SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1900

Raspoređivanje laboratorijskih vježbi paralelnim evolucijskim algoritmima

Zlatko Bratković

Zagreb, listopad 2011.

SADRŽAJ

1. Uvod

U gotovo svim poslovima koji se obavljaju, ljudi se susreću sa problemom raspoređivanja. Uvijek je potrebno odrediti na koji način i kojim redoslijedom se može uraditi neki posao. Takva tematika nije zaobišla ni obrazovne ustanove. Jedni su od takvih problema su i problemi raspoređivanja nastavnih aktivnosti.

Kroz povijest način raspoređivanja se mijenjao. Tako se u početku raspored slagao ručno, često su se koristile drvene ploče na koje su se slagali određeni predmeti u obliku pribadača. S vremenom drvene ploče su zamijenile klasične ploče sa kredom, a u najnovije doba ploču je zamijenilo računalo. Kako su se mijenjali alati kojima se radio raspored, tako se i napretkom ljudske vrste povećavalo znanje koje je trebalo prenijeti na mlade generacije. Sa većom količinom gradiva dolazilo se i do većih i kompleksnijih problema raspoređivanja. Problemi su tako postajali sve kompleksniji.

Na Fakultetu elektrotehnike i računarstva 2005/2006 godine uveden je bolonjski sustav. Studenti imaju veću slobodu u biranju predmeta, pa je moguće slušati predmete i sa različitih godina, ali i sa različitih zavoda. Prije uvođenja bolonjskog sustava, studenti su bili podjeljeni u velike grupe koje su imale jednake obveze većinom vezane samo za određeni zavod. Bolonjski sustav, budući da studentima daje veću slobodu u odabiru predmeta je smanjio te grupe, te raspršio studente istog smjera na više zavoda. Kako se raspored nekad radio na razini zavoda, stari sistem je mogao funkcionirati jer su zavodi praktično bili neovisni, no bolonjom takav sistem više nije mogao funkcionirati. Problem u takvom načinu razmišljanja je da su studenti mogli imati velik broj preklapanja u obavezama sa različitih zavoda.

Tradicionalno rješavanje problema ručnim raspoređivanjem više nije bilo moguće. Također zbog kompleksnosti samog problema nije moguće koristiti ni iscrpnu pretragu. Jedan od načina rješavanja takvih problema je korištenje heurističkih metoda. Ove metode neće nužno naći najbolje rješenje, ali će naći rješenje koje je u većini slučajeva prihvatljivo i što je važnije naći će ga u razumnom i prihvatljivom vremenu.

Ovaj rad je nastao kao nadogradnja projekta jAgenda koji je na Fakultetu elektrotehnike i računarstva pokrenut u jesen 2007. godine. Cilj projekta bio je riješiti

problem raspoređivanja laboratorijskih vježbi za sve studente, vodeći računa o svim njihovim obvezama, raspoloživosti resursa, te zahtjevima asistenata zaduženih za pojedine predmete. Sama aplikacija koristila je dvije metode : genetski algoritam i optimizacija kolonijom mrava. Obje metode pokazale su se u praksi uspješnim u rješavanju problema.

Rad dodaje mogućnost pokretanja genetskog algoritma u paralelnom okruženju, te ispituje prednosti odnosno nedostatke takve metode.

2. Evolucijsko računarstvo

Evolucijsko računarstvo je grana računalne znanosti koja se bavi proučavanjem heurističkih metoda optimizacije, a inspiraciju pronalaze u prirodnim procesima. Djelovanja metoda evolucijskog računarstva usmjerena su da heuristički i nasumično, ali i usmjereno smanje prostor rješenja, te pokušaju pronaći najbolje rješenje.

Umjetna evolucija je rađena po uzoru na prirodnu. Algoritmi koriste načela koja se mogu povezati sa prirodnim procesima što je vidljivo u tablici 2.1:

Tablica 2.1: Analogija prirode i algoritma

Priroda	Algoritam
Jedinka	Jedno rješenje problema
Populacija	Niz jedinki
Prirodni okoliš	Optimizacijski problem koji se nastoji riješiti

Također, koriste ste i procesi koji koji se događaju u prirodi, kao što su :

- križanje,
- mutacija,
- sposobnost preživljavanja jedinki,
- odumiranje jedinki,
- selekcija jedinki.

Svi procesi će podrobnije biti objašnjeni na primjeru korištenog genetskog algoritma.

2.1. Podjela Evolucijskih algoritama

Evolucijski se algoritmi po načinu djelovanja i reprezentaciji jedinke dijele u više podklasa:

- Genetski algoritam
- Genetsko programiranje
- Evolucijske strategije

2.2. Genetski algoritam

Genetski algoritmi zasnivaju se na procesima evolucije vrsta u prirodi [3]. U prirodi postoji određen broj jedinki neke vrste koje zajedno žive u istoj okolini. Neke jedinke su bolje od drugih, odnosno bolje su prilagođene okolini u kojoj žive, pa imaju i veću šansu za preživljavanje. Kako i u prirodi, ta svojstva jedinke se prenose nasljeđivanjem. Potomak dobiva dio svojstva, koja mogu biti dobra i loša, dio od jednog roditelja, a dio od drugog. Takvim načinom su jedinke sve prilagođenije uvjetima okoline.

Osim nasljeđivanja u algoritmima se koristi i drugi priroda proces u evoluciji, mutiranje. Mutiranje je važno jer ono donosi novi genetski materijal koji nijedan roditelj nema. Kao i u prirodi, te mutacije mogu biti dobre i loše, no generalno bez mutacije same jedinke bi s vremenom postala sve sličnije, ne bi se razlikovale, pa se nakon nekog vremena više ne bi mogao dogoditi neki napredak u prilagodbi. Sa strane algoritma, mutacija često služi i bijegu iz lokalnog minimuma koji se može dogoditi upravo zbog previše sličnih jedinki koje ne mogu više napredovati.

