge varijacije. Formalno, zadaća algoritma je jednoznačno definiranje globalnog optimuma u multidimenzionalnom prostoru. [5] Osnovna ideja algoritma je simulacija jata koje navigira kroz okruženje prilagođavajući svoje ponašanje vodećim jedinkama. Inteligentni agenti koji sačinjavaju jato definirani su svojim pozicijama u prostoru, domeni problema. Svakim korakom iteracije algoritma pridružuje se nova brzina pojedine čestice, zavisna o kretanju cijelog jata, i novoodređena pozicija kao rezultat napredovanja. PSO započinje raspršivanjem čestica među nasumično odabranim točkama ulaznog problema. Tu svaka čestica ocjenjuje svoju lokaciju i glasa za inicijalni optimum. Rezultat metode nova Brzina vrijednost je dobivena prema formuli 5.1 .

$$v_i(n+1) = v_i(n)+(A*rand()*(pos_{iM}-p_i(n)))+(B*rand()*(pos_G-p_i(n)) \quad (5.1)$$

Oznake u jednadžbi su iduće:

$v_i(n+1)$ brzina čestice i u aktualnom $(n+1)$ koraku iteracije

$v_i(n)$ brzina čestice i u prethodnom (n) koraku iteracije

A težinski koeficijent za osobnu najbolju poziciju

B težinski koeficijent za globalnu najbolju poziciju

pos_{iM} najbolja pozicija koju je čestica i posjetila

$p_i(n)$ trenutna pozicija čestice i u (n) koraku iteracije

pos_G najbolja pozicija koju je posjetio član jata

Pozicija se ažurira jednostavnim pridodavanjem vektora brzine na aktualnu poziciju. Konstante A i B najčešće se uzimaju kao brojevi u intervalu od 0 do 4 uključeno. Preporuka je da se i maksimalna brzina koju čestica može postići ograniči na fiksnu

brzinu kako bi se postiglo preciznije manevriranje i mogućnost da se optimalno rješenje „preleti“ više puta bez da je identificirano.

Primjer pseudokoda za algoritam roja čestica na slici 5.1 .

5.1. Modifikacije

Sada ćemo razmotriti kako nam algoritam roja čestica može pomoći pri potrazi za izlazom iz labirinta. Prvi korak tog procesa definiranje je funkcije koja mora razlikovati koji su agenti bliže a koji dalje od izlaza. Ako se prisjetimo definicije problema koji rješavamo, sjetit ćemo se da eksplicitno nemamo dostupnu tu informaciju. Naime, problemu pristupamo kao da subjekti promatraju labirint iz njegove unutrašnjosti te zato ne mogu sa sigurnošću zaključiti gdje je izlaz. Nadalje, jednostavni labirinti nemaju nikakve oznake, sve točke labirinta osim izlaza ne sadrže nikakvu informaciju o dobroti trenutne pozicije. Vidljivo je da klasični pristup neće biti od koristi iz razloga što je domena problema, razmatrani labirint, siromašan informacijama.

Kako bi se uspješno definirala funkcija dobrote, korištena je pomoćna struktura koja predstavlja kolektivnu memoriju roja. Tamo se pohranjuje informacija o posjećenosti pojedine pozicije. Inicijalno je cijela populacija postavljena u točci koja simbolizira ulaz u labirint uz početne brzine kako zahtijeva sam algoritam roja čestica. Zatim se za svakog agenta u svakom koraku iteracije vrši odluka u kojem će se smjeru kretati uzimajući u obzir posjećenost pozicija na koje može krenuti. Susjedne pozicije evaluiraju tako što se za svako dostupno mjesto računa posjećenost, koliko je već agenata na tu poziciju do sada stalo. Manje posjećena mjesta imaju veću vjerojatnost odabira, pa se između smjerova kretanja vrši proporcionalna selekcija, slično jednom od mahanizama selekcije agenata genetskog algoritma. Za najmanje posjećenu poziciju računa se vektor smjera i on pridonosi u izračunu ukupnog vektora smjera za tu iteraciju. Ostale komponente kretanja uključuju vektor smjera prema najboljem agentu, vektor smjera prema prethodno najboljoj poziciji, inerciju populacije i utjecaj kretanja susjednih agenata. Detaljan opis značenja tih vektora dan je u poglavlju 7.

Pseudokod 2 Optimizacija rojem čestica

Ulaz: $problem, brojUzoraka$ – definicija problema i veličina populacije.

Izlaz: $p_{globalni}$ –najbolje pronađeno rješenje.

$populacija \leftarrow \varnothing$

$p_{globalni} \leftarrow \varnothing$

for ($i \in brojUzoraka$) **do**

  $p_{brzina} \leftarrow nasumicnaBrzina()$

  $p_{pozicija} \leftarrow nasumicnaPozicija()$

  $p_{najbolji} \leftarrow p_{pozicija}$

  if $cijena(p_{najbolji}) \leq cijena(p_{globalni})$ **then**

    $p_{globalni} = p_{najbolji}$

  end if

  $dodaj(populacija, p)$

end for

repeat

  for ($p \in populacija$) **do**

    $p_{brzina} \leftarrow osvjeziBrzinu(p_{brzina}, p_{globalni}, p_{najbolji})$

    $p_{pozicija} \leftarrow osvjeziBrzinu(p_{brzina}, p_{pozicija})$

    if $cijena(p_{pozicija}) \leq cijena(p_{najbolji})$ **then**

      $p_{najbolji} = p_{pozicija}$

      if $cijena(p_{najbolji}) \leq cijena(p_{globalni})$ **then**

        $p_{globalni} = p_{najbolji}$

      end if

    end if

  end for

until $\neg uvjetPostignut()$

return $p_{globalni}$

Slika 5.1: Optimizacija rojem čestica

6. Vizualizacija

Učenje promatranjem osnovna je metoda usvajanja znanja kod velikog broja živućih organizama na zemlji. Od kad smo se rodili počeli smo upijati znanja iz okoline upravo kroz to osjetilo i ono je naš primarni alat pri orijentaciji u prostoru. I upravo zato što je jedan od prvih načina percipiranja okoline te time i shvaćanja našeg svijeta, oslanjanjem na vizualna pomagala može ubrzati proces usvajanja novih ideja. Ovo dolazi još više do izražaja u slučajevima kad su te misli koje moramo formirati na određenoj razini apstrakcije. Usporedbe radi, zadovoljiva razina apstraktnog razmišljanja u prosjeku se kod djece formira tek oko 12 godina. Više od jednog desetljeća primarni izvor spoznaje čine auditivni, taktilni i vizualni podražaji kojim se polako gradi mogućnost percipiranja apstraktnih pojmova. Iz tog je razloga odlučeno da se alat za rad s algoritmima evolucijskog računanja obogati sučeljem za grafički prikaz, kao i potrebnim elementima vizualizacije da bi se ponašanje agenata prikazalo na jedan lagan i intuitivan način.

