Minimum motion planning

Projekat u okviru kursa računarska inteligencija
Matematički fakultet

David Živković
mi20098@alas.matf.bg.ac.rs

Matija Stanković
mi20171@alas.matf.bg.ac.rs

Februar 2024

Contents

1 Uvod

Grafovi su veoma značajni prilikom rešavanja širokog spektra problema koji su sveopšte prisutni u svakodnevnom životu. Nalaze svoje primene na mestima gde možda i nije očigledno da se mogu primeniti. Rešavaju probleme navigacije, nalaženja puteva (najkraćih i ostalih), pa sve do generisanja rasporeda časova. Neki od primera koje rešavaju su: modelovanje povezanosti korisnika na društvenim mrežama, pravljenje modela molekula u hemiji i mnoge druge, u računarstvu se koriste mahom za nalaženje najkraćih puteva, istraživanja podataka, itd. Zbog svoje raznolike upotrebe su godinama unazad razvijani različiti algoritmi koji se bave rešavanjem praktičnih, pa tako i teorijskih problema. Shodno svemu dosad navedenim, može se zaključiti da grafovi predstavljaju pogodnu, verovatno čak i idealnu, strukturu podataka za modelovanje raznovrsnih problema.

Iako su grafovi široko primenljivi prilikom pristupanja, sagledavanja i rešavanja problema, nisu svi algoritmi podjednako primenljivi na sve probleme. Prevashodno je neophodno ostvarivanje odredjenih preduslova da bi taj algoritam bio ispravno primenljiv. Recimo čuveni Dijkstrin algoritam nije primenljiv ako graf nije težinski. Pri rešavanju problema nalaženja najkraćih puteva je primenjiv, ali toga ne čini uvek pogodnim, na primer ako imamo još dodatnih informacija, deluje kao da je bruteforce algoritam u odnosu na recimo algoritam A*, koji radi mnogo efikasnije.

Prethodno navedeni algoritmi se koriste prilikom rešavanja problema pretrage, iliti nalaženja (idealno, najkraćih) puteva. Iako se čini da su njima problemi rešeni u zavisnosti od zahteva, oba algoritma imaju svojih poteškoća. Problem je uglavnom što su algoritmi primenljivi samo ako postoji jasno rešenje, odnosno, ako na putu od početka do cilja ne postoji nekakav vid prepreke koji će onemogućiti postojanje putanje koja povezuje bilo koja dva čvora (početni i krajnji). Pri čemu se pod preprekom misli na onu koja skroz zatvara sve puteve. Ako se prepreka može zaobići, lako se može dati čvor, na kom se nalazi prepreka, označiti za nevažeći, odnosno da se mora zaobići (ne sme se na njega nikad kročiti). Postojanje takvog tipa problema, gde se do rešenja neće stići čak i obilaženjem prepreka, je upravo poslužio kao motivacija za pisanje ovog rada (kao i nalaženje algoritma koji će pokušati to da reši).

Prvo će biti opisani neki već postojeći pokušaji rešavanja datog problema, onda potom i naše rešenje, koje se prilikom testiranja pokazalo kao u potpunosti zadovoljavajuće. Što se tiče samog korišćenja metoda i ideja drugih problema nisu bili razmatrani sve do samog završetka našeg algoritma.

2 Opis problema

Sam naziv problema je "Minimum (graph) motion planning". Kao što se po imenu može i pretpostaviti, neophodno je naći minimalan graf koji će zadovoljiti dato rešenje. Po prethodnom opisu motivacije odabira problema za rešavanje se lako može pretpostaviti da je reč o težinskom grafu koji je (ne)usmeren, gde je neophodno naći najkraću cenu puta (posmatrano iz ugla težina grana datog grafa). Doduše sam opis problema (kao i mnogih sličnih) se fokusira na usmeravanje robota unutar koordinatne mreže, pa su težine svih grana 1, stoga je zapravo cilj naći rešenje koje zahteva što manje koraka.

Opis problema glasi ovako: Dat je graf G, početni čvor s, sa kog robot kreće, krajnji čvor t, koji predstavlja cilj do kog robot treba stići. Dat je i skup čvorova W, pri čemu je W podskup od G, gde W predstavlja skup svih prepreka koje se nalaze unutar tog početnog grafa. Rešenje predstavlja listu poteza koje je neophodno uraditi ne bi li se došlo do (unapred zadatog) cilja. Jedan potez može predstavljati pomeranje robota na susedni čvor, ili pomeranje prepreke na susedni čvor. Pri čemu treba voditi računa da taj susedni čvor treba biti "slobodan", odnosno da se na njemu ne nalazi ni robot, a ni prepreka, kao i to da nije ciljni čvor kako ne bismo došli u situaciju da napravimo nerešiv problem. Rešenja se medjusobno uporedjuju po broju koraka koliko je bilo neophodno ne bi li se stiglo do cilja. Odnosno rešenje čija je lista pomeraja najkraća predstavlja najbolje rešenje.