2.2.1. Implementacija genetskog algoritma

Genetski algoritmi se razlikuju od ostalih metoda pretraživanja prostora rješenja po tome što se pretraga ne vrši iz jedne početne točke u prostoru rješenja, nego iz više njih istovremeno. Skup točaka iz kojih počinje pretraga naziva se početna populacija. Veličina početne populacije ovisi o problemu koji se rješava i o njoj može ovisiti učinkovitost pretrage. Tijekom evolucije veličina populacije ostaje nepromijenjena: jedinke u populaciji koje zbog male vrijednosti dobrote izumru zamijenit će nove jedinke nastale primjenom genetskih operatora rekombinacije nad preživjelim jedinkama. Također, preživjele jedinke u populaciju mogu se promijeniti kao posljedica primjene operatora mutacije. Proces evolucije uvijek se ponavlja i u svakoj iteraciji primjenjuju se koraci kako je prikazano u algoritmu 1:

Algoritam 1 Genetski algoritam

$t = 0$
$P(0) = StvoriPocetnuPopulaciju()$
$ocijeni(P(0))$
repeat
  $t = t + 1$
  $K = izaberiJedinkuIz(P(t))$
  $makniJedinkuIzPopulacije(P(t), K)$
  $P'(t) = izaberiRoditeljeIz(P(t))$
  $D = krizaj(P'(t))$
  $mutiraj(D)$
  $ocijeni(D)$
  $dodaj(P(t), D)$
until zadovoljen uvjet zavrsetka

2.3. Genetsko programiranje

Genetsko programiranje razlikuje se od genetskog algoritma po tome što su jedinke genetskog algoritma programi koji stvaraju rješenje problema. Uobičajeno predstavljanje jedinki radi se pomoću strukture stabla. Pritom operator križanja zamjenjuje jedno podstablo drugim istovjetnim stablom iz druge jedinke, a mutacija mijenja značenje nekog čvora u stablu. Evaluacija se vrši na temelju kvalitete rješenja koja se dobivaju izvođenjem programa.

2.4. Evolucijske strategije

Kod evolucijskih strategija najveći je naglasak na mutaciji, a operator križanja se ponekad ni ne koristi.

3. Problem sveučilišnog rasporeda

Problem sveučilišnog rasporeda definiran je kao uređena četvorka (M,R,T,C) - Tablica 3.1 :

Tablica 3.1: uređena četvorka (M,R,T,C)

M	Nastavne aktivnosti koje je potrebno rasporediti) (predmeti, laboratorijske vježbe, ispiti itd.)
R	Resursi koje je potrebno rasporediti i dodijeliti nastavnim aktivnostima (npr. prostorije)
T	Termini u kojima je moguće raspoređivanje (dnevni, tjedni mjesečni i dr.)
C	Mogući zahtjevi i ograničenja nastavnih aktivnosti

Kao podskupine ovog problema moguće je definirati različite probleme raspoređivanja, kao što su problem rasporeda predavanja, problem rasporeda ispita, problem rasporeda laboratorijskih vježbi, te mnogi drugi. Probleme i načine rješavanja problema raspoređivanja može se naći u [5], [6], [2] i [7]

3.1. Problem raspoređivanja laboratorijskih vježbi

Problem raspoređivanja laboratorijskih vježbi (engl. LETP - *Laboratory exercise timetabling problem*) definiran je kao uređena šestorka (T,L,R,E,S,C) - Tablica 3.2

– Skup termina (T) je skup uzastopnih vremenskih kvanata. Kvant je najmanja dozvoljena vremenska jedinica, a svaki termin može se definirati kao niz jednog ili više uzastopnih vremenskih kvanata. Manja duljina vremenskog kvanta pruža veću zrnatost vremenskih odsječaka, dok se druge strane znatno povećava prostor pretraživanja. Stoga pri odabiru duljine vremenskog kvanta treba

Tablica 3.2: uređena šestorka (T,L,R,E,S,C)

T	skup termina u kojima je moguć raspored
L	skup ograničenih pomagala odnosno resursa u ustanovi
R	skup prostorija
E	skup laboratorijskih događaja za koje treba izraditi raspored
S	skup studenata koje treba rasporediti
C	skup ograničenja

vagati između bolje zrnatosti, te veličine prostora rješenja koje su obrnuto proporcionalne. U ovom radu odabrana je duljina kvanta od 15 minuta, a dopušteni termini su od ponedjeljka do petka, svakim danom od 08:00 do 20:00 sati

– Resursi (L) označavaju raspoloživa pomagala koja se mogu koristiti na raznim laboratorijskim vježbama. Tih pomagala imamo ograničen broj, pa je važno i o njima voditi računa. Kao najčešći primjer korišten u testovima, koristili su se programski paketi za koje postoji ograničen broj licenci, odnosno ograničen broj istovremenih korištenja istih. Licence su dopuštale korištenje u bilo kojim prostorijama, samo nisu dopuštale korištenje većeg broja licenci od dopuštenog.

– Prostorije (R) su definirane na sljedeći način:

  - *Broj radnih mjesta* zajednički je za sve vrste prostorija, no on može označavati različite stvari kao što su broj dostupnih računala, broj sjedaćih mjesta, broj stolova itd. Iako označava različite stvari, funkcionalno su one iste, uvijek se gleda koliko studenata se može dodijeliti prostoriji.
  - *Raspoloživost prostorija* označava kada je ta prostorija dostupna. Prostorije koje se koriste za laboratorijske vježbe mogu biti korištene i za druge svrhe, kao što su ispiti, predavanja, sastanci, te za mnoga druga događanja. Zbog toga, za svaku prostoriju je definiran skup vremenskih kvanata u kojima je prostorija raspoloživa.

– Događaji (E) su definirani je kao laboratorijske vježbe za neki predmet. Jedan događaj može biti raspoređen u jednom ili u više različitih termina. Tako se može dogoditi da laboratorijske vježbe za neke studente počinju rano ujutro u ponedjeljak a drugima počinju popodne u srijedu.

Događaju su pridruženi :

- trajanje događaja,
- dopušteni termini za događaj,
- skup studenata koji pohađaju događaj,
- broj studenata po radnom mjestu,
- skup demonstratora,
- skup dozvoljenih prostorija,
- broj asistenata po svakoj dopuštenoj dvorani,
- skup korištenih nastavnih pomagala,
- broj istovremeno korištenih prostorija,
- skup ograničenja koja se nameću događaju.