Postoji par razloga zašto je baš problem potrage za izlazom iz labirinta odabran u svrhu vizualizacije i koji su bili motivi iza odabira roja čestiza za algoritam rješavanja. Za početak, rješavanje labirinta je opće poznati zadatak i iz tog razloga korisnik ne treba uložiti dodatni trud poput upoznavanja sa klasifikatorima ili potrebom za funkcijskom optimizacijom. Labirint se također može prikazati kao matrica, koja razapinje prostor dozvoljenih i nedozvoljenih pozicija. Ta je terminologija u duhu one koja se koristi prilikom opisa problema rojeva čestica gdje se za rješenjem traga u domeni ili prostoru rješenja. Općenito, taj je prostor n-dimenzionalan pa je dvodimenzionalna reprezentacija idealna za vizualizaciju, ali i uhodavanje korisnika u taj koncept. Rojevi čestica spoj su jakog optimizacijskog aparata, konceptualno jasne ideje i vizualno interesantne pojave, kako u prirodi tako i u računalu.

7. Programsko ostvarenje

Ovo poglavlje opisat će izvedbu korisničkog sučelja, dati uvid u bitne dijelove koda te opisati ulazne parametre i njihov utjecaj na ponašanje roja.

7.1. Korisničko sučelje

Glavni prozor programa podijeljen je u dvija područja: prikaz labirinta na lijevoj strani i kontrolnu ploču na desnoj strani. Informacije vezane uz izgled labirinta, kretanje čestica i trenutna stanja memorije prikazuju se u prikazu labirinta jednom kad pokrenemo simulaciju. Desna strana prozora predstavlja sučelje prema korisniku koje omogućava različite akcije ovisno o stanju izvođenja programa. Inicijalno program je u stanju konfiguracije, gdje korisnik prolazi kroz konfiguraciju labirinta i algoritma.

Konfiguracija labirinta u prvom polju sadrži izbornik za odabir strategije izgradnje algoritma. U ovom radu podržan je samo slučajno generirani labirint s unaprijed poznatim ulazom. Taj generator treba iduće informacije: veličinu labirinta koju gradi, poziciju ulaza, kompleksnost labirinta, broj izlaza i širina hodnika. Veličina labirinta u ovoj implementaciji oslanja se na internu metriku generatora. Za skalarni ulaz x definiramo Sx kao odgovarajuću metriku dimenzionalnosti. Za dva unosa a i b preslikavaju se u Sa i Sb i vrijedi: ako a < b onda Sa < Sb. Korisniku će ti odnosi biti prikazani kao grublja ili finija granulacija elemenata labirina pošto je platno za iscrtavanje postavljeno na fiksnu širinu. Ovakva implemetacija veličine skriva od korisnika računalnu implementaciju ovog labirinta, konkretno matricu pozicija. Ona nosi specifične uvjete kako bi generiranje hodnika rezultiralo rješivim i smislenim labirintom. Ovakav dizajn automatizira proces postavljanja i olakšava konfiguraciju. Iduće u nizu je polje pozicije ulaza. Ono je dostupno kao izbornik sa stavkama: gore lijevo, gore desno, sredina, dolje lijevo i dolje desno. Nakon njega slijedi parametar za određivanje kompleksnosti. Generator definira kompleksnost kao stupanj konzistentnosti hodnika.

Što je kompleksnost veća, veći je i broj neprekinutih zidova. Kako opada kompleksnost, opada i broj komponenti zida u labirintu što rezultira "šupljikavim" (rupičastim) labirintom. Polje za unos broja izlaza omogućava unošenje većeg broja potencijalnih izlaza što može olakšati traganje. Posljednji konfiguracijski parametar označava širinu hodnika generiranog labirinta. Kao i u slučaju dimenzije labirinta ovaj parametar moguće je namjestiti preko pomične trake. Učinak je isti, samo ovaj put, zbog internog rada algoritma, širina hodika može ostati nepromijenjena za susjedne korake. Ako je generator u mogućnosti promijeniti širinu hodnika, a da pritom sva svojstva labirinta ostanu očuvana, promjena će biti vidljiva prilikom izdavanja zahtjeva za novi labirint.

7.2. Konfiguracija algoritma

Nakon što je prvi labirint postavljen, korisniku se omogućava pristup izborniku za konfiguraciju algoritma. Slično kao i kod generatora labirinta, okruženje pretpostavlja proširivost drugim algoritmima i zato ima izbornik za konfiguraciju budućih implementacija. Odabirom optimizacije rojem čestica postavlja se predložak parametara algoritma. On služi kao referentna točka prilikom konfiguracije, ali napomenimo još jednom da za jednake dimenzije labirinta isti parametri neće rezultirati jednako dobrim rezultatima. Korisnik može utjecati na iduće atribute: broj agenata populacije, maksimalna brzina kretanja agenta i broj susjeda koje pojedini agent vidi. Nakon toga slijedi utjecaj pojedine komponente kretanja pri izgradnji pomaka agenta u koraku iteracije. Ovi će parametri biti zasebno objašnjeni u narednim poglavljima. Za kraj korisnik može postaviti boju populacije kojom će se agenti iscrtavati i posebno izdvojiti boju agenta koji je proglašen kao najbolji u populaciji.
Broj agenata populacije intuitivno je jasan parametar: to je broj jediniki u populaciji koje će tragati za rješenjem. Maksimalna brzina agenta parametar je koji se unosi u slobodnom obliku. Naime, ova informacija je na samoj granici apstrahirane domene koju agent vidi ispod sebe i njegove percepcije brzine. Za trenutno podržane labirinte ovaj parametar najbolje je izabrati iz intervala od 1.0 do 10.0. Broj susjeda po agentu parametar je koji se uzima u obzir prilikom kretanja agenta. Topologija mreže je prstenaste strukture, gdje je za n susjeda njih n/2 lijevo od, a n/2 desno od promatranog agenta. Ukoliko je broj neparan, dodatni susjed bit će dodijeljen jednoj od strana.

