SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Projekt u okviru predmeta "Heurističke metode optimizacija"
2017/18

Problem raspoređivanja testova

Dokumentacija

Filip Gulan **Jan Kelemen**

Zagreb, siječanj 2018.

Sadržaj

1. Opis problema

Problem raspoređivanja testova je vrsta problema raspoređivanja čiji je cilj rasporediti unaprijed poznati skup zadataka nad unaprijed poznatim skupom resursa uz što kraće vremensko trajanje uz poštivanje određenih ograničenja.

Općenito ograničenja mogu biti čvrsta i meka. Čvrsta ograničenja su ona koja moraju biti zadovoljena da bi rješenje problema bilo valjano. Meka ograničenja su ona za koja je poželjno da budu zadovoljena. Prihvatljivost mekih ograničenja ne utječe na valjanost rješenja već na njegovu kvalitetu.

Kod problema raspoređivanja kvaliteta rješenja je određena tako da vrijeme trajanja izvođenja zadataka bude što kraće uz što veću količinu zadovoljenih mekih ograničenja [1].

1.1. Problem raspoređivanja testova

Cilj ovog projekta je rasporediti unaprijed poznat skup testova u što kraćem vremenskom trajanju. Pojedinačni test ima svoje određeno trajanje. Vremensko trajanje je definirano kao razlika između početka izvođenja najranijeg testa i vremena završetka testa koji zadnji završava.

Uz kriterij da vremensko trajanje bude što kraće, postoje još dva ograničenja. Pojedinačni test ima zadan niz strojeva na kojim se može izvoditi, te niz globalnih resursa koji su nužni za izvođenje testa. Oba navedena ograničenja su tvrda ograničenja.

2. Opis primjenjenog algoritma

Za probleme raspoređivanja može se primijeniti nekoliko načina rješavanja: slijedne metode, metode grupiranja, pristupi temeljeni na ograničenjima i meta-heurističke metode [1]. Na projektu je korištena meta-heuristička metoda, odnosno, eliminacijski (engl. *steady-state*) genetski algoritam. Kod genetskog algoritma se razlikuju dva pristupa: eliminacijski i generacijski.

Eliminacijski (engl. *steady-state*) genetski algoritam uzima određen broj jedinki iz populacije, uz dani kriterij, eliminira ih te ih nadokanđuje novima dobivenim operacijama križanja i mutacije.

Generacijski (engl. *generational*) genetski algoritam uzima određen broj jedinki iz populacije, uz dani kriterij, te od njih operacijama križanja i mutacije stvara potpuno novu populaciju.

Za genetski algoritam postoji nekoliko važnih karakteristika koji ga definiraju, a to su: reprezentacija rješenja, funkcija cilja i prikladnosti, stvaranje početne populacije, način odabira jedinki, kriterij zaustavljanja, operacije križanja i mutacije [1, 2].

2.1. Reprezentacija rješenja

Rješenje, odnosno fenotip, genetskog algoritma reprezentirano je kao skup parova $(\text{zadatak}, \text{stroj}, \text{početno vrijeme})$. Reprezentacija rješenja nad kojim se obavljaju genetički operatori, odnosno genotip, prikazan je skup parova $(\text{zadatak}, \text{stroj})$.

2.2. Funkcija cilja i prikladnosti

Funkcije cilja i funkcije prikladnosti dodjeljuju numeričku vrijednost pojedinom rješenju, koja govori koliko je rješenje kvalitetno [2]. Budući da u ovom problemu

nema mekih ograničenja, funkcija cilja i funkcija prikladnosti je definirana kao ukupno trajanje rješenja.