– Studenti (S) koji moraju biti raspoređeni imaju sljedeća svojstva :

- Raspoloživost studenta u terminima. Studenti, osim laboratorijskih vježbi u istom periodu mogu imati i druge obaveze poput ispita i predavanja, pa nisu dostupni algoritmu u svako vrijeme.
- Skup događaja koje student mora pohađati

– Skup ograničenja (C) podijeljen je u skup tvrdih ograničenja, koja se moraju nužno zadovoljiti, i skup mekih ograničenja, koja ne treba nužno zadovoljiti. Tvrda ograničenja definirana su na sljedeći način:

- Jedna prostorija može biti zauzeta sa najviše jednim događajem u istom trenutku. Ne smije se dogoditi da algoritam dodijeli dvije vježbe u isto vrijeme na istom mjestu.
- Da bi se događaju dodijelila prostorija u nekom vremenskom terminu, ona u tom terminu mora biti raspoloživa.
- Događaj se može održavati samo u prostorijama koje su dopuštene za taj događaj. Kao primjer tog ograničenja može se navesti vježba koja zahtijeva od studenata rad na računalu. Kako računala nemaju sve dvorane, potrebno je osigurati da se studenti ne smjeste u prostoriju koja možda jest prazna u to vrijeme, ali nije prikladna za samu vježbu.
- Prostorija ne može primiti veći broj studenata od definiranog kapaciteta za tu prostoriju. ovo čvrsto ograničenje se može donekle i olabaviti takozvanim *rezervama*. Rezerve su dodatna mjesta koja, u slučaju da to algoritmu odgovara, može popuniti. Tu je važno napomenuti da takva

opcija nije moguća za sve vježbe, jer one to zbog sistema rada ne dopuštaju (npr. zbog pisanja ispita na računalu), no postoje vježbe u kojima je takva opcija moguća (npr. predaja laboratorijskih vježbi gdje studenti kada predaju vježbu mogu prepustiti radno mjesto drugom studentu).

- Događaj se može održati samo u vremenu koje je definirano kao prikladno za održavanje događaja.
- Sve instance događaja moraju biti smještene unutar, za taj događaj, dopuštenih vremenskih kvanata. Takvo ograničenje posebno je zanimljivo ukoliko se na vježbama piši kratki ispiti, pa je zbog onemogućavanja stjecanja prednosti studenata koji kasnije polažu, potrebno ograničiti termine vježbe tako da se smanji ta mogućnost.
- Nastavna pomagala ne može biti korišteno istovremeno na većem broju radnih mjesta od dozvoljenog. Kod licenci takvo ograničenje se nikako ne može prekršiti, jer sustav blokira sve prekomjerne zahtjeve za korištenjem, pa se i algoritam mora strogo držati ograničenja.
- Kada se događaj održava istovremeno u različitim prostorijama, mora postojati dovoljan broj nastavnog osoblja za sve prostorije. Ne smije se dogoditi da se rezervira više dvorana nego što ima asistenata, odnosno osoblja.
- Studenti moraju biti prisutni na svim događajima u koje su uključeni. Ne treba dodatno objašnjenje s obzirom da je i cilj samog problema svrstati sve studente na sve vježbe.

– Skup mekih ograničenja sastoji se od:

- Student može biti prisutan na događaju jedino ako u vrijeme održavanja događaja nema drugih obaveza. Dakle, logično je da ukoliko student ima predavanje ili ispit u nekom periodu, da mu se u to vrijeme ne dodijeli neka laboratorijska vježba. Pokazalo se u praksi da je ovo najčešći oblik ograničenja koje se krši.
- Student može istovremeno biti prisutan na najviše jednom događaju.

4. Primjena genetskog algoritma na zadani problem

4.1. Stvaranje početne populacije

Kod stvaranja početne populacije najvažnije je da stvorena jedinka zadovoljava čvrsta ograničenja. Jedinke se generiraju nasumično poštujući čvrsta ograničenja. Za svaku jedinku prvo se generiraju događaji, a zatim se tako dobiveni raspored puni studentima. Za svaki događaj generira se potreban broj instanci (skup dvorana koje se nalaze u istom terminu), te se nakon toga te instance pune studentima. Postupak generiranja populacije prikazan je algoritmom 2.

Algoritam dodjele instanci ne mora uvijek uspjeti. Određen događaj ne mora uvijek imati na raspolaganju dovoljan broj dvorana za sve studente, pritom se pridržavajući čvrstih ograničenja. Najčešće se to događa zbog slučajnog odabira termina i dvorana za prethodno ubačene događaje. Takva situacija se rješava jednostavnim ponavljanjem cijelog algoritma jer, budući da je odabir slučajan, drugo raspoređivanje će slagati na drugačiji način, te će se dodijeljene dvorane i termini ispremiješati.

Postotak uspješnosti generiranja početnih rješenja regulira se i *važnošću* neke laboratorijske vježbe. Svi predmeti imaju jednake šanse biti prvi odabrani za dodjelu dvorana. Ukoliko algoritam bude previše puta neuspješan za neku laboratorijsku vježbu, ta vježba postepeno dobiva veće šanse da bude prije izabrana od drugih. Time se osigurava brže generiranje početne populacije.

Može se također dogoditi da i rješenja uopće ne mogu biti generirana. Takav slučaj se redovno događa zbog čvrstih ograničenja. Čvrsta ograničenja mogu biti postavljena tako da ne postoji rješenje koje će ih zadovoljiti.

Nakon raspoređivanja studenata uvijek postoje preklapanja između njihovih obaveza. Budući da preklapanje obaveza spada u kategoriju mekih ograničenja, kod generiranja populacije o tome se ne vodi računa.

Algoritam 2 Generiranje populacije

for svaki događaj $e \in E$ **do**
  D_e = skup odgovarajučih dana za događaj e;
  N_e = broj studenata koji pohađaju događaj e;
  while (D_e nije prazan) AND ($N_e > 0$) **do**
    d = odaberi i ukloni slučajni dan iz D_e;
    $T_{e,d}$ = skup valjanih vremenskih termina za događaj e u danu d;
    while ($T_{e,d}$ nije prazan) AND ($N_e > 0$) **do**
      odaberi slučajni vremenski termin t i makni ga iz $T_{e,d}$;
      stvori instancu događaja $i = (t, R_i, S_i)$ sa $R_i = \emptyset$ i $S_i = \emptyset$;
      for svaku prikladnu prostoriju $r \in R_e$ **do**
        if (r raspoloživa u terminu t) AND (r zadovoljava čvrsta ograničenja) **then**
          zauzmi prostoriju r i dodaj ju u R_i;
          popuni prostoriju studentima
          $N_e = N_e - kapacitet_r \cdot sprm_e$;
        end if
      end for
      pridruži instancu događaja i događaju e;
    end while
  end while
  if $N_e > 0$ **then**
    return dodjela neuspješna;
  end if
end for

4.2. Operator križanja

Križanjem dviju jedinki nastaje potomak tako da od svakog roditelja dobije dio instanci. Operator križanja djeluje tako da potomak nasljeđuje raspoređen cijeli događaj od prvog ili drugog roditelja. Za svaki događaj se nasumično bira od kojeg će roditelja biti naslijeđen. Tu može doći do kršenja čvrstih ograničenja, pa se za te događaje u instancama gdje postoje kršenja radi rekreiranje događaja, svojevrsna mutacija koja rješava problem čvrstih ograničenja. Uzima se nasumičan broj instanci od prvog roditelja, te svi ostali događaji od drugog roditelja koji dodavanjem ne krše čvrsta ograničenja. ostali događaji se ponovno kreiraju.