Cilj ovog rada je pokušaj pronalaženja opmitizacinog algrotima, tj algoritma koji će minimizovati/maksimizovati ciljanu funkciju (što je broj koraka u našem slučaju). Uzevši ovo u obzir, algoritam ne mora uvek naći optimalno rešenje, već postoji i mogućnost za nalaženjem rešenja koje je "približno" optimalnom, ali dovoljno dobro.

2.1 Opis rešenja

Algoritmi koji će biti korišćen su:

- Brute force (sa manjom optimizacijom)
- Inteligencija rojeva (konkretno u ovom slučaju, kolonija mrava)
- VNS

2.1.1 Brute force

Algoritam iscrpne pretrage je pomalo problematičan jer je sistematičnost preobimna čak i za primere gde se nalazi i manje od 10 čvorova. Stoga je odlučeno da se nasumično pomera prepreka ili robot u nadi da će se vremenom robot naći na svom mestu. Verovatnoća pomeranja robota je značajno veća od verovatnoće pomeranja prepreka. Ako treba pomeriti prepreku, nasumično (uniformno) se bira koja će se prepreka pomeriti na dozvoljeno mesto.

2.1.2 Optimizacija kolonijom mrava

Konkretno kod kolonije mrava imamo situaciju u kojoj mravi treba da stignu do hrane (cilja), a da pritom mapiraju najkraći mogući put, kako bi maksimizovali svoj učinak. Ukoliko je putanja jednostavno pravolinijska, i još na sve to jedina, mravi će bez problema dolaziti do cilja. Šta će se desiti ukoliko dodje do neke prepreke na postojećem putu? Pošto je prepreku nemoguće otkoniti (to je pretpostavka), mora se doći do novog (zaobilaznog) puta. Problem rešavanja takvog problema uglavnom nije deterministički, iz prostog razloga što je nemoguće unapred znati koja od mogućih putanja predstavlja najbolje rešenje problema. Za početak je neophodno navesti ključne pojmove i neophodnosti algoritma.

- d_{ij} - udaljenost izmedju dva susedna (i, j) čvora u grafu
- τ_{ij} - količina feremona na grani (i, j)
- n - broj jedinki (mrava), pri čemu svaki stvara svoju putanju
- u svakom koraku verovatnoća odlaska odredjenom granom od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ je srazmero formuli $(\tau_{ij})^a(d_{ij})^{-b}$
- Feromon na svakoj grani isprava po formuli: $\tau(1-\rho)\tau$

Jedinke započinju svoj put od polaznog čvora ka ciljnom. Svaki pojedinac ažurira svoju tabelu rutiranja, što je u ovom slučaju lista koja predstavlja putanju kojom se kretao do cilja, i komunicira sa ostalima. Sama ideja koja stoji iza algoritma (optimizacije) glasi: "ako ideš ka ciljnom čvoru putem kojim je prethodni već bio ranije, dobićeš savet od njega gde bi trebalo ići". Moguće je da su se prethodne putanje pokazale kao loše, što se olako može dogoditi uzevši u obzir da se unapred ne zna koje su putanje dobre, već se do njih dolazi isprobavanjem. Ukoliko do toga dodje, valjalo bi uticaj pojedinca (validnost saveta), smanjiti ispravanjem, odnosno kako vreme prolazi.

Uprkos prethodno opisanim metodama, svakako postoji bojazan da se napravi takozvani "krug (putanja) smrti" koju će svi mravi pratiti, uprkos činjenici da ih vodi u neizbežnu smrt. Moguće je vremenom rešiti se takve situacije, pri čemu je neophodno "odoleti istaknutom stimulativnom feremonskom tragu". Svakako je poželjno uvek je izbegavati.

3 Implementacija

Nakon što je problem definisan, potrebno je još opisati koja svojstva je moguće dodeliti grafu. Konkretno za prilike razvijanja ovog algoritma i dolaženja do rešenja problema korišćeni su nasumično generisani grafovi. Nasumično je generisan broj prepreka i njihove pozicija, takodje su i početna i krajnja pozicija nasumično postavljene. Ideja koja je podstakla ovakvu odluku je bojazan da bi korišćenje fiksnog grafa dovelo do toga da se rešenje previše prilagodi datom

izgledu grafa (čak iako bismo menjali pozicije prepreka). Pored samog prilagodjavanja rešenja, problem koji smo se trudili da izbegnemo je bio ljudske prirode (da ne posmatramo problem kroz konkretan primer). Naime, tragili smo za što opštijim rešenjem, što bismo uspevali da dobijemo ukoliko bismo rešavali što više različitih nasumičnih grafova. Do datih grafova smo došli korišćenjem Python-ove biblioteke networkx. Biblioteka mnogo olakšava pravaljenje kao i vizuelizaciju grafova.