2.3. Stvaranje početne populacije

Početna populacija stvara se pohlepnim algoritmom. Za svaki pojedini test, u nasumičnom poretku, traži se najraniji trenutak kada on može započeti s izvršavanjem na nekom od strojeva na kojima mu je dozvoljeno izvršavanje. No osim strojeva, istovremeno se provjerava i dostupnost resursa za pojedini test, te se traži minimalno moguće vrijeme za koje bi test mogao biti obavljen na danom stroju uz zadane resurse. Prilikom pretraživanja mogućih vremenskih intervala za strojeve i resurse traži se najmanja/najveća (ovisno o korištenoj heuristici, objašnjeno u poglavlju 3) moguća "rupa" u rasporedu za resurse i strojeve. Također, iako u skolpu ovog projekta to nije bio zahtjev, omogućeno je dodavanje brojnosti globalnih resursa, to jest postojanje više instanci istog resursa što bi moglo skratiti samo vrijeme izvođenja.

Tako proveden algoritam automatski stvara samo valjana (engl. *feasible*) rješenja u vidu dostupnih resursa i strojeva što uvelike olakšava kasniju implementaciju operatora križanja i mutacije. Na ovaj način preslikavanje testova na strojeve jednoznačno definira pojedino rješenje te se prije navedeni operatori vrlo lako mogu primjeniti na tablici kojoj je test ključ, a stroj vrijednost.

Pseudokod 1 prikazuje opisani algoritam za stvaranje valjanog rješenja te se u njemu se koristi nekoliko pomoćnih funkcija. Funkcija IS_FREE koja pronalazi vrijeme kada je stroj slobodan, te funkcija ARE_FREE koja pronalazi vrijeme kada su svi potrebni resursi slobodni.

Pseudokod 1 Stvaranje početnog rješenja

```
1: function GENERATE_SOLUTION(definition)
2:     solution ← {}
3:     for each task ← definition.tasks do
4:         insertion ← (time : int.max, machine : null)
5:         for each mach ← task.machines do
6:             tp ← 0                                          ▷ timepoint
7:             while true do
8:                 machine_t ← IS_FREE(mach, tp)
9:                 resource_t ← ARE_FREE(mach.resources, tp)
10:                if machine_t = resource_t then
11:                    if machine_t ≤ insertion.time then
12:                        insertion ← (time, mach)
13:                    end if
14:                end if
15:            end while
16:        end for
17:        ADD(solution, (task, insertion))
18:    end for
19:    return solution
20: end function
```

2.4. Kriterij zaustavljanja

Kriterij zaustavljanja genetskog algoritma je karakteristika koja govori pod kojim uvjetima se genetski algoritam zaustavlja. On može biti na primjer maksimalan broj generacija ili trenutak kada se vrijednost funkcije prikladnosti prethodnog i trenutnog koraka razlikuje za manje od nekog praga [2]. Za potrebe ovog projekta korišteno je ograničenje maksimalnog broja generacija te makismalnog broja generacija bez promjene, uz dodatak vremenskog kriterija prilikom evaluacije zadanih instanci problema.

2.5. Korak eliminacijskog genetskog algoritma

Jedan korak genetskog algoritma sastoji se od odabira jedinki, operacija križanja nad odabranim jedinkama te operacija mutacije dobivenog djeteta, te zamjene jedinki iz populacije novo-dobivenim jedinkama.

2.5.1. Način odabira jedinki

Kao način odabira jedinki u ovom projektu odabrana je k-turnirska ($k > 2$) selekcija. K-turnirska selekcije je selekcija pri kojoj se iz populacije izvlači nasumičan uzorak od k rješenja [3]. Zatim se dva najbolja rješenja, od k odabarnih, križaju, provodi se operacija mutacije nad dobivenim djetetom, te se tako dobiveno dijete zamijeni s najlošijom jedinkom iz k odabranih, no naravno uz uvjet da je dobivena jedinka bolja od one koja se mijenja.

2.5.2. Operacije križanja i mutacije

Operacija križanja je operacija kojom se kombiniraju dva postojeća rješenja u novo rješenje. Implementirane su dvije verzije operacije križanja, nasumično križanje te križanje s n točaka prekida.

Nasumično križanje uzima dvije instance rješenja te za svaki test i uz određenu vjerojatnost uzima stroj pridružen testu jednoj od instanci rješenja.