Ukoliko ni takva operacija ne može uvijek riješiti problem, odustaje se od križanja i pokušava se napraviti novo križanje nad istim roditeljima. Ukoliko ni nakon određenog broja pokušaja ne može doći do križanja, od križanja se odustaje, te jedinka predviđena za izbacivanje iz populacije ostaje u njoj.

Algoritam 3 Križanje

E_1 = odaberi slučajni podskup iz događaja iz E;
$E_2 = E - E_1$;
$E_n = \emptyset$;
for svaki događaj iz E_1 **do**
  dodaj događaj od roditelja 1 u dijete;
end for
for svaki događaj iz E_2 **do**
  if događaj od roditelja 2 ne krši čvrsta ograničenja u djetetu **then**
    dodaj događaj od roditelja 2 u dijete;
  else
    dodaj događaj u E_n;
  end if
end for
for svaki događaj iz E_n **do**
  dodijeli nasumično novi skup instanci za događaj i dodaj u dijete;
end for

4.3. Operator mutacije

Mutacija se događa na razini cijele jedinke. Prvo se odabire jedinka za mutiranje, a zatim se mutira određen broj događaja u njoj. Odabrani događaji se miču iz jedinke te ponovno kreiraju. Broj događaja koji se mutira je parametar koji se može mijenjati tijekom izvođenja. On se mijenja ukoliko je došlo do stagnacije. Stagnacija je definirana kao broj generacija u kojima se ne dogodi napredak u dobroti kod najbolje jedinke. Tada se broj događaja za mutaciju povećava, a kada se nakon toga dogodi napredak, broj događaja za mutaciju se vraća na početno definiran broj.

Algoritam 4 Mutacija

X= Broj događaja za mutiranje
E_x = odaberi slučajno X događaja iz E;
for svaki događaj iz E_x **do**
  izbriši događaj
  dodijeli nasumično novi skup instanci za taj događaj
end for

4.4. Funkcija dobrote

Kako sve jedinke zadovoljavaju čvrsta ograničenja, funkcija dobrote ovisi o mekim ograničenjima. U algoritmu umjesto funkcije dobrote koristimo njoj reverzan pojam funkcija *kazne*. Rješenje je bolje ako krši manje mekih rješenja. Kako su meka rješenja preklapanja studenata, to mogu biti preklapanja sa nekim vanjskim uvjetom (predavanje) ili preklapanja sa nekom drugom laboratorijskom vježbom. Kazna za jednog studenta definirana je sljedećom formulom:

$$K(s) = \sum_{t \in T} \frac{n(t)(n(t)-1)}{2} \tag{4.1}$$

gdje je T skup svih vremenskih kvanata, a $n(t)$ broj obaveza studenta u kvantu t. Svaki vremenski kvant koji nema preklapanja (kvant sa najviše jednom obavezom) ne pridonosi kazni, kvant sa dvije obaveze pridonosi kazni sa vrijednošću 1, a kvanti sa više od dvije obaveze pridonose kazni sve većim vrijednostima. Ukupna kazna jedinke dobiva se zbrajanjem kazni za sve studente unutar rasporeda:

$$K = \sum_{s \in S} K(s) \tag{4.2}$$

4.5. Selekcija

Selekcija služi čuvanju i prenošenju dobrih svojstava na sljedeću generaciju jedinki. Selekcijom se odabiru jedinke koje će sudjelovati u reprodukciji. Selekcija se vodi idejom da se dobri dijelovi jedinke prenose na sljedeće generacija, a loši dijelovi se postepeno odbacuju. Zbog sprečavanja preranog konvergiranja selekcija mora osigurati da se kod odabira jediniki za reprodukciju dopušta da se koriste i slabije prilagođene jedinke, naravno sa manjom vjerojatnošću odabira.

U ovom radu korišten je genetski algoritam s eliminacijskom turnirskom selekcijom. Eliminacijska turnirska selekcija nasumice odabire n jedinki, ali eliminira najlošiju i nadomješta je s djetetom dviju slučajno odabranih jedinki.

4.6. Elitizam

Postoji opasnost da se dobro rješenje dobiveno nakon puno iteracija uništi genetskim operatorima. Stoga je poželjno u nekim vrstama GA kreirati mehanizam zaštite najbolje jedinke. Takav mehanizam se naziva elitizam. Genetski algoritam s ugrađenim elitizmom, iz generacije u generaciju, monotono teži ka globalnom optimumu, odnosno najboljem rješenju jer se ne može dogoditi da se najbolja jedinka izgubi ili pogorša.

4.7. Uvjet završetka

Idealno rješenje nema konflikata, odnosno kazna mu je 0. Kako to nije uvijek moguće dostići, sam algoritam može dati rješenje i prije dostizanja minimuma, no tada je generalno potrebno intervenirati i zaustaviti algoritam, odnosno dojaviti mu da snimi trenutno najbolju jedinku. pokazalo se u praksi da postoje zadaci za koje se ne može doći do rješenja bez kazne, no postoje i zadaci u kojima nije moguće odrediti da li takvo rješenje postoji ili ne.

5. Paralelni genetski algoritam

5.1. Opis

Sa sve većom računalnom moći i sve boljim povezivanjem računala postavlja se pitanje može li se to dodatno iskoristiti paralelnim obrađivanjem. Cilj paralelizacije genetskog algoritma je skraćenje vremena izvođenja. Idealno, genetski algoritam bi trebao trajati onoliko puta kraće koliko računalo ima procesora. U realnim situacijama naravno ipak se ne dostižu takvi rezultati zbog mnogih čimbenika kao što su arhitektura računala, struktura mreže, ali i odabir samog algoritma za paralelizaciju. Pri takvom postupku, mora se paziti da se ne naruše osnovna svojstva genetskog algoritma.

