Diskretni Time-Cost Trade-Off problem

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Ana Marković
mi16127@alas.math.rs

Septembar 2022

Sažetak

U ovom radu će biti obrađena diskretna verzija Time-Cost Trade-Off problema, konkretno budžet problema. Razmotrićemo algoritam grube sile, kao i optimizacione algoritme prilagođene problemu i njihovo ponašanje.

Sadržaj

1 Uvod

Dat je projekat P i konačan skup aktivnosti J nad kojim je zadat parcijalni poredak (J, <). Da bi se projekat završio aktivnosti moraju da budu izvršene u skladu sa ograničenjima zadatim parcijalnim poretkom: ako je j < k, aktivnost k ne sme da počne pre nego što se aktivnost j završila. Aktivnosti su nedeljivi zadaci. Trajanje aktivnosti $j \in J$, odnosno razlika između vremena početka i završetka, zavisi od njene cene. Korelacija između vremena i cene opisana je nerastućom nenegativnom funkcijom cene $c_j : \mathbb{R}_+ \to \mathbb{R}_+ \cup \infty$ za svaku aktivnost $j \in J$, gde je $c_j(x_j)$ količina novca koja mora da se plati da bi se pokrenuo zadatak j sa trajanjem x_j.

U ovom radu ćemo projekat P predstaviti usmerenim acikličnim grafom. Aktivnost $j \in J$ se predstavlja čvorom u grafu, tako da za svake dve aktivnosti $j, k \in J$ postoji putanja od j do k samo ako j < k u zadatom parcijanom poretku. Dodajemo i početni i krajnji čvor koji će označiti početak i kraj projekta, ovi čvorovi nisu stvarne aktivnosti projekta pa ih nazivamo i *dummy* čvorovi, njihvo vreme izvršavanje je 0 kao i cena izvršavanja. Iz početnog čvora, kog nazivamo *source*, povlačimo grane u sve čvorove čiji je ulazni stepen 0, odnosno povlačimo granu ka zadacima pre kojih ne mora da se izvrši ni jedan drugi zadatak. Ka krajnjem čvoru povlačimo grane iz onih čvorova koji imaju izlazni stepen 0, odnosno povlačimo granu iz zadataka od kojih ne zavisi izvršenje ni jednog drugog zadatka ka čvoru koji označava završetak projekta.

Za dati projekat P označimo njegovu realizaciju sa x, što znači da je projekat uspešno počeo i da su se izvršili svi zadaci u okviru projekta. Ukupna cena realizacije projekta x je:

$$c(x) := \sum_{j \in J} c_j(x_j)$$

Pored cene realizacije za dati projekat možemo da izrazimo i dužinu trajanja projekta. Dužina trajanja se može predstaviti kao najduži put, posmatrajući vremena izvršavanja zadataka, između početnog i krajnjeg čvora. Drugim rečima, naredni zadatak ne može početi pre nego sto se prethodni nije završio, prateći zadato uređenje, tako da ako posmatramo najranije vreme početka (*eng. eariest start time*) ono je ustvari jednako vremenu najkasnijeg završetka zadatka koji mu prethode.

Ono što možemo da primetimo je da imamo dva parametra po kojima možemo da optimizujemo zadati projekat P a to su cena i vreme. Naravno idealno bi bilo optimizovati oba, ali u tom slučaju mora doći na neki način do biranja prioriteta da li nam je bitnije optimizovati cenu ili vreme, jer prema datoj postavci kraće vreme izvršavanja dovodi do skupljih projekata, i obrnuto duže izvršavanje košta manje. Ovaj problem najčešće se dalje deli na problem budžeta (*eng.* **budget problem**) i problem roka (*eng.* **deadline problem**). Na dalje ćemo se baviti budžet problemom. [2]