Križanje s n točaka prekida uzima dvije instance rješenja te u n točaka prekida za pojedini genotip testova uzima strojeve iz jednog rješenja pa nakon toga iz drugog rješenja naizmjence, s tim da se točke prekida odabiru nasumično.

Operacija mutacije je operacija kojom se izmjenjuje jedno rješenje. Primjenom operacije mutacije moguće je poboljšati kvalitetu tog rješenja ili ga pomaknuti iz lokalnog optimuma. Također se primjenjuje uz određeni vjerojatnosni prag. Implementirana operacija mutacije mijenja stroj na kojemu se pojedini test izvršava s jednim od mogućih strojeva za dani test.

2.6. Pseudokod

Pseudokod 2 prikazuje primijenjeni algoritam. U njemu se koristi nekoliko pomoćnih funkcija. Funkcija K_BEST_SOLUTIONS koja vraća k najboljih rješenja iz

predane populacije. Funkcija SELECT_RANDOM koja vraća k nasumičnih rješenja iz predane populacije. Funkcija WORST_SOLUTION koja vraća najgore rješenje iz predane populacije. Funkcija FITNESS čije je ponašanje opisano u poglavlju 2.2. Funkcija SWAP koja zamjenjuje predane jedinke iz populacije.

Pseudokod 2 Eliminacijski genetski algoritam

```
1:  function GENETIC(definition, population, max_iterations)
2:      iteration ← 0
3:      best_solution ← K_BEST_SOLUTIONS(population, 1)
4:      while iteration ≠ max_iterations do
5:          iteration ← iteration + 1
6:          k_random ← SELECT_RANDOM(population, k)
7:          two_best ← K_BEST_SOLUTIONS(k_random, 2)
8:          child ← CROSSOVER(two_best)
9:          child ← MUTATE(child)
10:         worst ← WORST_SOLUTION(k_random)
11:         if FITNESS(child)≤FITNESS(worst) then
12:             SWAP(child, worst)
13:         end if
14:     end while
15:     return bestSolution
16: end function
```

3. Dobiveni rezultati

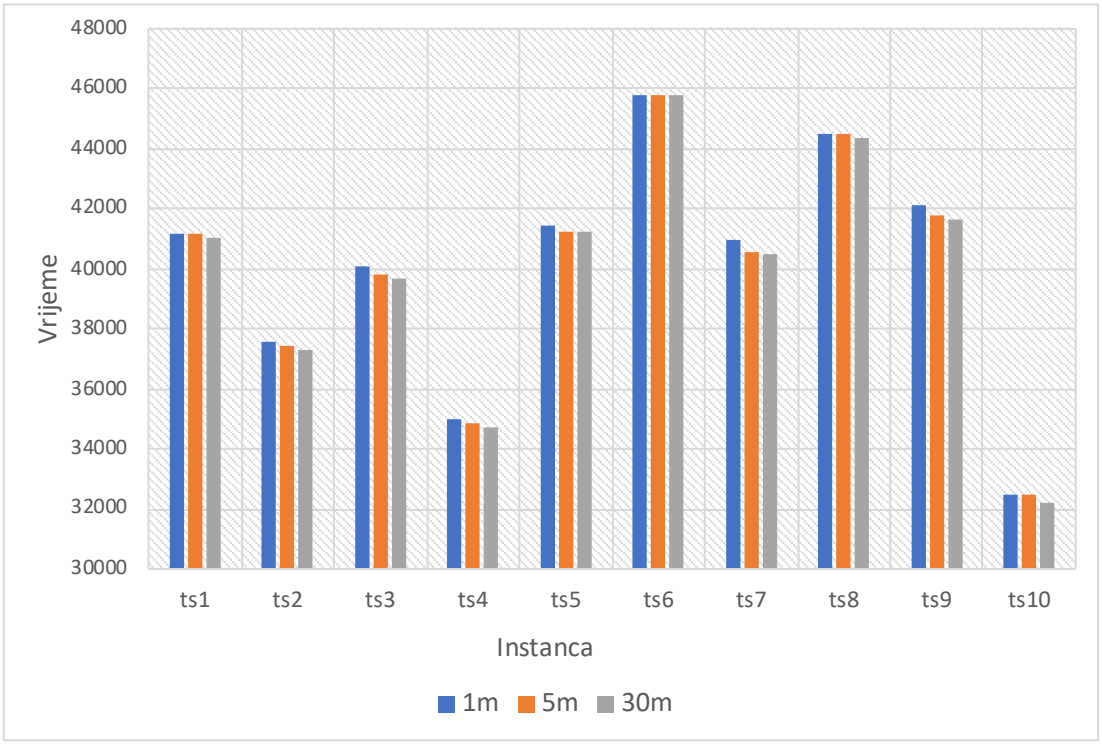
Sam pohlepni algoritam, odnosno heuristika, za generiranje valjanog rješenja iz niza parova (zadatak, stroj) nema hiperparametre te kao takva nije bila dio optimizacijskog algoritma no isprobane su razne implementacije. Kao najbolja heuristika se pokazala ona gdje su testovi raspoređivani na način da "rupa" između trenutnog testa i testa prije bude što veća jer na taj način će se prvo popuniti veliki odmaci između dva testa. Iduća bolja heuristika je uključivala isto "rupe", samo u obrnutom uvjetu u odnosu na prvu, to jest prvo bi se popunile manje rupe, zatim one veće, ali zbog mogućih ovisnosti o velikom broju resursa u većini slučajeva te velike "rupe" nije moguće smanjiti. Uz već navedene heuristike, također je isprobana i ona gdje su se prvo popunjavali prvi slobodni strojevi, no bila je lošija nego ostale dvije, za neke instance i znatno.