5.2. Vrste paralelnih genetskih algoritama

Kod paralelnog algoritma potrebno je odrediti što će se paralelizirati i na koji način. Kako algoritam iz iteracije u iteraciju obavlja isti posao u kojem imamo genetske operatore i funkciju dobrote, paralelizacija se može raditi na razini iteracije. Postoje dva pristupa paraleliziranja genetskih algoritama:

- standardni pristup – paralelizirati genetske operatore i izračunavati vrijednosti funkcije cilja paralelno.
- dekompozicijski pristup – podijeliti populaciju na manje dijelove - podpopulacije i obavljati cijeli genetski algoritam nad podpopulacijama.

U ovom diplomskom radu korišten je drugi pristup iz razloga što se izračun funkcije dobrote računa automatizirano prilikom svake promjene jedinke, a ne odvojeno od genetskih operatora.

Postoje nekoliko mogućih razina paraleliziranja genetskih algoritama: paraleliziranje na razini populacije, na razini jedinki te na razini evaluacije. Prema razini paralelizacije, postoje tri osnovna načina podjele genetskog algoritma na podzadatke:

– Krupnozrnata podjela je podjela velike populacije na manje dijelove – podpopulacije. U ovom se slučaju radi o dekompozicijskom pristupu ili o višepopulacijskom paralelnom genetskom algoritmu koji je raspodijeljen tako da se paralelno izvodi nekoliko genetskih algoritama nad manjim populacijama.

– Sitnozrnata podjela je ekstremni oblik podjele velike populacije na podpopulacije veličine jedne jedinke. Svaki procesor obavlja genetske operatore nad njemu dodijeljenom jedinkom i nad susjednim jedinkama. Ovakva podjela je također predstavnik višepopulacijskog modela.

– Moguće je paralelno obavljati izračunavanje vrijednosti funkcije dobrote, dok se genetski operatori izvode sekvencijski.Takva podjela se naziva podjelom na "gospodara" i "sluge". Gospodar obavlja genetski algoritam nad zajedničkom populacijom, stoga je to jednopopulacijski model. U svakoj iteraciji sluge paralelno izračunavaju vrijednosti funkcije dobrote nakon što gospodar obavi svoj sekvencijski dio posla.

U ovisnosti o načinu podjele algoritma na podzadatke, postoje tri osnovna modela paralelnih genetskih algoritama:

– Distribuirani genetski algoritam (DGA) ili raspodijeljeni genetski algoritam sastoji se prema krupnozrnatoj podjeli od nekoliko podpopulacija pa se naziva još i višepopulacijski genetski algoritam. To je najpopularniji model paralelnih genetskih algoritama, a korišten je i u ovom radu.

– Masovno paralelni genetski algoritam (MPGA) se prema sitnozrnatoj podjeli sastoji od N procesora koji predstavljaju N jedinki. Dakle, veličina populacije je jednaka broju procesora. Svaki procesor obavlja genetske operatore nad svojom jedinkom i nad susjednim jedinkama.

– Globalni paralelni genetski algoritam (kratica: GPGA) je predstavnik podjele na gospodara i sluge. Paralelni dio posla obavljaju sluge, dok sekvencijski gospodar. Sluge su najčešće zadužene samo za evaluaciju jedinki, dok gospodar obavlja sve ostale genetske operatore.

Ta tri osnovna modela se mogu međusobno kombinirati ili nadograditi nekom drugom metodom optimiranja. Osim tih kombinacija, postoji i trivijalni paralelni genetski algoritam (TPGA). Radi se o više genetskih algoritama koji se paralelno obavljaju na nekoliko potpuno nezavisnih računala (računala ne moraju biti povezana) kako bi se, primjerice, statistički obradili eksperimentalno dobiveni rezultati ili kako bi se odredio optimalan skup parametara.

5.3. Raspodijeljeni genetski algoritam (DGA)

Raspodijeljeni genetski algoritam je oblik paralelnog genetskog algoritma koji radi sa raspodijeljenom populacijom [4]. Umjesto izvođenja genetskog algoritma nad jednom velikom populacijom, raspodijeljeni genetski algoritam djeluje na više manjih populacija. Svaka populacija nalazi se u jednom čvoru. Čvor može biti računalo u mreži ili procesor u višeprocesorskom sustavu. Genetski algoritmi u pojedinim čvorovima ne moraju biti jednaki, ali optimizacijski problem koji rješavaju mora biti isti. Čvorovi međusobno razmjenjuju jedinke u nadi da će jedinka koju dobiju usmjeriti GA u još neistražene dijelove područja rješenja te tako postići bolje rješenje.

Algoritam DGA 5 Raspodijeljeni genetski algoritam

$t = 0$
$P(0) = StvoriPocetnePopulacije()$
$ocijeni(P(0)$
repeat
  ZA SVAKU PODPOPULACIJU
  $t = t + 1$
  $K = izaberiJedinkuIz(P(t))$
  $makniJedinkuIzPopulacije(P(t), K)$
  if nema jedinki koje su stigle od drugih podpopulacija **then**
    $P'(t) = izaberiRoditeljeIz(P(t)$
    $D = krizaj(P'(t))$
  else
    jedinku koja je stigla od druge podpopulacije stavi u populaciju;
  end if
  $mutiraj(D)$
  $ocijeni(D)$
  $dodaj(P(t), D)$
  if broj iteracija MOD period migracije == 0 **then**
    $M = izaberiJedinkuZaMigraciju(P(t)$
    $PosaljiJedinkuSusjedima(M)$
  end if
until zadovoljen uvjet zavrsetka

5.3.1. Migracija

Migracija i način njene implementacije imaju snažan utjecaj na uspješnost DGA. Postupak migracije uvodi nove parametre u GA: migracijski interval, migracijsku stopu, strategiju odabira boljih jedinki, strategiju odabira jedinki za eliminaciju i topologiju razmjene jedinki.

5.3.2. Migracijski interval

Migracijski interval određuje broj iteracija između dvije migracije. Migracijski interval može biti konstantan (unaprijed određen) ili slučajan. U slučaju nekonstantnog migracijskog intervala, za analizu rada GA-a, važan je prosječan faktor migracijskog intervala. Interval može biti i uvjetovan. U tom slučaju migracija se odvija samo kada je ispunjen unaprijed određeni uvjet. Što je frekvencija migracije manja to su populacije izoliranije. Ukoliko su populacije izolirane, tada GA pretražuje različite prostore rješenja. Ukoliko su populacije potpuno izolirane, tada se DGA degradira na svojstva GA-a sa veličinom populacije veličine jedne subpopulacije DGA.