Korišćena su dva različita izgleda grafa. Jedan je koordinatna mreža (takozvani grid), dok je drugi običan graf koji nema konstantno definisanu formu. Grid je korišćen kako bi bliže simulirao konkretnu instancu problema rešavanja ako je lavirint u pitanju. Dok je drugi graf izabran da nema previše povezanosti medju čvorovima kako bi se smanjile opcije u svakom koraku i time forsiralo da se put može naći jedino pomeranjem prepreka. Naime, neretko se desi da se nasumično generiše graf koji je rešiv odmah, bez pomeranja prepreka (moguće je dobiti bolje rešenje ako se ipak pomeri neka prepreka, što će takodje biti razmatrano).

Sama veličina i povezanost grafova, kao i izgled je moguće prilagodjavati po želji. Feromoni koji se inicijalno nalaze na granama su nasumično postavljeni, ali njihova vrednost je svakako mala tako da nema uticaja na feromone koje će mravi svakako ostavljati kasnije. Takodje je moguće odrediti učestalost prepreka, u smislu koliko će ih biti, što smo nasumično birali broj od 1 do polovine ukupnog broja grafova. Taj broj se činio kao najbolji jer je preko toga postojala bojazan da se dodje do toga da bude toliko prepreka da je problem praktično nerešiv (ne samo za ovaj algoritam). Pored izgleda samog grafa jedina promenljiva stvar u algoritmu je broj iteracija (i veličina kolonije), što bi valjalo skalirati shodno veličini grafa. Osim toga je moguće i štelovati same parametre optimizacije putem rojeva, stepen feromona i ostalo. Ostale stvari koje su prosledjivane funkciji su dodatne strukture kojim se refaktorisao kod čime je postao čitljiviji.

3.1 Bruteforce algoritmi

Iscrpni algoritam (bruteforce) je implementiran tako da radi nasumično. Zamisao je da je "dovoljno loše" ako radi ovako, nego ako bismo pokušavali sistematično da isprobavamo sve mogućnosti na kojima bi mogla da se pomeri svaka prepreka i posle svake provere ispitati da li postoji put i kakvog je kvaliteta. Glavni razlog odustajanja od ovog pristupa je postojanje mogućnosti da se do rešenja dolazi samo ako se pomeri više od jedne prepreke više puta, što je praktično nemoguće sve ispitati čak i za grafove sa malim brojem čvorova (jer bi trajalo predugo, a i sam postupak dolaženja do sistematičnog pretraživanja je problematičan sam po sebi).

Sama ideja algoritma je vrlo prosta. Sa odredjenom verovatnoćom će biti izabrano da se na nasumično mesto (iz svoje okoline) pomeri neka prepreka ili robot. Robot ima prednost pošto je verovatnoća 0.8 da se on pomera. Ideja koja stoji iza toga je da se do prepreke može doći jedino ako se robot pomeri, ne ako se rešenje pomeri, stoga bi trebalo pustiti robota da istražuje svoju okolinu. Ako se pomera prepreka sa, onda se bira koja će prepreka biti pomerena na (nasumično odabrani) susedni čvor. Ako se pomera robot, samo se pomeri na nasumični odabrani susedni čvor. Postupak se ponavalja sve dok se ne dodje do rešenja. Algoritam se ponavlja konačan broj iteracija, pri čemu pamti broj koraka neophodan da se do rešenja dodje, kao i u kojoj iteraciji se došlo do tog rešenja. Kako bi se ostavilo prostora da se potencijalno nadje što bolje rešenje, odlučeno je da se algoritam ponavlja veći broj puta.

Pored toga, ubačena je i jedna optimizacija ovom algoritmu. Kako bismo prikazali rad nekog naivnog algoritma, koji je više promišljen od običnog iscrpnog traženja rešenja dok se do rešenja ne dodje. Glavni problem prethodnog algoritma je bio u tome što su se nasumično pomerale prepreke čak i ako je postojalo neko rešenje u datom trenutku. Tim pomeranjem je postojeći put možda zatvoren, stoga se za potrebe razrešavanja te problematike koristi BFS algoritam za obilazak grafa, kako bi se u svakom trenutku znalo da li uopšte postoji direktno rešenje. Ako se desi da rešenje ne postoji onda će se lako odabrati neka od prepreka da se pomera. Napomenućemo još i da postojanje direktnog rešenja ne utiče na to da se ne pokušava sa pomeranjeme neke prepreke kako bi se došlo do boljeg rešenja. Algoritam radi tako što u svakoj iteraciji poziva BFS kako bi pokušao da nadje put. Ako uspe da nadje put, on je najkraći mogući za dati raspored prepreka, stoga se prekida trenutna iteracija, čuva se rezultat i vraćaju se prepreke na inicijalne položaje odakle se u narednoj iteraciji ponovo kreće ispočetka. Ukoliko BFS ne uspe da nadje put, nasumično se bira broj prepreka (bar 1) koje će biti pomerene već pomenutom *moveobstacle* funkcijom.