Nakon pohlepno generirane populacije gore korištenim heuristikama slijedio je genetski algoritam. Kao najbolji se pokazao eliminacijski genetski algoritam koji je koristio 3-turnirsku selekciju. Kao najbolji operator križanja pokazalo se križanje s dvije točke prekida uz vjerojatnost mutacije 0.35. Isproban je i generacijski genetski algoritam, no konvergirao je puno sporije u odnosu na eliminacijski te je pretraživanje često znalo jako varirati, stoga smo se u projektu posvetili eliminacijskoj inačici.

Najbolja dobivena rješenja uz prije opisanu heuristiku generiranja valjanog rješenja i genetski algoritam, prikazana su u tablici 3.1 te na grafu 3.1. Svaki redak tablice predstavlja jednu zadanu instancu, a svaki pojedini stupac najbolje rješenje nakon jedne minute, pet minuta i pola sata izvođenja genetskog algoritma. Također, u tablici 3.2 te na grafu 3.2 su prikazani brojevi evaluacije funkcije cilja do danog trenutka. Kao i kod tablice 3.1, redak predstavlja jednu zadanu instancu, a svaki pojedini stupac broj evaluacija funkcije nakon jedne minute, pet minuta i pola sata izvođenja genetskog algoritma.

Tablica 3.1: Vremena za pojedine instance

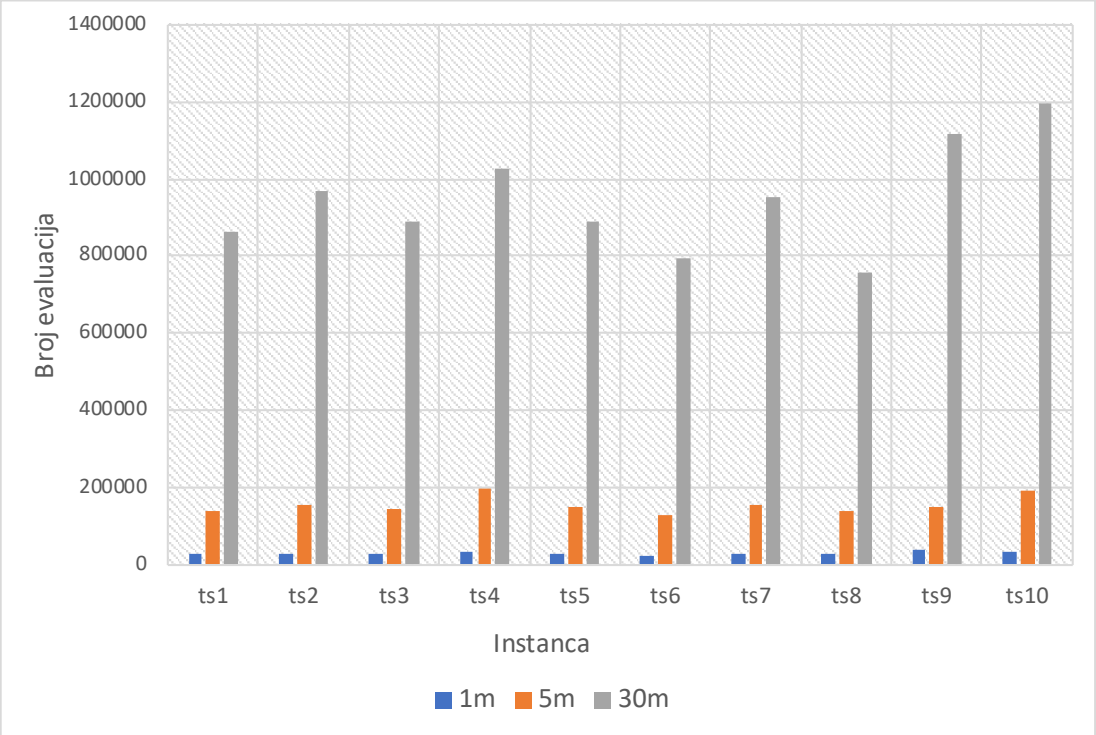
	1m	5m	30m
ts1	41162	41162	41020
ts2	37575	37437	37292
ts3	40080	39841	39644
ts4	35005	34857	34721
ts5	41471	41210	41210
ts6	45799	45799	45799
ts7	41000	40541	40472
ts8	44523	44523	44355
ts9	42088	41778	41645
ts10	32478	32462	32181



Slika 3.1: Vremena za pojedine instance