5.3.3. Migracijska stopa

Migracijska stopa određuje broj jedinki koji se razmjenjuje između podpopulacija. Migracijska stopa utječe na različitost populacija. Ukoliko se velik broj jedinki prebacuje iz jedne populacije u drugu, tada će te populacije postajati sve sličnije. U tom slučaju podpopulacije će se usmjeriti na isti prostor rješenja, te će DGA biti manje učinkovit od klasičnog GA-a s istom veličinom populacije. Uobičajeno je u migraciji izmijeniti samo jednu jedinku između susjednih podpopulacija.

5.3.4. Odabir jedinki za migraciju

Strategija odabira jedinki značajno utječe na selekcijski pritisak, a time i na brzinu konvergencije algoritma. Selekcijski pritisak veći je što bolje jedinke imaju veću vjerojatnost preživljavanja u odnosu na loše jedinke i obrnuto.

Prilikom razmjene potrebno je odabrati jedinke za razmjenu i jedinke koje će nove jedinke zamijeniti. U oba slučaja mogu se koristiti strategija slučajnog odabira ili strategija odabira najboljih, odnosno najlošijih jedinki. U primjeni se najčešće koristi izbor najboljih jedinki za razmjenu, a slučajan odabir ili odabir najgorih jedinki za eliminaciju.

5.3.5. Topologija migracije

Topologija razmjene jedinki može biti statička ili dinamička. Uobičajeno se koristi statička topologija, koja se definira na početku, te se ne mijenja do kraja izvođenja GA-a. Ako se koristi dinamička topologija, tada GA slučajnim odabirom odlučuje koje će podpopulacije poslati svoje jedinke kojim podpopulacijama.

Brzina širenja nekog rješenja među podpopulacijama ovisi o broju susjeda kojima se šalje jedinka. Podpopulacije koje primaju jedinke od neke druge podpopulacije nazivaju se njenim susjedima. Što topologija ima više susjedstva, to je veća brzina kojom se šire rješenja po drugim subpopulacijama.

5.4. Učinkovitost algoritma

Algoritam kao podlogu koristi GA koji je napravljen u sklopu jAgenda projekta [1] koji je na Fakultetu elektrotehnike i računarstva započeo 2007. godine. Parametri genetskog algoritma koji su korišteni mogu se vidjeti u tablici 5.1.

Tablica 5.1: Parametri genetskog algoritma

parametar	vrijednost
veličina populacije	30
selekcijski algoritam	turnirski odabir
turnirska veličina	10
vjerojatnost mutacije jedinke	55 %
početni/maksimalni nivo mutacije	1 / 10
granica za aktivaciju stagnacije	5
kriterij zaustavljanja	10000 generacija ili minimalni kazna = 0

Uz korištenje parametara genetskog algoritma, korišteni su i parametri paralelnog genetskog algoritma. Parametri se mogu vidjeti u tablici 5.2.

Tablica 5.2: Parametri paralelnog genetskog algoritma

parametar	vrijednost
veličina podpopulacije	30
migracijski interval	100 iteracija
migracijska stopa	1 jedinka
odabir jednike za migraciju	najbolja jedinka %
topologija	lančano povezivanje
dijeljenje jedinki	asinkrono

U testiranjima koristili su se stvarni podaci [1].

[1] Problemi su zahtjevi za izradu rasporeda za različite cikluse iz akademske godine 2008/2009 na Fakultetu elektrotehnike i računarstva. U radu su označeni kao *C3* i *C6*.

5.4.1. Brzina konvergiranja

Kao najvažniji faktor učinkovitosti može se smatrati brzina izvođenja algoritma. Grafički prikaz takve usporedbe može se vidjeti na slikama 5.1 i 5.3 za primjer *C3*, te slikama 5.2 i 5.4 za primjer *C6*.

Zanimljivost kod primjera *C3* je da za 8 i 16 paralelnih instanci algoritam ima približno isto rješenje, a kod *C6* takav slučaj imamo kod 4 i 8 paralelnih instanci, dok kod 16 instanci ipak dobivamo znatnije ubrzanje.

5.4.2. Ubrzanje u odnosu na slijedni algoritam

Budući da su se za migraciju koristile posebne dretve u samom procesu, slanje i primanje nije utjecalo na brzinu izvođenja aplikacije. Zato ubrzanje nije gledano u vremenskoj bazi, nego po broju generacija potrebnih da se dođe do globalnog minimuma.

Ako ubrzanje definiramo kao :