Ono što dodatno treba napomenuti da ovaj problem ima linearnu i diskretnu varijantu, linearna varijanta ima poznato rešenje u polinomijalnom vremenu, dok je diskretna varijanta NP težak problem, tako da ćemo se na dalje baviti diskretnom varijantom. U *diskretnom projektu* svaka aktivnost ima konačan broj kombinacija dužina izvršavanja i cena za datu dužinu izvršavanja, umesto da je cena data kao linearna funkcija zavisna od vremena izvršavanja kao što je slučaj u linarnoj varijanti. Tako da je cena izvršavanja u zavisnosti od vremena izvršavanja zadatka $j \in J$ zadata kao

stepenasta funkcija:

$$c_j(t) = \left\{ \begin{array}{ll} \infty, & 0 \leq t \leq a_{j,1} \\ c_j(a_{j,i}), & a_{j,i} \leq t \leq a_{j,i+1} \\ & 1 \leq i < l_j \\ 0, & a_{j,l_j} \leq t \end{array} \right\}$$

gde je l_j broj datih kombinacija vreme-cena izvršavanja. Ostalo je još eksplicitno da zadamo problem budžeta:

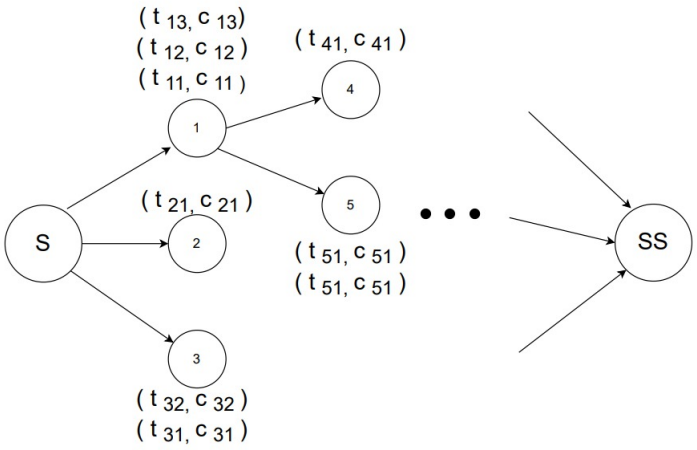
Za datu nenegativnu vrednost budžeta $B \geq 0$ naći realizaciju x projekta P koja zadovoljava:

$$c(x) \leq B \wedge T_{opt}(B) := min\{t(x) | x \in \mathbb{R}_+^j, c(x) \leq B\}.$$

2 Algoritam grube sile

Algoritam grube sile ispituje sve moguće kombinacije vremena izvršavanja zadataka svih čvorova, a zatim od svih tih kombinacija bira onu sa najmanjim vremenom realizacije projekta koja zadovoljava budžet. Ovo je egzaktni algoritam i sigurno dobijamo optimalno rešenje, ali jasno je da je složenost izvršavanja eksponencijalna. Koliko brzo broj kombinacija raste možemo zaključiti na malom primeru sa slike 1.

```
function BruteForce(graf, cene, budzet):
 minimalnaRealizacija = inf
 for(sve moguce kombinacije vremena izvrsavanja):
  if duzina(kombinacija) < minimalnaRealizacija:
   if cena(kombinacija) < budzet:
    minimalnaRealizacija = duzina(kombinacija)
 return minimalnaRealizacija
```



Slika 1: Primer jednog grafa sa pridruženim cenama

3 S metaheuristike

Ova grupa algoritama pretražuje prostor rešenja počevši od jednog nasumično izabranog rešenja koje pripada domenu, zatim diverzifikacijom i intenzifikacijom tog rešenja u konačnom broju koraka pokušavaju da nađu optimalno rešenje problema.

U svim algoritmima koji slede značajno mesto zauzima mala izmena rešenja radi širenja prosora pretrage. Rešenje problema u slučaju problema bužeta predstavljeno je kao niz celobrojnih vrednosti gde vrednost na poziciji i u nizu predstavlja koji je od parova (cena, vrednost) uzeti u razmatranje za čvor i. Izmena rešenja je vršena tako što je za proizvoljni čvor i uzet jedan od preostalih parova (cena, vrednost). Ukoliko izabrani čvor i ima samo jedan par biramo novu pozicijju, odnosno novi čvor, ovim se osiguravamo da u promeni rešenja zaista dođemo do promene.