Najbitniji dio konfiguracije čine vektori koji diktiraju u kojem će se smjeru kretati agent. Prisjetimo se, svakom agentu u poplaciji se inicijalno pridružuje točka u

domeni problema i slučajno odabrani vektor pomaka. Ta komponenta je bazni vektor koji se inicijalno uzima u obzir prilikom određivanja trenutnog smjera i kasnije služi za pohranu te iste komponente jednom kad se sustav uhoda. Klizni izbornici za sve parametre označavaju koliki postotak od najveće moguće brzine agenta otpada na pojedinu komponentu.

Prvi u nizu navedenih jest vektor utjecaja globalno najbolje jedinke. Kao i u klasičnom algoritmu rojeva čestica ova komponenta predstavlja udaljenost agenta od onog najboljeg. Jednom kad se odredi jedinični vektor smjera, njegov se iznos postavlja na predefinirani iznos i taj se vektor pridodaje ukupnoj sumi vektora smjera.

Još jedan karakteristični vektor rojeva čestica čini vektor utjecaja prethodno najbolje pozicije. Svaki agent pamti najbolju poziciju koju je ikada posjetio i iznos dobrote u tom trenutku. Smjer tog vektora računa se oduzimanjem trenutne pozicije od prethodno najbolje, a u izračunu magnitude uključuje se i faktor koliko je prethodna pozicija bila bolja od trenutne.

Sljedeće komponente nisu dio standardnog modela algoritma te su nastale kao produkt loših rezultata prethodnih testiranja. Specifičnosti problema zahtijevale su dodatne informacije prilikom kretanja. Te su informacije pohranjene u dodatni set vektora i medij su putem kojeg agenti percipiraju labirint. Prvi i najvažniji od njih je sigurno vektor utjecaja nepoznatih pozicija. Za tu komponentu agent najprije provjerava svoje lokalno susjedstvo. Ova faza izgradnje vekora smjera započinje analizom karakterističnih smjerova koje algoritam dobiva na zahtjev od labirinta. Zatim za te pozicije agent u memoriji gleda koji su iznosi posječenosti tih pozicija. Vrijednosti se zatim skaliraju na interval od 0 do 1 i koriste u proporcionalnoj selekciji kako bi se odredio idući smjer kretanja.

No samo favoriziranje lokalnog susjedstva pokazalo se kroz cijeli niz problema nepoželjno zbog opasnosti koju predstavljaju lokalni optimumi. Kako bi obogatili informaciju o smjeru neistraženih pozicija, uvest ćemo još jednu komponentu, vektor utjecaja kretanja populacije. Ta je komponenta vektorski zbroj smjerova svih agenata u populaciji i sadrži informaciju o inerciji roja. Ukoliko se roj počne gibati duž hodnika, agent može prilagoditi svoje kretanje kako bi se pridružio ostatku populacije. U trenutku kad se agent sudari sa zidom, njegov se vektor smjera kretanja postavlja na nulu. Nedostatak te komponente bit će zabilježen pri idućem izračunu ovog vektora i

ako njegov iznos padne na nulu, agenti će se naći u poziciji traganja za novim smjerom kretanja.

Vektor kretanja populacije značajan je prilikom kretanja roja jednom kad je populacija stabilna i usmjerena prema zajedničkom cilju. No taj je mehanizam poprilično krut i uzima u obzir cijelu populaciju. Ukoliko želimo fleksibilniju razmjenu informacija među agentima, možemo se poslužiti lokalnim susjedstvom prilikom izgradnje prikladne komponente. Vektor utjecaja susjedstva čuva upravu tu informaciju. Slično vektoru kretanja populacije, ova se komponenta dobiva zbrajanjem vektora smjera agenata lokalnog susjedstva opisanog na početku poglavlja. Primijetimo da se i ovim mehanizmom informacija kretanja jednog agenta može stići do svakog drugog. Iteraciju po iteraciju preko poznanstva agenata dijeljenja informacija bit će propagirana kroz populaciju, ali što je još zanimljivije, ista će se komponenta modificirati trenutnim saznanjem svakog agenta prije nego što se proslijedi i time postati sve kvalitetnija.

Mehanizmi zaustavljanja podržanih u ovom okruženju uključuju broj proteklih iteracija, vremensko ograničenje i zaustavljanje prilikom pronalaska izlaza od strana n agenata. Pojedini mehanizmi zaustavljanja mogu se vidjeti na slikama:

Zadnji korak prilikom donošenja odluke o vektoru smjera agenta je onaj o disperziji populacije. Za svaku iteraciju algoritam računa centar mase roja. Ta se točka koristi u analizi raspršenosti populacije tako da se izračuna udaljenost agenta od te zamišljene točke. Parametar najvećeg radiusa populacija govori koliko ta udaljenost može biti velika prije nego što se uključi ovaj mehanizam. Ukoliko je udaljenost čestice od središta roja veća od predefinirane vrijednosti, agent je izvan radiusa i njemu se pridjeljuje komponenta koja ispravlja njegovu putanju usmjeravajući ga prema trenutnom centru. Parametar nakon toga, postotak mirne zone, označava područje unutar radiusa populacije koje neće utjecati na kretanje. Ako je kojim slučajem agent preblizu centru mase roja, ova će vrijednost rezultirati odbijanjem agenta u smjeru suprotnom od spojnice agent-središte.

Pri kraju izbornika gumb je za dodavanje konfiguriranog algoritma u red za izvođenje. Trenutna inačica programa omogućuje konfiguraciju i dodavanje do dva algoritma rojeva čestica koja bi se mogla pralalelno izvršavati. Iz tog su razloga gumbu za dodavanje algoritma pridjeljena dva dodatna gumba za praćenje konfiguriranih algoritama. Ako nema konfiguriranih algoritama, gumbi su inaktivni, signalizirajući tu činjenicu

korisniku. Prilikom pritiska gumba za dodavanje algoritma on se dodaje u red za izvođenje i pali se njemu pridružen gumb. Ako se želi ukloniti algoritam iz reda za izvođenje, dovoljno je pritisnuti na njemu pridružen gumb i on se oslobađa iz sustava. Simulaciju nije moguće pokrenuti ako nema niti jednog konfiguriranog algoritma ili labirinta.