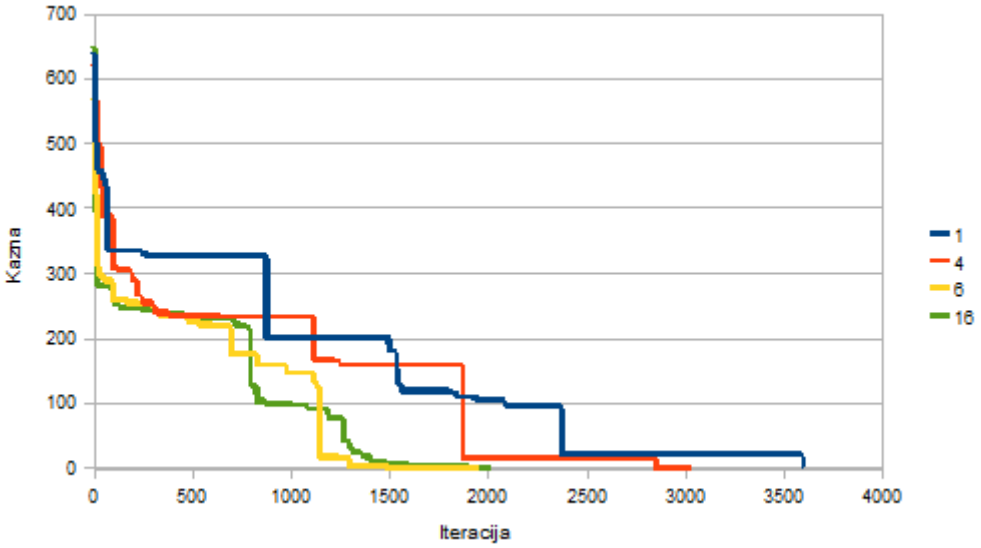
N= broj generacija slijednog algoritma

M= broj generacija paralelnog algoritma

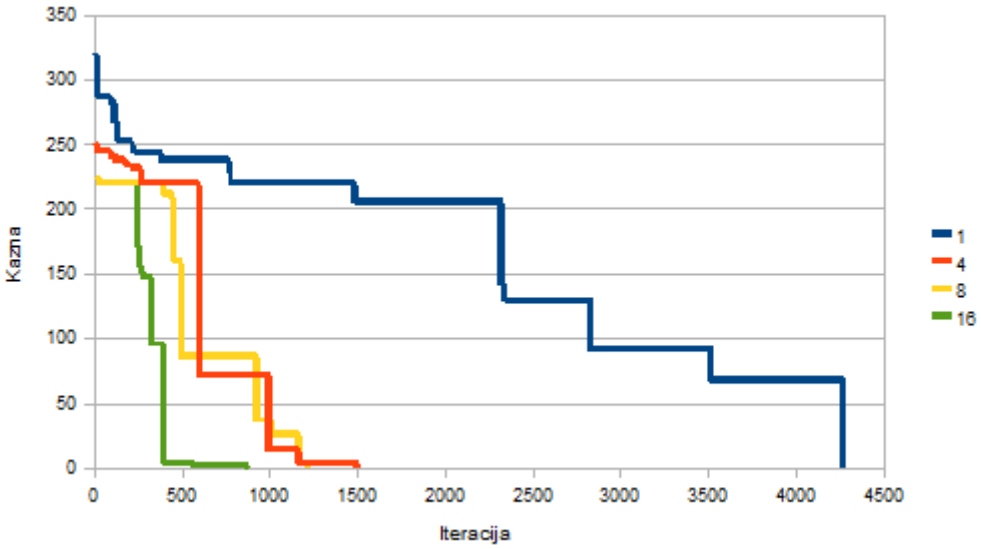
$Ubrzanje = \frac{N}{M}$;

Tablica 5.3: Ubrzanje paralelnog algoritma u odnosu na slijedni

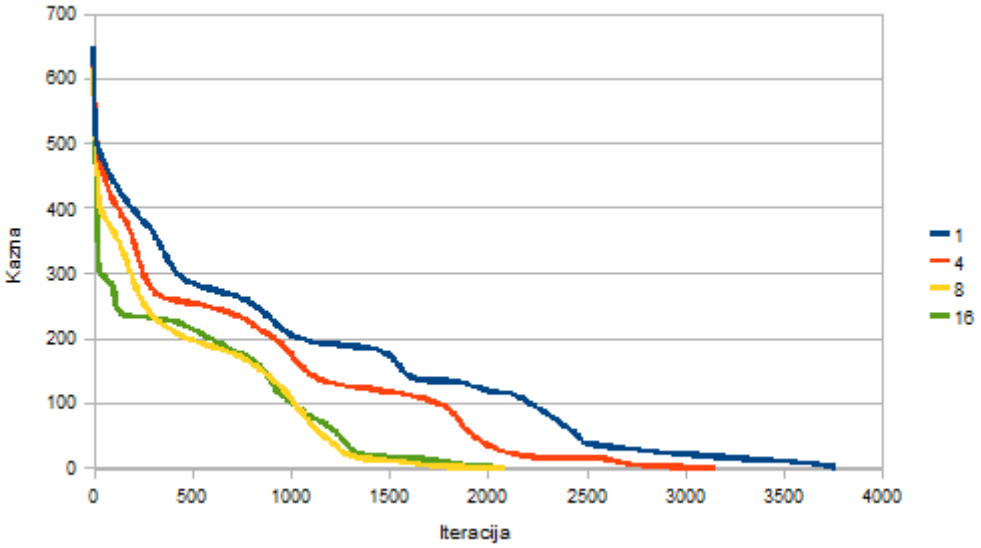
ciklus	broj procesa	ubrzanje
C3	4	1.19
C6	4	2.78
C3	8	1.8
C6	8	4,85
C3	16	1.83
C6	16	6.17



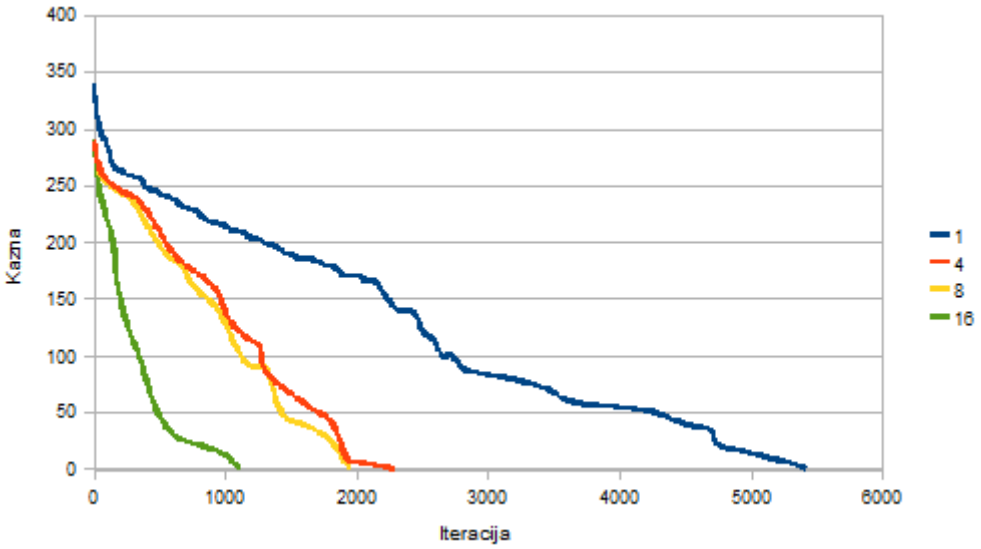
Slika 5.1: usporedba broja instanci paralelnog algoritma i brzine pronalaska odnosno konvergencije rješenja (primjer *C3*). Na slici je prikaznm najbolji rezultat