3.1 Lokalna pretraga

Ideja ovog algoritma je jednostavna, biramo jedno rešenje iz skupa dopustivih rešenja, dok se ne ispuni uslov zaustavljanje vršićemo promenu rešenja na prethodno opisan način i proveru da li je to novo rešenje bolje od trenutnog, ako jeste to rešenje postaje novo rešenje koje intenzifikujemo. Rezultat rada je minimalno vreme izvršavanja projekta od svih realizacija projekta koje smo obradili.

```
function LocalSearch(graf, cene, budzet):
 trenutnoResenje = inicijalizacija(graf,cene,budzet)
 minResenje = trenutno

 while not kriterijumZaustavljanja:
  novoResenje = izmeniResenje(trenutnoResenje)
  if prekoracujeBudzet(novoResenje, budzet, cene):
   nastavi
  else:
   if trajanje(novoResenje) < trajanje(trenutno):
    trenutnoResenje = novoResenje
    if trajanje(novoResenje) < trajanje(minResenje):
     minResenje = novoResenje
 return trajanje(minResenje)
```

3.2 Simulirano kaljenje

Simulirano kaljenje je optimizacioni algoritam inspirisan procesom termičke obrade metala. Prilikom kaljenja što duže zagrejani metal hladimo u vodi ili ulju to njegova struktura postaje stabilnija i čvršća. Preneseno u svet optimizacionih algoritama ideja je sledeća: biramo jedno rešenje iz skupa dopustivih rešenja, zatim uzimamo rešenje iz okoline na opisan način, ukoliko je rešenje bolje od trenutnog njega uzimamo za trenutno ako nije primenjujemo koncept kaljenja. To znači da iako je rešenje lošije mi želimo da ga uzmemo u razmatranje, ali pod određenim uslovima. Što je vreme izvršavanja algoritma do tog trenutka kraće to je veća verovatnoća da ćemo uzeti to lošije rešenje i na taj način proširimo prostor pretrage, što je vreme izvršavanja algoritma duže to je verovatnoća da ćemo uzeti

lošije rešenje manja jer rešenje postaje „čvršće i stabilnije" i vršimo intenzifikaciju trenutnog rešenja. Algoritam se izvršava dok nije ispunjen neki uslov zaustavljanja, što je u našem slučaju zadati broj iteracija.

```
function SimulatedAnnealing(graf, cene, budzet):
 trenutnoResenje = inicijalizacija(graf,cene,budzet)
 minResenje = trenutno

 while not kriterijumZaustavljanja:
  novoResenje = izmeniResenje(trenutnoResenje)
  if prekoracujeBudzet(novoResenje, budzet, cene):
   nastavi
  else:
   if trajanje(novoResenje) < trajanje(trenutno):
    trenutnoResenje = novoResenje
    if trajanje(novoResenje) < trajanje(minResenje):
     minResenje = novoResenje
   else:
    p = 1 / sqrt(i)
    r = uniform(0,1)
    if p > r:
     trenutnoResenje = novoResenje
 return trajanje(minResenje)
```

3.3 Metoda promenljivih okolina

Do sada su i algoritam lokalne pretrage i simuliranog kaljenja u svakoj iteraciji menjali najviše vreme izvršavanja za jedan zadatak, ideja ovog algoritma je da u svakoj iteraciji ispita k suseda trenutnog rešenja ili dok ne naiđe na bolje rešenje u tih k poseta, gde će za j-tog suseda menjati vreme izvršavanja za j zadataka , ako naiđemo na to bolje rešenje postaje trenutno i ponavljamo postupak do ispunjenja uslova zaustavljanja.

```
function RVNS(graf, cene, budzet, num_k):
 trenutno = inicijalizacija(graf,cene,budzet)
 minResenje = trenutno

 while not kriterijumZaustavljanja:
  k = 0
  while k <= num_k:
   novoResenje = nadjiSuseda(trenutno, k)
   if prekoracujeBudzet(novoResenje, budzet, cene):
    nastavi
   if trajanje(novoResenje) < trajanje(trenutno):
     trenutno = novoResenje
     if trajanje(novoResenje) < trajanje(minResenje):
      minResenje = novoResenje
     break
    else:
     k = k + 1
 return trajanje(minResenje)
```