Prethodno opisani naivni algoritmi nemaju nikakav vid napretka (iako drugi lakše vidi puteve), u smislu da prethodne iteracije nikako ne utiču na naredne. Naime, ako dodje do pomeranja prepreka ne postoji nikakav vid povratne informacije da li se time nešto uradilo, u smislu da li se došlo do boljeg ili lošijeg rešenja, stoga će naredni algoritmi probati da se pozabave tom problematikom.

3.2 Optimizacija kolonijom mrava

Prva pokušana metoda prevazilaženja prethodno pomenutih nedostaka je korišćenjem optimizacije kolonijom mrava. Algoritam nije korišćen u svom osnovnom obliku, već je prilagodjen dodavanjem dodatnih optimizacija kako bi ga unapredile. Prilagodjavanje datog algoritma našim potrebama je mahom bilo individualne prirode, pod invidiualnim misli se na pojedine instance unutar kolonije, čisto da ne bude nekih zabuna. Odnosno, implementirali smo kako bi trebalo pojedinačni mravi da se ponašaju u raznim situacijama, kada ostavarivanje cilja nije moguće usled postojećih prepreka.

Algoritam se odvija unutar petlje što se podudara sa samim radom algoritma. Petlja će se odvijati sve dok trenutna jedinka (mrav) ima kuda da ode (sve dok ima dostupne čvorove u svojoj okilini), a da pritom već nije posećeno (kako se ne bi beskonačno vrteo u krug). Na kraju svake iteracije resetujemo prepreke na grafu, kako bismo dali šansu ostalima da se oprobaju sa već postojećim feromonima. Time dobijamo i na diverzifikaciji rešenja.

Ukoliko se dodje do cilja, prekida se rad tog pojedinca i na dosadašnji broj koraka se dodaje i broj prepreka koje je u medjuvremenu morao da pomeri kako bi uspešno došao do cilja. Nakon toga sledi jedna novina, koju smo implementirali. Dodata je funkcionalnost brisanja "slepih ulica". Do ovog problema je nemoguće doći u koordinatnom sistemu, ali pošto je razmatran malo širi skup problema valjalo bi otkloniti ga. Postoji mogućnost dolaženja do takozvane slepe ulice bar jednom, gde se kasnije naglašava ostalim mravima da je tamo nemoguće ići. Kad se dodje do slepe ulice, iz grafa se "briše" čvor u kom je mrav završio svoju karijeru (sportski rečeno). Ovim ne brišemo prepreke, već samo takozvane slepe ulice koje u 100% slučajeva rezultiraju da mrav ne uradi ništa. Kako ne bismo cele kolonije zaglavljivali tamo, neophodno je da se takvih čvorova rešimo, što i jesmo. Konkretno u kodu, pozivom se funkcije *sawnodes* otklanjamo ovakve nepremostive probleme.

Ukoliko mrav nema više susednih čvorova na koje može otići, ispisuje poruku o svojoj smrti, i započinje dalji rad svog posla. Ukoliko je umro u okolini prepreka, bira se nasumična medju njima, poziva se rekurzivna funkcija *processobstacle*. Data funkcija gleda da li ima smisla pomerati datu prepreku. To radi tako što proverava da li ima račvanja ili da li je pored cilja, ako je bilo šta od ovoga ispunjeno, nećemo je pomerati, kako time ne bismo potencijalno večno zaglavljivali put, stoga ne pomeramo prepreku. Ukoliko do toga ne dodje, prepreka se pomera.

Prepreka se pomera pozivanjem rekurzivne funkcije *moveobstacle*. Ona pomera prepreku na nasumičnu poziciju pored one na kojoj se nalazi prepreka (ukoliko postoji više mogućnosti za pomeranjem). U slučaju da se i na toj poziciji nalazi prepreka, ponavlja se rekurzija koja radi isto. Ako se ne nalazi prepreka i ako je slobodan čvor na toj poziciji, onda se prepreka pomera na tu poziciju. Kako ne bismo stalno pomerali prepreku za jedno mesto napred, zatim nazad na istu i tako u nedogled i time stajali u mestu što se iteracija pojedinaca tiče. Uvodimo stepen valjanosti mesta za postavljanje prepreke. U kodu implementirano kroz listu (*movecoefficients*) koja za svaki čvor grafa sadrži informaciju o njegovoj "validnosti". Ukoliko je prepreka već bila na datom čvoru, pri pomeranju sa tog mesta se toj poziciji smanjuje vrednost, kako bi se naglasilo da bi valjalo izbeći tu prepreku sledećeg puta.

Ako se desi da je mrav došao do kraja svog puta, a nije pored sebe imao prepreku, sa odredjenom verovatnoćom će pomeriti nasumičnu prepreku u grafu.

Na putanji na kojoj mrav umire često se takodje dodaje stepen kvaliteta tih čvorova kako bi se narednim mravima naglasilo da tu nije poželjno ići.

Nakon toga ostaje samo pustiti mrave da "sami" traže puteve i da pri tome ažuriraju količinu feromona posle svake iteracije. Smanjujemo količinu feromona koji su izraženi na datoj grani svakom iteracijom (simuliranja prirodnog ispravanja vremenom). Povećavamo količinu feremona posle svakog prolaska mrava tim putem, odnosno tom granom.