Posljednja opcija koju korisnik može uključiti je iscrtavanje kolektivne memorije algoritma. Kako okruženje podržava samo algoritme koji koriste internu memoriju za pohranu posjećenih pozicija, ova zastavica propagira se svim algoritmima i ukoliko je uključena, omogućava iscrtavanje memorije koju vide agenti pri donošenju svojih odluka. Na zaslonu se prikazuju kao nijansa boje populacije čiji intenzitet raste kako preko pojedine pozicije prijeđe više agenata.

Pošto je kretanje agenata centralna mehanika ovog algoritma, slijedi njena implemntacija iz razreda Agent koji je prikazan na slici 7.1 .

7.3. Primjeri pokretanja

U ovom su potpoglavlju prikazani neki od ishoda ponašanja algoritma prilikom izmjene prethodnih parametara. Istaknuta će ponašanja, radi preglednosti, biti prikazana na praznom labirintu, bez zidova i izlaza. Ponašanje ostaje zadržano za primjere kada se zidovi uvedu, ali je tijek izvođenja simulacije znatno dinamičniji.

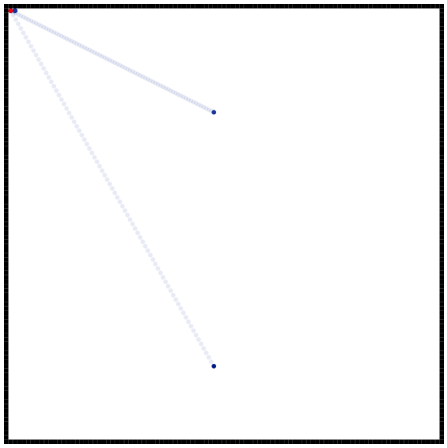
Slijedi pregled istaknutih konfiguracija uz popratne komentare. Svi će primjeri imati prikazanu kolektivnu memoriju kako bi se lakše pratila trajektorija agenata na slikama.

7.3.1. Kretanje u početnom smjeru

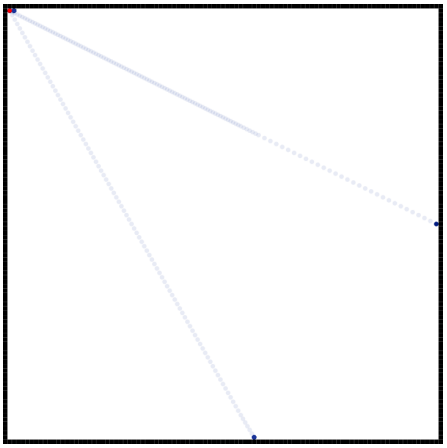
Počet ćemo od najjednostavnije konfiguracije. Broj agenata stavit ćemo na 5, najveću brzinu koju agent može postići na 10, dok će ostali parametri biti postavljeni na nulu. Rezultat je kretanje koje čestica opisuje vođena samo nasumično odabranim vektorom smjera. Primjer takvog kretanja prikazan je na slici 7.2. Prisjetimo se, agent prilikom kolizije u potpunosti gubi svoju orijentaciju i oslanja se na neki drugi mehanizam prilikom rekonfiguracije svog kretanja. Kako takav mehanizam nismo omogućili, agent ostaje stajati uza zid.

```
@Override
public IVector move(IVector directionVector, ILabyrinth labyrinth, ISharedMemory popMem) {
 double beforeMovMagnitude=directionVector.norm();
 List<IVector> positions = labyrinth.generatePosition(this.curPos, directionVector);
 double afterMovMagnitude=directionVector.norm();

 if(beforeMovMagnitude!=afterMovMagnitude)
  this.colidedRecently=true;
 else
  this.colidedRecently=false;

 if (!positions.isEmpty())
  positions.remove(0);
 for (IVector position : positions) {
  this.qualityOfPositionsFound += popMem.steppedOn(position);
  this.totalPositionsPassed++;
 }

 // remember n-1 position
 this.prevPos = curPos;

 // process for this iteration: new position, and fitness
 if (!positions.isEmpty())
  this.curPos = positions.get(positions.size() - 1); // else stay at same position
 double nextPosFitness = qualityOfPositionsFound / totalPositionsPassed;
 this.fitness = nextPosFitness;

 // previous best position
 if (this.bestPrevFitness <= this.fitness) {
  this.bestPrevFitness = fitness;
  this.persBestPrevPos = curPos;
 }

 // is finished
 this.isFinished = labyrinth.isFinished(curPos);

 /// direction vector
 this.curDirVector = directionVector;

 return this.curPos;
}
```

Slika 7.1: Kretanje agenta

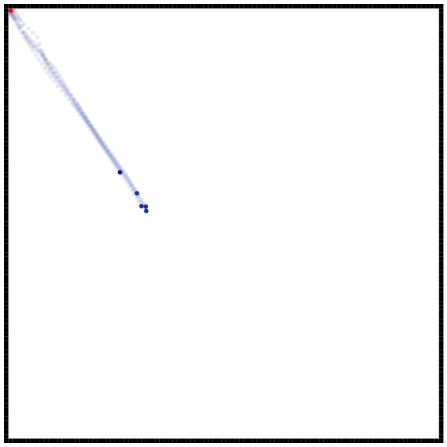


(a) Rana faza optimizacije.

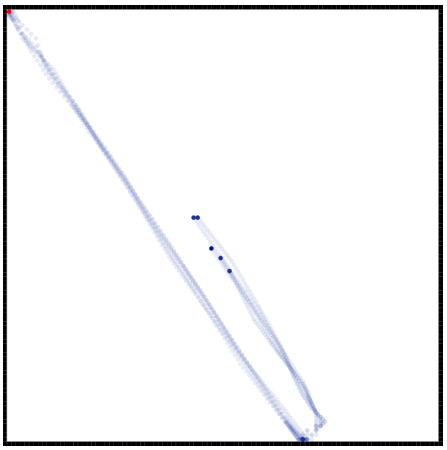


(b) Kasna faza optimizacije.

Slika 7.2: Jednostavno kretanje.



(a) Rana faza optimizacije.