4 Genetski algoritam

Prethodna grupa algoritama koje smo obrađivali su *s metaheuristike*, njihova koncentracija je bila na optimizovanju jednog rešenja. Sledeći algoritam koji ćemo obraditi pripada grupi *p metaheuristika*, odnosno populacionim algoritmima. To znači da umesto da imamo jedno rešenje koje optimizujemo sada ćemo imati čitavu populaciju.

Genetski algoritam pripada još i grupi evolutivnih algoritama, koji su inspirisani evolucijom, tako da ćemo na dalje razmotriti predstavljanje problema budžeta u terminima evolutivnog izračunavanja.

Jedna *jedinka* populacije predstavlja jedno rešenje našeg problema, ona sadrži *hromozom* predstavlja reprezentaciju rešenja, kao i do sada, niz celobrojnih vrednosti koje označavaju koji je od parova (vreme, cena) uzet za dati zadatak. Za svaku jedniku računamo prilagođenost, što je u našem slučaju i funkcija cilja, a to je najmanje vreme potrebno da se izvrši ceo projekat. Što je vreme manje to je jedinka prilagođenija.

Na početku potrebno je generisati populaciju, tako da je broj jedinki zadat parametar algoritma. Jedinke generišemo tako da svaka jedinka ima hromozom iz dopustivog prostora rešenja. Kako ne želimo da se najbolje jedinke izgube izabran je elitistički pristup, pa se n najboljih jedinki iz trenutne populacije smešta u novu populaciju, gde je n parametar algoritma. Dalje je potrebno izabrati jedinke koje će učestvovati u ukrštanju i da bi se popunio ostatak nove populacije. Za svako od preostalih mesta u novoj populaciji biramo roditelje operatorom selekcije, zatim za dva dobijena roditelja vršimo ukrštanje i na dva dobijena deteta iz ukrštanja vršimo mutaciju. Jedinke koje su nastale ovim putem smeštamo u novu populaciju. Sve osim inicijalizacije algoritma se ponavlja onoliko puta koliko smo zadali kao parametar algoritma za veličinu generacije [1].

Mutacija je vršena tako što je za proizvoljan zadatak iz hromozoma izabrano neko drugo od ponuđenih vremena izvršavanja, u slučaju da je ovo dovelo do prekoračenja budžeta popravljamo rešenje. Ukrštanje je vršeno jednopoziciono, za prvo dete do slučajno generisane pozicije uzimamo deo hromozoma jednog roditelja, a od te pozicije uzimamo hromozom drugog roditelja i za drugo dete obrnuto, takođe je vršena popravka rešenja u slučaju prekoračenja budžeta. Selekcija roditelja je turnirska, gde na osnovu veličine turnira koja se zadaje kao parametar algoritma mi biramo toliko jedniki iz populacije i ona koja je najbolja, ima najbolji fitnes pobeđuje.

Ostalo je još opisati popravku rešenja, u slučaju da prekoračimo budžet možemo da povećamo vreme izvršavanja projekta i time smanjimo cenu. Umesto da povećavamo vreme izvršavanja proizvoljnim zadacima mi trazimo zadatke koje imaju najveću cenu i povećavamo baš njima vreme izvršavanja sa namerom da smanjimo cenu.