Algoritam na kraju ispisuje najbolje rešenje do kog se došlo, takodje ispisuje i neophodan broj koraka (broj pomeranja prepreka i cena puta) kako bi se stiglo do rešenja.

Bilo je ideja uvodjenja nekog vida heuristike kako mravi ne bi u potpunosti lutali, ali se nekako činilo da bi se time poremetio celokupan koncept optimizacije mravima i optimizacije uopšte, stoga se od te ideje odustalo.

3.3 VNS

VNS algoritam je jedan od već poznatih optimizacija, spada pod takozvane S-metaheuristike. Ideja koja krasi ovaj algoritam je da rešenje "napreduje" u smislu da će se skoro uvek za naredno rešenje uzeti ono čija je vrednost bolja. Naglašeno je skoro uvek pošto postoji problem "lokalnog optimuma" koji gledamo da izbegnemo što više. Naime, princip rada algoritma je da prihvata promene samo ukoliko pogoduju boljem rešenju, odnosno ako je rešenje lokalno bolje od trenutnog, takozvana intenzifikacija rešenja (eksploatišemo najbolje rešenje u datoj okolini). To se ne dešava u 100% slučajeva, naime postoji mala verovatnoća da će se uzeti rešenje koje nije nužno bolje (neće se odbaciti prethondno nadjeno najbolje rešenje). Taj postupak se sprovodi kako bismo pokušali da pokrijemo što veće polje pretrage, takozvana diverzifikacija (radimo eksploraciju okoline). Iako je rešenje lošije, prihvatamo ga sa nadom da će nas možda odvesti ka globalno još boljem (što ne mora uvek biti slučaj, ali pokušamo). Sam VNS je smenjivanje diverzifikacije i intenzifikacije kako bismo došli do što boljeg rešenja.

Pošto VNS u svom izvornom obliku treba da čuva stanje najboljeg rešenja i njegovu vrednost ovde nailazimo na mali problem. Naime, opis našeg rešenja nalaže da bi trebalo naći najkraći put, pri čemu se i broj pomeraja prepreke takodje računa kao pomeraj. Stoga, odlučili smo da kodiramo rešenje kao raspored prepreka, a vrednost rešenja tražimo korišćenjem BFS algoritma (pošto garantuje da će nadjeni put biti najkraći, obzirom da su sve grane iste težine) na čiju vrednost dodajemo ukupan broj pomeranja prepreka koje su prethodile izvršavanju BFS pretrage. Time korektno računamo vrednost rešenja shodno zahtevima problema. Ako se desi da BFS ne može naći rešenje dodeljuje mu se maksimalna moguća vrednost (najveći mogući broj u pokretnom zarezu u

Python-u) kako bi se to rešenje skroz isključilo iz razmatranja. Takvo rešenje je praktnično neupotrebljivo, stoga je i nedozvoljeno, time smo došli do ograničenja u vidu rešenja, konkretno za ovu instancu VNS problema.

Graf je predstavljen kao niz boolean vrednosti, na mestima na kojima se nalazi prepreka vrednost je *True*, inace je *False*. Prepreke se pomeraju već pomenutom *moveObstacle* funkcijom. Intenzifikacija je odvija funkcijom *localSearch-FirstImprovement*. Funkcija sprovodi algoritam lokalne pretrage, ali samo do prvog poboljsanja trenutno posmatranog rasporeda prepreka.

Nasumično bira prepreku koju će pomeriti na dozvoljeno mesto i shodno rezultatu tog pomeraja će prekinuti sa radom (trenutnog rasporeda pomeranja prepreka) ili nastaviti na narednu prepreku. Iteracije će se ponavljati sve dok postoji poboljšanje vrednosti rešenja (cilj je naći što manju vrednost). Ostatak VNS je standardno implementiran. Postupak se ponavlja u konačno mnogo iteracija, može se prihvatiti lošije rešenje sa odredjenom verovatnoćom ili nastaviti sa već postojećim rešenjem. Prepreke se postavljaju na početne položaje na početku svake iteracije (kako bi se probalo drugačije premeštanje prepreka funkcijom *shaking*, koja premešta k prepreka).

4 Eksperimenti

Nakon upoznavanja sa algoritmima, ovde ćemo se malo više pozabaviti samim kvalitetom dobijenih rešenja. Analiziraće se dobijeni rezultati i biće uporedjivani medjusobno. Testiraće se više slučajeva grafova, kao i kako utiče promena iteracija na dobijene rezultate. Već poznati test primeri nisu poznati, stoga ćemo generisati svoje grafove različitih veličina i atributa i nad istim proveravati kako algoritam radi. Posebno će se razmatrati dvodimenzina koordinatna mreža (takozvani grid) i obični neusmereni grafovi.