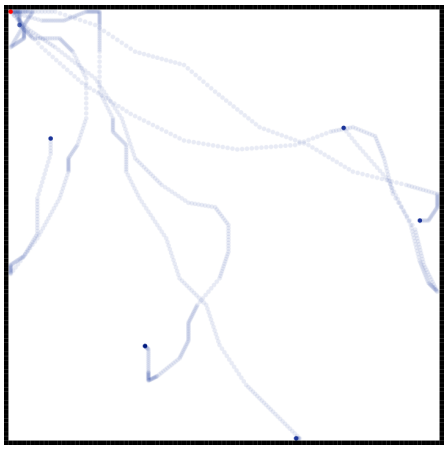


(b) Kasna faza optimizacije.

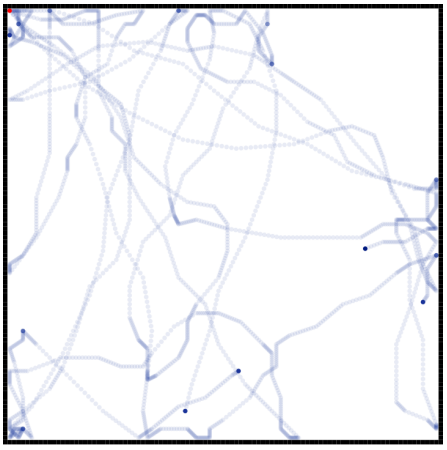
Slika 7.3: Složeno kretanje.

7.3.2. Utjecaj globalnog optimuma i utjecaj prethodno najbolje pozicije

Obogatimo sada raspoložive informacije pozicijama najboljeg agenta u populaciji i dozvolimo da agent razmatra svoje prethodno najbolje pozicije. Ovaj mehanizam dodaje još malo dinamike u sustav. Agenti sada slijede svoga vođu, a ako dođe do kolizije, prethodno najbolja pozicija potaknut će ih da se počnu gibati u novom smjeru. Primjer takvog ponašanja vidimo na slici 7.3.



(a) Rana faza optimizacije.



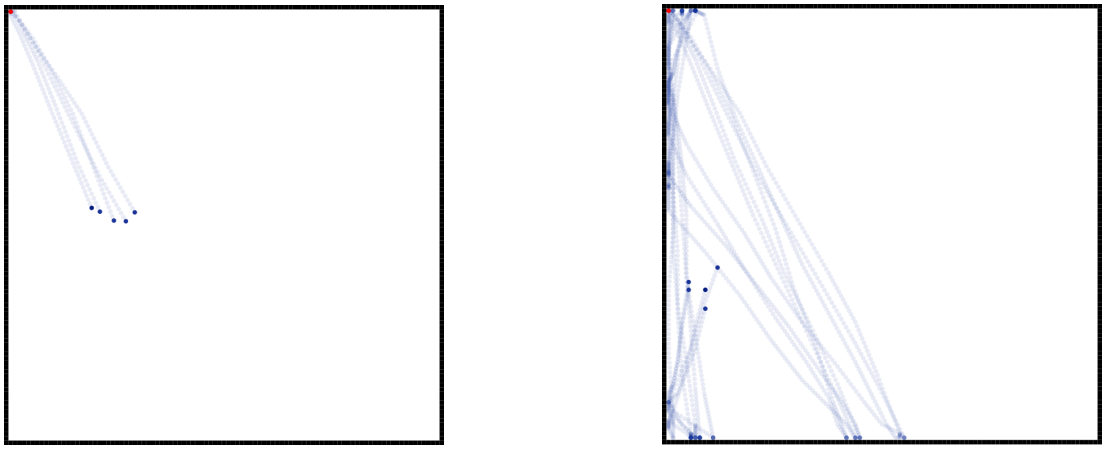
(b) Kasna faza optimizacije.

Slika 7.4: Istraživanje.

7.3.3. Utjecaj neistraženih pozicija

U ovom primjeru uvest ćemo prvu specifičnu komponentu gibanja. Slika 7.4 a) prikazuje kako mali faktor istraživanja utječe na kretanje algoritma. Vidimo da agent pokrije puno veću površinu zato što izbjegava do sad posjećena područja. Slika 7.4 b) objedinjuje sve tri do sad korištene komponente kroz koje se polako nazire kompleksnije ponašanje.

```
@Override
private IVector getDirVector(IAgent agent) {
  List<IVector> neightList = getLegalNeighbourPositions(agent.getCurrentPosition());
  List<Double> fitList = calculatePositionsFitnessValues(neightList);

  sortDescneding(neightList, fitList);
  normaliseFitness(fitList);

  IVector nextPosition = rouletteWheleDirection(neightList, fitList);
  IVector nextDirection = null;
  try {
   nextDirection = nextPosition.sub(agent.getCurrentPosition());
  } catch (IncompatibleOperandException ignorable) {
   // this code won't run since the vector dimensions are
   // consistent throughout the algorithm
  }
  nextDirection.scalarMultiply(this.unvisitedInfluence);
  return nextDirection;
 }
```



(a) Rana faza optimizacije.

(b) Kasna faza optimizacije.

Slika 7.5: Utjecaj populacijske komponente smjera.

7.3.4. Utjecaj populacije

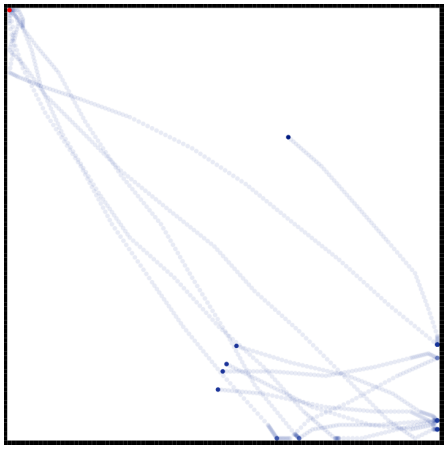
Utjecaj populacije usrednjuje vektore gibanja agenata i koristi ga kao reprezentaciju kolektivnog znanja. Na slici 7.5 a) populacija raspolaže informacijama najboljeg agenta, prethodno najbolje pozicije i utjecajem populacije, dok se na slici 7.5 b) spomenutim komponentama pridodaje još i utjecaj neistraženih pozicija. U ovom slučaju vidimo da iako raštrkani, kretanje agenata ipak djeluje usmjereno i strukturirano.