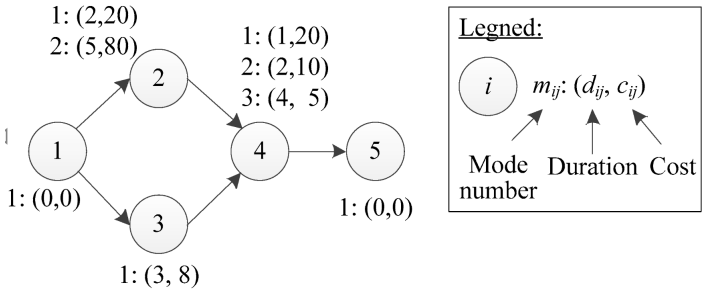
```
function GeneticAlgorithm(graf, cene, budzet):
 populacija = inicijalizacija(graf, cene, budzet)

 while not izvrsenoZaBrojGeneracija:
  novaPopulacija = najboljeJedinke(populacija, n)
  while not len(novaPopulacija) == velicinaPopulacija:
  roditelj1 = selekcija(populacija)
  roditelj2 = selekcija(populacija)
  dete1, dete2 = ukrstanje(roditelj1, roditelj2)
  mutacija(dete1)
  mutacija(dete2)
  novaPopulacija = novaPopulacij + dete1 + dete2
  populacija = novaPopulacija
 return najboljaJedinka(populacija)
```

5 Realizacija opisanih metoda i rezultati

Za potrebe rada bilo je teško pronaći već generisane grafove zavisnosti zadataka, ali kako je lako generisati usmerene aciklične grafove napisan je kod za to i grafovi su generisani kao i cene i dužine zadataka za svaki od čvorova grafa. Generisani su grafovi računih veličina gde su varirali broj datih zadataka i njihova zavisnost, odnosno broj čvorova i broj grana. Na slici 2 je dat graf iz literature zajedno sa načinom predstavljanja dužine trajanja zadataka i odgovarajuće cene za datu dužinu, ovaj graf je korišćen za testiranje ispravnosti algoritam [1].

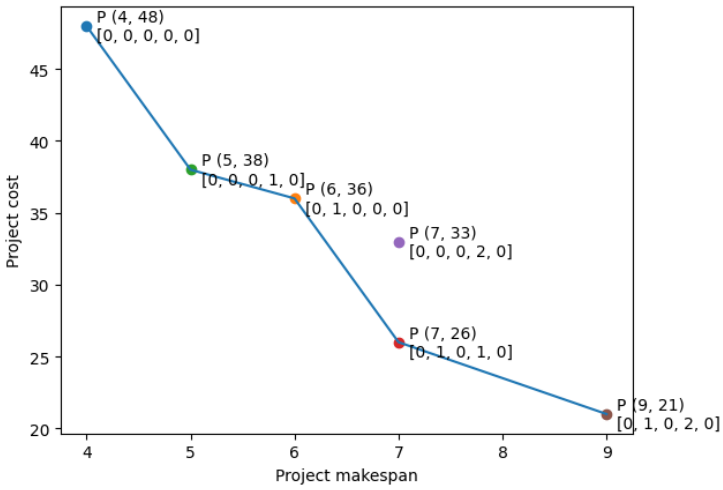
Svi optimizacioni algoritmi su imali znatno lošija rešenja nego algoritam grube sile, u nekim slučajevima i do 3 puta gore rešenje. Za male ulaze imamo dobra rešenja i sa optimizacionim algoritmima, ali naravno to nije cilj. Dalje je razmatrano ponašanje u odnosu na broj iteracija.