Nakon odradjenih eksperimenata nad narednim grafovima,

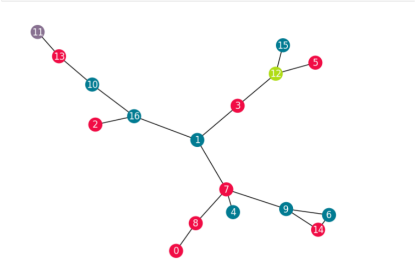

Figure 1: Graf veličine 15 čvorova

Nad ovim grafom, svi algoritmi su došli do rešenja bez većih problema. Ovakav rezultat je donekle i bio očekivan, obzirom da je graf veličine samo 15 čvorova. Nad ovakvim grafovima nije postojalo pravilo koji je najbolji, dešavalo bi se da se nekom od prostijih algoritama "posreći" i da dodju do istog rešenja kao i optimizacije, u zanemarljivo lošijem vremenu. Broj iteracija je postavljen da bude 100, pokazalo se kao i više nego dovoljno.

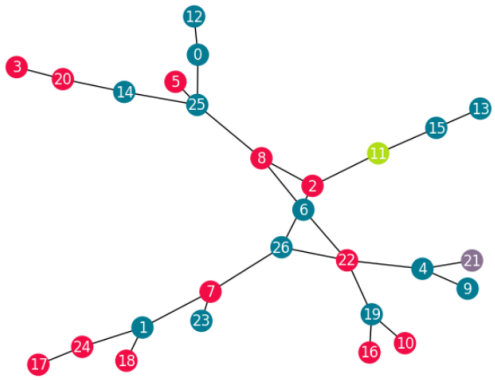

Figure 2: Graf veličine 25 čvorova

Nad ovim grafom, prostiji algoritmi su se mučili kako bi došli do upotrebljivog rešenja, čak i ako bi uspevali da dodju do bilo kakvog vida rešenja, trebalo bi značajno više vremena. Nad ovakvim grafovima već postaje evidentno koji algoritmi su bolji za primenu. VNS pomalo zaostaje za performansama koje ima algoritam optimizacije mravima, ali je još uvek upotrebljiv. Ovakvi rezultati su dobijeni uprkos povećanju broja iteracija na 1000, pokušalo se i sa malo većim brojem, ali se performanse slabijih algoritama svakako ne poboljšavaju značajno.

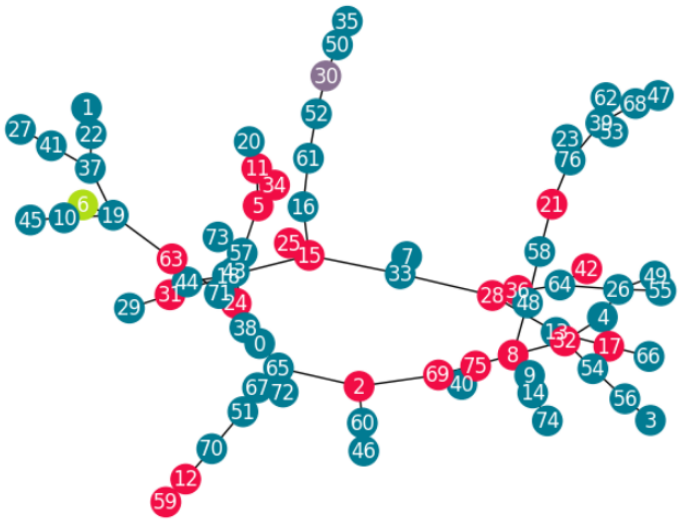

Figure 3: Graf veličine 75 čvorova

Nad ovim (i njemu sličnim grafovima) veličine 75 čvorova, pokušano je traženje rešenja sa čak 10000 iteracija (prilagodjeno upotrebi kolonije mrava). Ovakav graf je odličan prikaz razlike izmedju algoritama iscprne pretrage i naprednijih algoritama optimizacije. Naime, algoritmi isrcpne prepreke uglavnom nisu uspevali da nadju bilo kakav vid upotrebljivog rešenja (u unapred zacrtanom vremenskom periodu), ili se do rešenja ne bi došlo uopšte, ili bi se našlo rešenje koje bi zahtevalo preko 500 koraka (što je za graf veličine 75 čvorova sumanuto). Algoritam optimizacije kolonijom mrava se izuzetno dobro pokazao, daje skoro optimalno rešenje skoro uvek. Isprobavane su razne kombinacije broja mrava i broja iteracija, ali se čini da sa 100 mrava na 100 iteracija radi najbolje. VNS radi osetno dugo (preko 10 sekundi), naspram prethodnog algoritma kome treba relativno malo vremena (manje od jedne sekunde), ali time ne proizvodi bolje rezultate čime bi mogao opravdati svoju sporost. Neretko se dešavalao da VNS ne dodje do bilo kakvog rešenja, stim što je jedina razlika u odnosu na proste algoritme, ta što zapravo završi sa svim iteracijama. Pokušano je štelovanje parametara pokretanja VNS algoritma (menjanje verovatnoće uzimanja lošijeg rešenja, menjanje broja nasumično odabranih pozicija itd.), time se algoritam malo više prilagodjavao trenutnom izgledu grafa, ali u globalu radio malo bolje.