7.3.5. Utjecaj susjeda

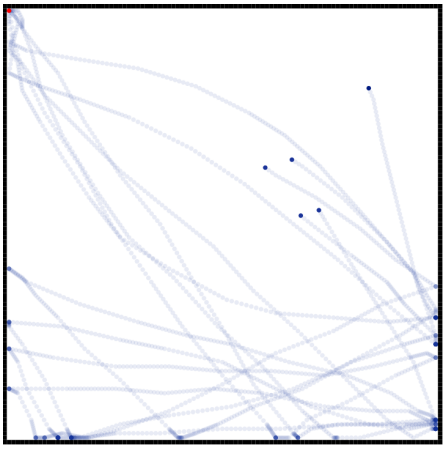
Utjecaj lokalnog susjedstva agenta nešto je slobodnija inačica utjecaja populacije. Agenti više ne putuju paraleno jedan s drugim, već migriraju u smjeru koji odgovara većini. To ponašanje možemo vidjeti na slici 7.6 a) gdje utjecaj susjedstva djeluje uz informacije najboljeg agenta i najbolje prethodne pozicije, te na slici 7.6 b) gdje su prilikom konfiguriranja uključeni svi do sad korišteni parametri.

7.3.6. Ograničenje na disperziju populacije

Mehanizam reguliranja disperzije populacije pod pravim je okolnostima odličan dodatak strategije kretanja pri konfiguriranju algoritma. Za probleme s relativno uskim hodnicima jači faktor stezanja populacije rezultirat će uspješnijim kretanjem. Agent u ovakvim okolnostima razmjenjuje više informacija s ostatkom populacije i na njega samog više utječu povratne informacije koje dobiva od roja. Takvo ponašanje je poželjno u labirintima uskih hodnika gdje roj nema dovoljno prostora za manevriranje. No za labirinte gdje su hodnici dovoljno široki, gdje su kolizije sa zidovima rijetka

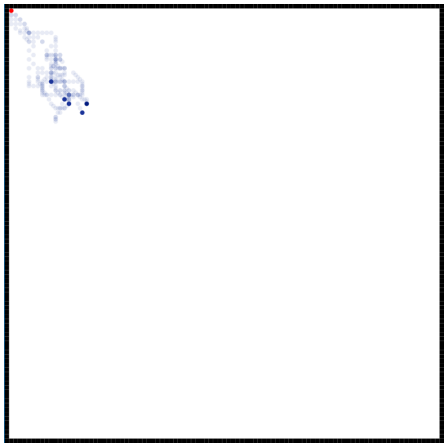


(a) Rana faza optimizacije.

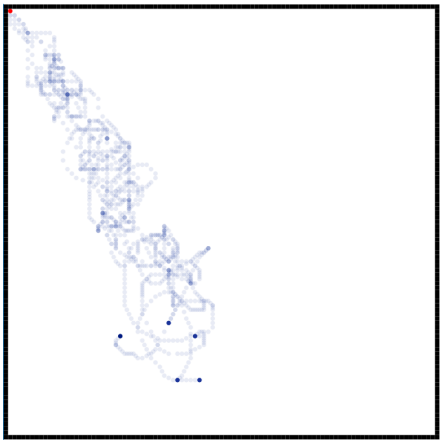


(b) Kasna faza optimizacije.

Slika 7.6: Utjecaj komponente smjera susjeda.



(a) Rana faza optimizacije.



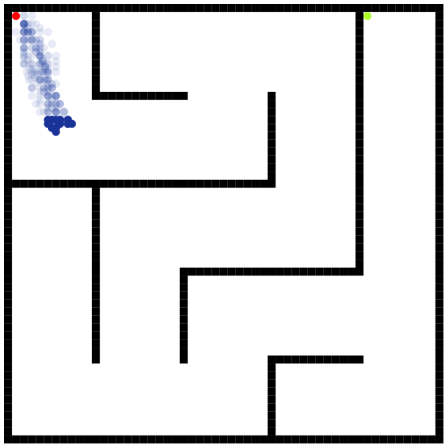
(b) Kasna faza optimizacije.

Slika 7.7: Utjecaj ograničenja disperzije.

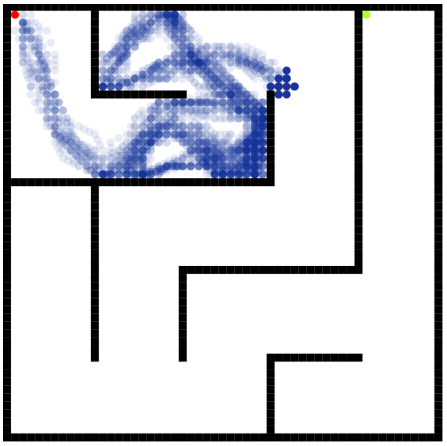
pojava i gdje pokrivanje šireg područja domene u koraku iteracije znači ubrzanje pri potrazi za rješenjem, nije nam u interesu držati populaciju previše blizu. Ovaj primjer odlično ilustrira problematiku postavljanja početnih parametara i posljedice kakve loši parametri mogu imati na performanse. Ovo je ponašanje prkazano na slici 7.7 .

7.4. Izdvojeni primjeri

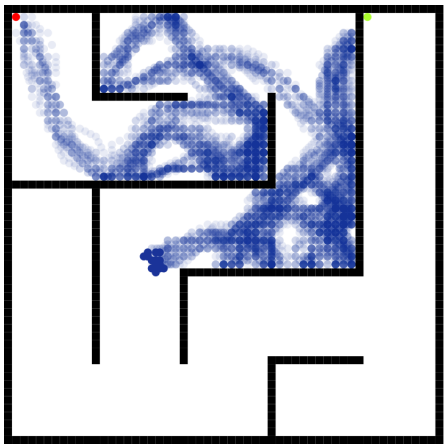
Kako bi proces konfiguriranja parametara olakšali korisniku, poželjno je koristiti paralelno dva optimizacijska procesa sličnih konfiguracija i promatrati kako se ponašanje roja mijenja dok simulacija napreduje s vremenom. U nastavku su prikazani neki specifični slučajevi izvođenja uz njihove konfiguracije i karakteristično ponašanje.



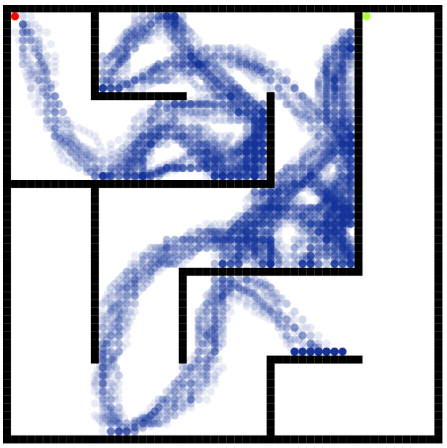
Slika 7.8: Pokretanje 1 - Faza 1.