Slika 2: Prikaz jednog projekta

5.1 Algoritam grube sile

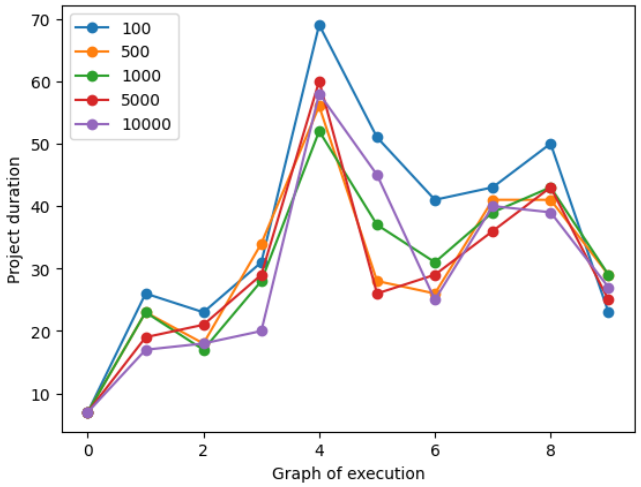
Algoritam grube sile je pored toga što je služio da dobijemo egzaktno rešenje pomogao i da dobijemo sva međurešenja za dati projekat na slici 2 i uporedimo sa vrednostima koje su u literaturi [1]. Vrednosti se poklapaju, a sa slike 3 možemo da posmatramo i Pareto-optimalan skup u slučaju razmatranja optimizovanja po oba kriterijuma, i cene i dužine trajanja projekta.



Slika 3: Prikaz dobijenih rešenja za sva moguća trajanja i sve moguće budžete projekta

5.2 Lokalna pretraga

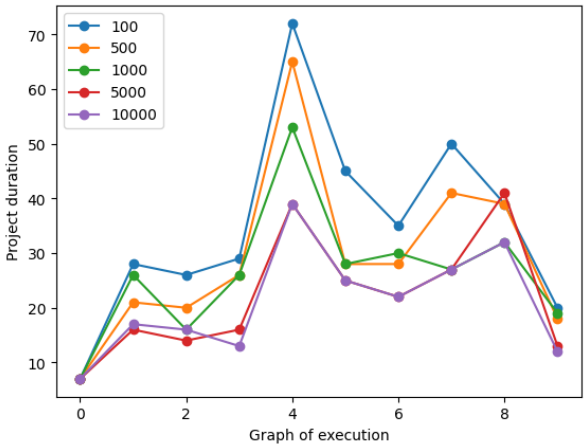
Lokalnom pretragom smo optimizovali jedno slučajno rešenje za različiti broj iteracija. Ponašanje možemo da vidimo na slici 4. Najveće odstupanja u vrednostima kroz iteracije vidimo da su za graf broj 5, što se i ne uklapa u očekivanja jer je to projekat srednje veličine, na primer graf broj 4 je sa znatno većim brojem zadataka i sa mnogo većom povezanošću između zadataka i vrednosti manje odstupaju.



Slika 4: Rezultat rada algoritma lokalne pretrage za različite grafove i različit broj iteracija

5.3 Simulirano kaljenje

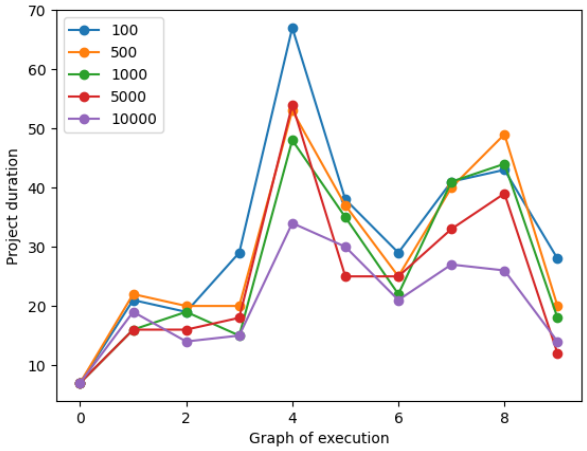
Kao i kod lokalne pretrage kao uslov zaustavljanja koristili smo broj iteracija. Vrednosti značajnije odstupaju iz iteracije u iteraciju u odnosu na lokalnu pretragu ali ono što možemo da primetimo je da je zavisnost između broja iteracija i vrednosti koje dobijamo veća i da možemo da se oslonimo na veći broj iteracija za bolje rešenje (slika 5).