Kao što je bilo i naglašeno, pored običnih grafova, testirano je i kako radi nad takozvanim koordinatnim mrežama. Dobijeni su naredni rezultati:

Prvo se pokušalo sa mrežom veličine 10*10 (ukupno 100 čvorova). Radilo se samo sa 100 iteracija, uglavnom su svi algoritmi prolazili testiranje u zavidnom vremenu. Naravno, uvidja se nadmoć optimizacionih algoritama, ali nije dominantno izražena za ove primere. Iako su grafovi veći po broju čvorova nego prethodni primeri, sam zadatak je lakši zbog povećane povezanosti grafova kao i zbog samog izgleda (oblik mreže).

Kasnije se pristupilo povećanju veličine mreže na 15*15. Ovde već počinje da se uvidja razlika izmedju prostijih i optimizacionih algoritama. Prostiji algoritmi bi ponekad imali sreće i našli relativno optimlano rešenje, ali bi im svakog puta trebalo značajno više vremena nego optimizacionim algoritmima, kao što je i očekivano. Broj iteracije je povećan na 1000, što još uvek nije preterano mnogo, ali se razlika u vremenu izvršavanja itekako primećuje. Na slici ispod se mogu videti performanse datih grafika, kao i kako je tekao tok nalaženja rešenja. Iako za VNS treba znatno više vremena nego za algoritam kolonije mrava, što se tiče kvaliteta dobijenog rešenja, uopšte ne gubi korak. Neretko bi se desilo da rešenje pronadjeno VNS algoritmom bude bolje od onog dobijenog algoritmom kolonije mrava, što se ranije nije dešavalo.

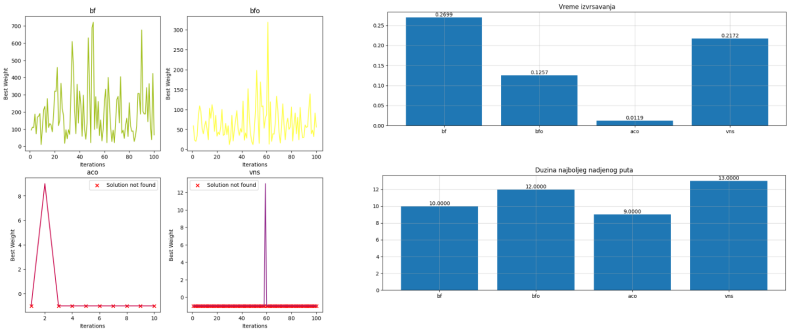

Figure 4: Performanse algoritma

Poslednji eksperiment koji je odradjen bio je nad grafom (mrežom) veličine 25*25. Sama preglednost mreže postaje otežana i izuetno neuredna, stoga se mahom fokusiramo na dobijene rezultate performansi. Prostiji algoritmi postaju praktično neupotrebljivi, treba im previše vremena (postoji gornje ograničenje kako ne bi radili beskonačno dugo) da bi došli do bilo kakvog vida rešenja, i ono obično nije ni približno optimalnom (može im se "posrećiti", ali to ih

nimalo ne čini pouzdanim). Algoritam optimizacije kolonijom mrava radi već dosad naviknuto (i očekivano) najbrže, i skoro optimalno. Jedina novina koja je primećena je u radu VNS algoritma koji radi uobičajeno sporije, ali daje uglavnom najbolje rezultate. Možda neočekivan rezultat, ali nakon kratkog promišljanja i razmatranja, rezultat koji ima smisla. Naime, VNS zbog svoje prirode itekako profitira kad je reč o algoritmima koji imaju oblik koordinatne mreže. Imaju dosta opcija za unapredjenjem u svakom koraku, i dosta smislenih pokušaja diverzifikacije, što im daje prednost nad svim ostalim (kada je reč samo o kvalitetu rešenja), ali brzina može kod većih primera možda predstavljati još uvek nepremostivu prepreku. Dosad opisani rezultati su grafički prikazini u slici ispod.