Tablica 3.2: Broj evaluacija pojedine instance do danog trenutka

	1m	5m	30m
ts1	27957	140768	863393
ts2	30880	157448	968336
ts3	28213	145032	887500
ts4	32671	198274	1028706
ts5	29947	149138	888751
ts6	24975	129993	791991
ts7	30853	156879	952876
ts8	27357	138378	756266
ts9	37029	152286	1118637
ts10	33298	190504	1194937



Slika 3.2: Broj evaluacija pojedine instance do danog trenutka

4. Zaključak

Eliminacijski genetski algoritam u kombinaciji s dobrom heuristikom pri pohlepnom algoritmu se pokazao iznimno dobrim za rješavanje problema raspoređivanja testova. Sva dobivena rješenja algoritam je uspio pronaći u nekoliko minuta, i to bez paralelizacije genetskog algoritma. Sva dobivena rješenja su vrlo blizu samoj donjoj granici uzevši u obzir pojedine resurse.

Kao poboljšanja bi se mogle ukomponirati prije opisane heuristike u jednu. Također, mogao bi se isprobati paralelni genetski algoritam, uz generacijsku inačicu s nešto kompleksnijim operatorima križanja. Isto tako bi se mogla isprobati nešto drugačija reprezentacija genotipa gdje bi implicitno bili uključeni i resursi.

5. Literatura

[1] Marko Čupić. *Raspoređivanje nastavnih aktivnosti evolucijskim računanjem.* Doktorska disertacija, Fakultet elektrotehnike i računarstva, 2011.

[2] Nina Skorin-Kapov. Predavanja s predmeta „Heurističke metode optimizacija", 2017.

[3] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi.* Fakultet elektrotehnike i računarstva, 2013.