Slika 5: Rezultat rada algoritma simulirano kaljenje za različite grafove i različit broj iteracija

5.4 RVNS

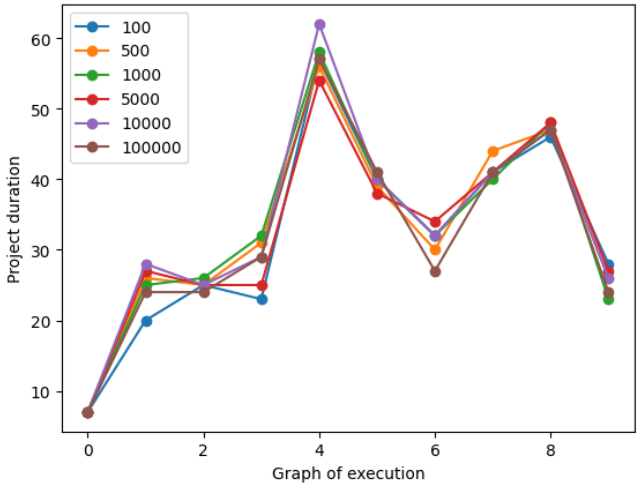
Situacija je slična kao i kod rada algoritma simulirano kaljenje, s tim što su vrednosti bliže iz iteracije u iteraciju u odnosu na rezultate simuliranog kaljenja (slika 6).



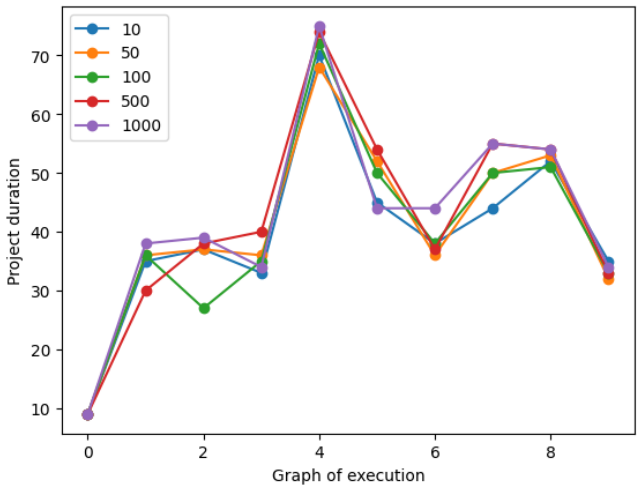
Slika 6: Rezultat rada algoritma RVNS za različite grafove i različit broj iteracija

5.5 Genetski algoritam

U ovom slučaju umesto iteracija predstavljen je grafik za broj generacija (slika 7). Ono što možemo da zapazimo da rešenja ne odstupaju mnogo, broj generacija nije značajno uticao na poboljšanje. U kombinaciji sa simuliranim kaljenjem (slika 8) rešenja nešto više odstupaju. Ostali parametri genetskog algoritma su bili fiksirani: veličina populacija, verovatnoća mutacije, veličina elitizam grupe.



Slika 7: Rezultat rada genetskog algoritma za različite grafove i različit broj generaciaj



Slika 8: Rezultat rada genetskog algoritma u kombinaciji sa simuliranim kaljenjem za različite grafove i različit broj generacija

6 Zaključak

Očekivano najbolje ponašanje imao je RVNS. Sa trenutnom reprezentacijom problema nije značajna promena okoline ukoliko promenimo vreme izvršavanja za samo jedan zadatak, sa promenom vremena izvršavanja za više zadataka dobili smo bolje ispitan prostor rešenja i optimalnije rezultate. Rešenja divergiraju kada uzmemo u obzir broj iteracija, ali možemo da se oslonimo na to da će veći broj iteracija dovesti do boljih rešenja, algoritam se relativno brzo izvršava, najduže u odnosu na ostale s metaheuristike ali svakako mnogo brže od algoritma grube sile.

Literatura

[1] Hongbo Li, Zhe Xu, and Wenchao Wei. Bi-objective scheduling optimization for discrete time/cost trade-off in projects. *Sustainability*, (8), 2018.

[2] Martin Skutella. Approximation algorithms for the discrete time-cost tradeoff problem. *Mathematics of Operations Research*, 23(4):909–929, 1998.