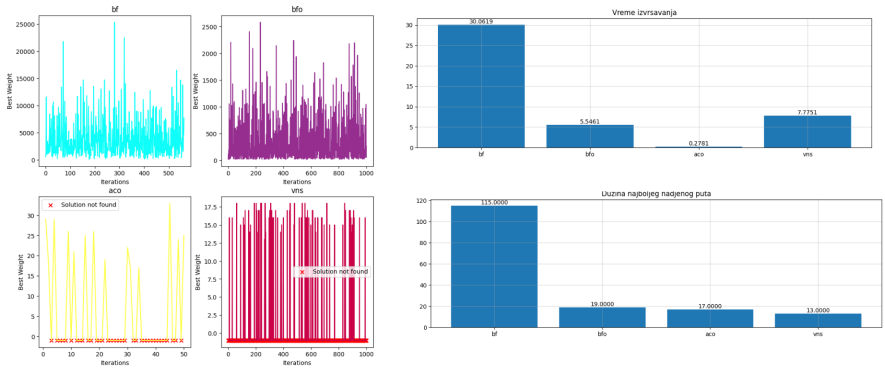

Figure 5: Performanse algoritma

5 Dosadašnji rezultati

Stručna literatura nije bila korišćena prilikom izrade rešenja, ali ćemo se ovde svakako osvrnuti na nju. Rešenja su većinski fokusirana na inžinjerske probleme koje treba primenjavati prilikom dizajniranja robota. Ovaj problem je zapravo konkretizacija grupe problema gde se radi sa k robota, dok je ovde k konkretno jednako jedan. Svi pokušaji rešavanja problema su uglavnom teorijske prirode primenom modifikacije već pomenutog A* algoritma, pod nazivom D* (i njegova modifikacija D*lite). Sam D* lite se primenjuje slično kao A*, stim što ima mogućnost da barata sa takozvanim dinamičkim, odnosno promenljivim prostorima, koji se javlja u ovom problemu. Postoje i pokušaji primene različitih tehnika poput gradijetnog spusta (minimizovanjem ciljne funkcije), korišćenje grafovske neuronske mreže, kao i razni pokušaji aproksimacije rešenja do na epsilon grešku broja koraka.

Pored ovih algoritama pokušaji rešavanja su mahom u realnim situacijama u kojima se robot može naći. Rešenja se mahom svode na nalaženje adekvatne heuristike i priemenjivanjem iste prilikom usmeravanja robota ka rešenju. Ali bez postojanja nekih konkretnih instanci koje su pogodne za testiranje postojećih rešenja. Zanimljivo je bilo uopštavanje problema gde se ne nalazi jedna robot, već proizvoljno mnogo, gde ne smeju u istom trenutu da se na istom grafu nalaze ni koja dva različita. Takva definicija problema uvodi dodatne restrikcije na samo rešenje, kao i razne mogućnosti za rešavanjem nad jednim te istim prostorom.

Nakon malo dužeg istraživanja pronadjeno je rešenje koje je takodje koristilo optimizaciju kolonijom mrava, razlika je bila što se problem razmatrao nad značajno kompleksnijim prostorom (nisu sve grane iste težine, nisu svi putevi podjednako bezbedni, putanje nisu nužno linearne, itd.), stoga je upotrebljen Dijkstrin algoritam, koji smo i mi razmatrali ali smatrali suvišnim za naš problem. Naime, rešenje se fokusira na nalaženje nekih konstantnih stanja grafova za koje se lemama pokazuje da zapravo imaju rešenje, koje je do odredjene granice lošije od postojećeg. Samo eksperimentisanje je izvedeno na grafovima veličine manje od 20 čvorova, tako da predmet posmatranja eksperimenta nije naročito primamljiv da bismo pokušali da ga kopiramo.

6 Zaključak

Kao što je i u radu prikazano, problem nalaženja puta do cilja uz postojanje prepreka će u mnogome zavisiti od samog rasporeda prepreka, i to je čak ako bismo uzeli u obzir da postoji rešenje datog grafa sa postavljenim preprekama. Osim toga bi možda bilo poželjno proveriti više puta neke neuspele pokušaje jer je možda rano zatvoren put nasumičnim pomeranjem.

Naivni algoritmi su praktično neupotrebljivi nad ovakvim problemima čim se imalo zadje na veličine grafova koje će biti upotrebljavane u stvarnom svetu. Problem, koji se primećuje već i na grafovima koji imaju samo nekoliko desetina čvorova. Korišćenjem optimizacije kolonijom mrava smo dospeli do zadovoljavajućih rezultata, ali ne treba ni VNS odbacivati kao neuoptrebljiv, iako sporiji, pokazuje bolje rezultate u odredjenim domenima problema (konkretno koordinatne mreže).

Mišljenja smo da smo samo zagrebali površinu kompleksnosti rešenja ovog problema. I to ako bismo ostali na konkretnoj istanci problema, bez dodavanja novih pravila za broj robota, vrste težina grafova itd. Postoji bezbroj mogućih situacija u kojima se mogu uzeti u obzir razni slučajevi izgleda grafova, pomeranja prepreka kako bi se u što manje koraka stiglo do rešenja, itd. Mi smo ovde pokušali da što opštije obuhvatimo neke instance problema koje se pojavljuju. Ukoliko bismo bili ograničeni na pojedine tipove problema, verujemo da bismo

prilagodjavanjem samih algoritama optimizacije dobili značajno na kvalitetu rešenja, pa čak i na brzini izvršavanja.

Kada se sve uzme u obzir optimizacioni algoritmi rade zadovoljavajuće dobro. Postoje mogućnosti za unapredjenjem, koje će se sigurno ostvariti u bliskoj budućnosti, ali se dosad prikazano može smatrati za dovoljno uradjenim kako bi se premostile osnovne prepreke i nedostaci.