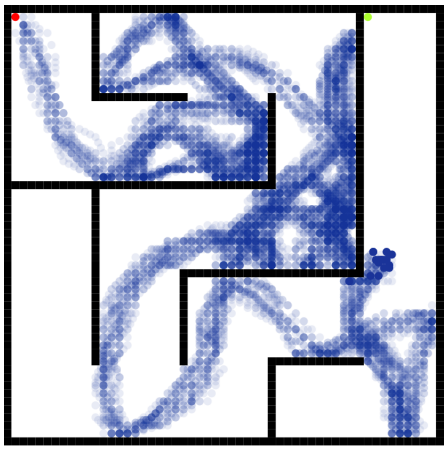
Slika 7.9: Pokretanje 1 - Faza 2.



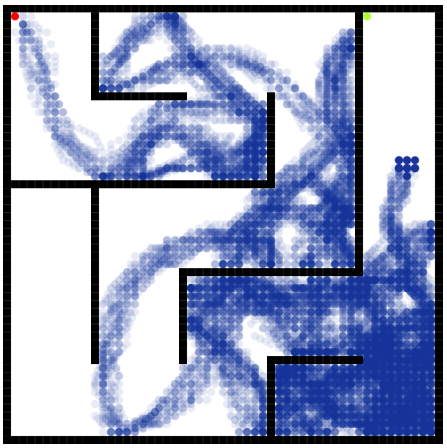
Slika 7.10: Pokretanje 1 - Faza 3.



Slika 7.11: Pokretanje 1 - Faza 4.

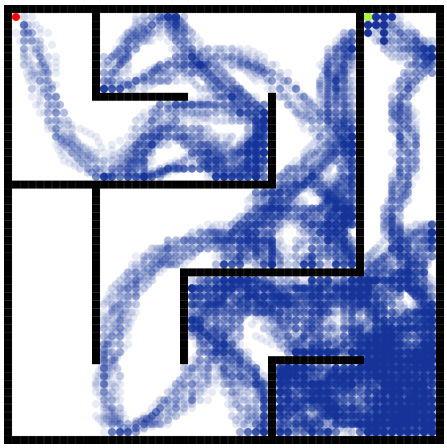


Slika 7.12: Pokretanje 1 - Faza 5.

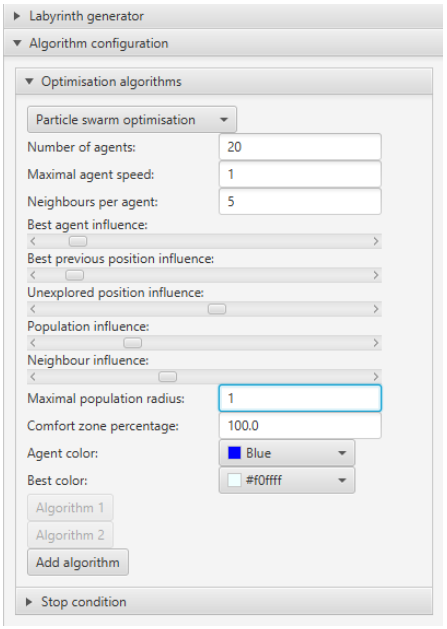


Slika 7.13: Pokretanje 1 - Faza 6.

Prvi slučaj razmatra optimizacijski proces jednog algoritma u labirintu srednje dimenzije s širokim hodnicima. Slike 7.8 do 7.14 prikazuju jedan od mogućih trajektorija

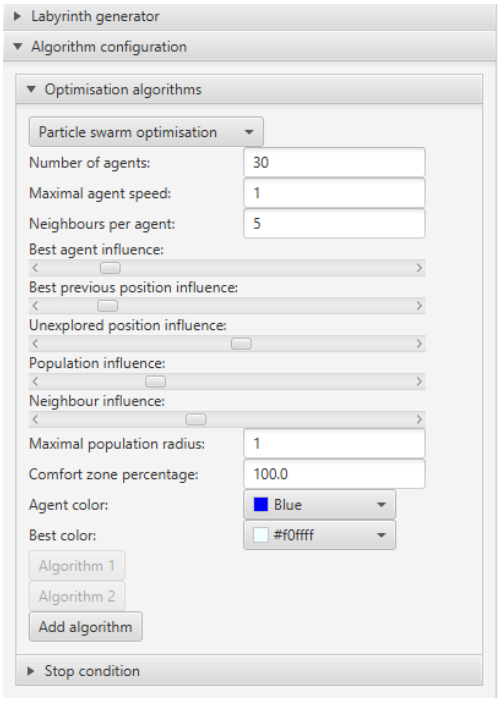


Slika 7.14: Pokretanje 1 - Faza 7.

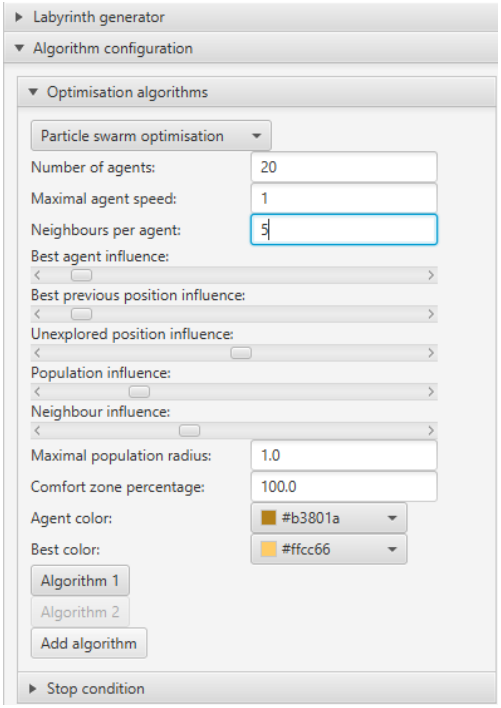


Slika 7.15: Pokretanje 1 - Konfiguracija

kretanja populacije koja završava uspješnim pronalaskom izlaza. Parametri konkretnog labirinta dani su na slici 7.15. Drugi slučaj prikazuje dva paralelno pokrenuta algoritma s sličnim konfiguracijom parametra, uz različite boje radi lakšeg razlikovanja. Tijek izvođenja slijed izvođenja zabilježen je na slikama 7.18. do 7.21. Primjetimo kako smeđe označena populacija preuzima vodstvo i prodire duboko u labirint samo kako bi na zadnjoj promatranoj iteraciji promijenila strategiju, krenula natrag i mimoišla se sa zelenom populacijom koja se tek tada odmiče od ulazne točke labirinta.



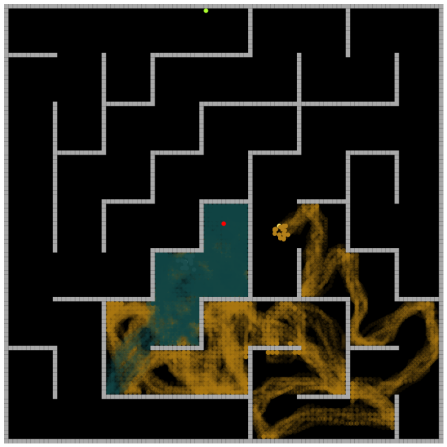
Slika 7.16: Pokretanje 2 - Konfiguracija prvog algoritma.



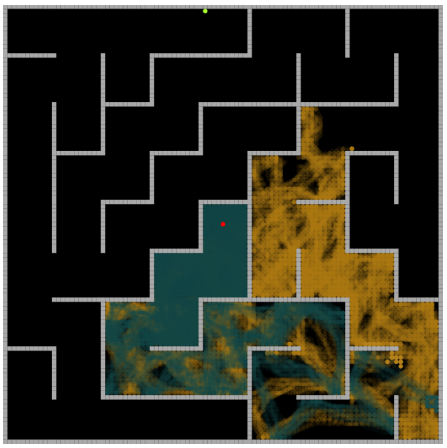
Slika 7.17: Pokretanje 2 - Konfiguracija drugog algoritama.

Slika 7.18: Pokretanje 2 - Faza 1.

Slika 7.19: Pokretanje 2 - Faza 2.



Slika 7.20: Pokretanje 2 - Faza 3.



Slika 7.21: Pokretanje 2 - Faza 4.

8. Zaključak

U okviru ovog rada pokazano je kako algoritam optimizacije rojem čestica može uspješno pronalaziti izlaze iz labirinta različitih dimenzija i složenosti. Rezultirajuća programska potpora sadrži prilagođenu inačicu algoritma rojeva čestica, specifičnu za problem labirinta, ali i njemu sličnim zadacima. Razvijena je i potrebna logika za komunikaciju algoritma i korisničkog sučelja u višedretvenom okruženju. Uz to, ponašanje algoritma vjerno je predočeno kroz grafičko sučelje i promjene ulaznih parametara odražavaju se na ponašanje algoritma u skladu s očekivanjem. Zbog takve povratne veze između ulaznih parametara i grafički predočenog ponašanja, korisnik može lakše razviti intuiciju za prilagodbu parametara algoritma konkretnom problemu. Prvi rezultati su obećavajući, ali potrebno je provesti dodatno ispitivanje na većem uzorku korisnika kako bi se njegova primjena u edukacijske svrhe pokazala valjanom. Daljnji rad na ovom projektu može se fokusirati na implementaciju novih domena, poput labirinata koji svoju strukturu mijenjaju kroz vrijeme, implementacija novih algoritama čije ponašanje može biti opisano razvijanom okolinom ili daljnje poboljšanje ponašanja optimizacije rojem čestica. Dodatne funkcionalnosti simulacijskog okruženja poput sučelja za crtanje proizvoljnog labirinta, izbornika za kontrolu tijeka simulacije ili mijenjanje parametara algoritma za vrijeme njegovog izvođenja mogu poboljšati kvalitetu korištenja simulatora. Ishod rada doprinio je širem kontekstu razmatranja algoritama rojeva čestica, dajući drugačiji pogled na već poznatu problematiku labirinta, dokazujući mogućnost njezine realizacije i primjenjivosti.

LITERATURA

[1] Walter D. Pullen. Maze classification.

[2] Nathaniel Johnston. Maze generation algorithm.

[3] M. Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.* FER, Sveučilište u Zagrebu, 2013.

[4] Shi Y. Kennedy J., Eberthart R.C. *Swarm intelligence*. Morgan Kaufmann Publishers, 2001.

[5] J. Brownlee. *Clever Algoriths: Nature-Inspired Programming Recipes*. 2001.

Vizualizacija ponašanja algoritma rojeva čestica prilikom rješavanja problema labirinta

Sažetak

Opće je poznata činjenica da ljudi nova znanja najbrže savladavaju kroz aktivno djelovanje prilikom njegovog usvajanja. Kako bi se olakšao proces usvajanja koncepata inteligencije rojeva čestica, u okviru ovog rada izgrađena je programska potpora za simulaciju optimizacijskih procesa rojeva čestica. Razvijeno okruženje na problemu potrage za izlazom iz labirinta pruža korisniku uvid u ponašanje algoritma roja čestica. Rad također obuhvaća motivaciju iz ovog zadatka, kratki uvod u algoritme evolucijskog računanja, posebno algoritama rojeva čestica uz diskusiju o preinakama izvornog algoritma kako bi poslužio za rješavanje ovog problema. Za kraj daje se opis ulaznih parametara, njihovo značenje za algoritam i primjeri izvođenja.

Ključne riječi: Evolucijsko računanje, inteligencija roja čestica, simulacija, umjetna inteligencija, razvojna okolina

Visualisation of particle swarm optimisation navigation through a labyrinth

Abstract

It is a well known fact that humans acquire new knowledge with greater ease if the learning process includes enough effort invested, as well as a moderate amount of interaction with the subject we are eager to understand. This thesis focuses on implementing a simulation environment for simulating evolutionary computation algorithms with the goal of better understanding it's mechanics. The simulations displays a labyrinth and a population of configurable agents capable of solving the task ahead. It is up to the user to understand the parameters with which the algorithm works and adapts them to the specific layout. Besides the simulation, the paper includes initial motivation for engaging with this topic, a brief overview of the algorithms available, and explains the particular parameters the algorithm utilises.

Keywords: Evolutionary computing, swarm intelligence, simulation, artificial intelligence, development environment