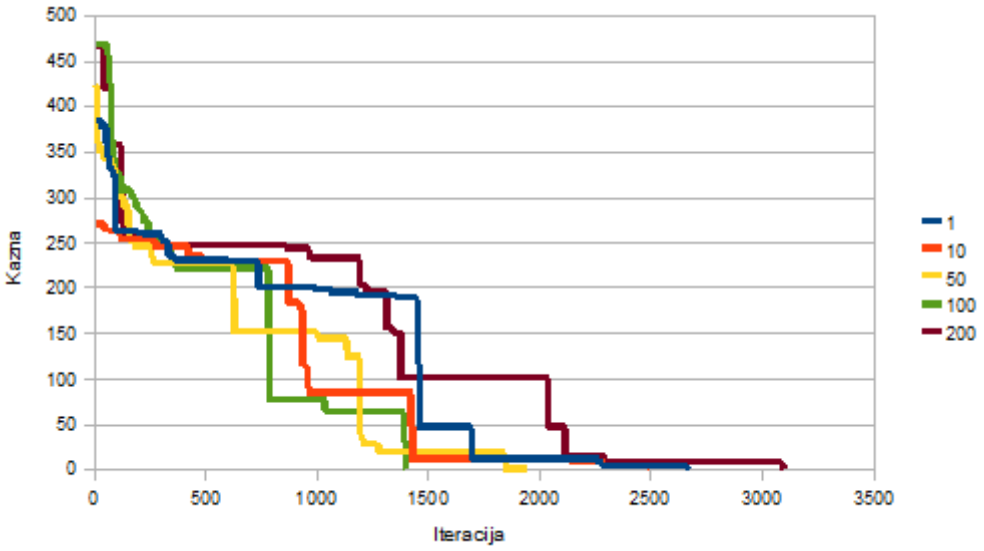
Slika 5.2: usporedba broja instanci paralelnog algoritma i brzine pronalaska odnosno konvergencije rješenja (primjer *C6*). Na slici je prikazan najbolji rezultat



Slika 5.3: usporedba broja instanci paralelnog algoritma i brzine pronalaska odnosno konvergencije rješenja (primjer *C3*). Na slici je prikazan prosječni rezultat od 10 dobivenih rješenja



Slika 5.4: usporedba broja instanci paralelnog algoritma i brzine pronalaska odnosno konvergencije rješenja (primjer *C6*). Na slici je prikazan prosječni rezultat od 10 dobivenih rješenja



Slika 5.5: Utjecaj migracijskog intervala na brzinu pronalaska rješenja (primjer *C3*)

5.4.3. Migracijski interval

Migracijski interval u ovom radu definiran je kao konstanta, odnosno ne mijenja se tijekom izvođenja algoritma. Na slici 5.5 korišten je paralelni GA sa 8 instanci, te populacijom veličine 30. Može se vidjeti da interval ne smije biti ni premalen ni prevelik ukoliko želimo dobiti optimum.

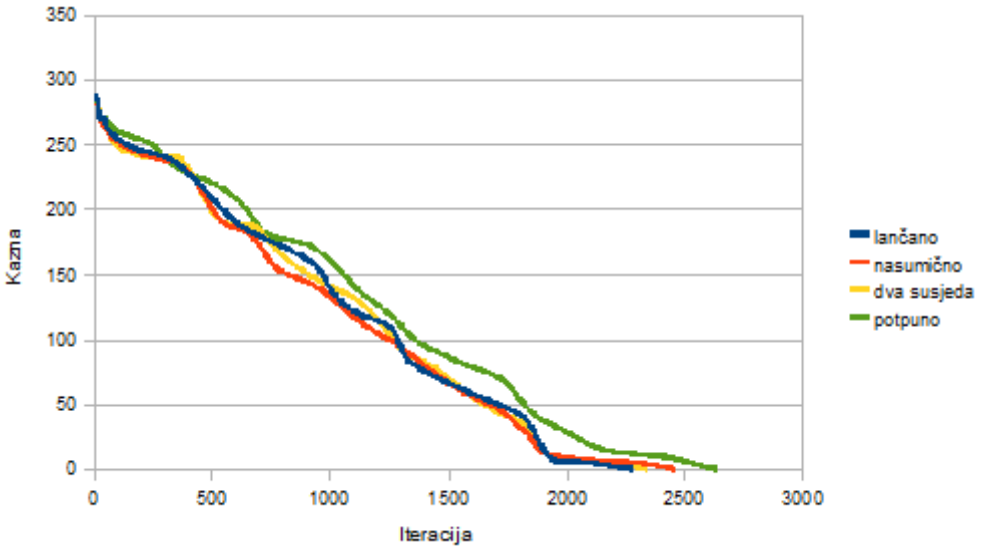
Takvo ponašanje je i očekivano, jer ako je interval premalen, smanjuje se utjecaj paralelnog algoritma, te on postaje sličniji klasičnom. Isto tako ni prevelika brojka nije dobra jer onda dobijemo previše izolirane podpopulacije koje se, svaka zasebno, također ponaša kao klasičan GA.

5.4.4. Topologija migracije

Topologija migracije u ovom algoritmu nije imala velik utjecaj na rezultate osim u slučaju potpunog povezivanja. Usporedbu možemo vidjeti na slici 5.6.

Korišteno je

- lančano povezivanje - svaka instanca ima svojeg lijevog susjeda od kojeg prima jedinku, te desnog susjeda kome šalje svoju jedinku.
- nasumično povezivanje - svaki puta bira se instanca kojoj se šalje jedinka.
- povezivanje sa dva susjeda - svaka instanca šalje svoju jedinku dvjema instancama.



Slika 5.6: usporedba topologija (primjer *C6*). Na slici je prikazan prosječan rezulat iz 10 pokretanja

– potpuno povezivanje - svaka instanca šalje svakoj instanci svoje rješenje. Takav oblik povezivanja pokazao se manje neučinkovit jer se prebrzo dolazilo do lokalnih minimuma, u kojima je zapinjao cijeli algoritam, što se paralelizmom željelo izbjeći.

6. Zaključak

Paralelizacija, kao način ubrzanja algoritma nekog problema, pokazala se uspješnom metodom. Iako brzina kojom se dobivaju rješenja nije linearno proporcionalna sa dodavanjem broja instanci, nego manja, može se reći da postoji određeno ubrzanje.

Kao je već prije pokazano [1], algoritam se najbolje ponaša za veličinu populacije oko 30 jedinki, te da povećanje tog broja ne pridonosi brzini ili učinkovitosti rješavanja problema, pokušao se taj broj povećati preko paralelnog algoritma. Svaka takva podpopulacija, odnosno dio veće cijele populacije imao je broj jedinki isto 30, algoritam je tada radio optimalno, a takva stvorena rješenja dijelila su između procesa algoritma. Takav način rješavanja problema pokazao se kao nešto brži nego klasičan GA.

Pokazalo se i da topologija migracije ne utječe bitno na brzinu rješavanja.

Što se tiče kvalitete rješenja, jedna instanca, odnosno klasičan GA, jednako je sposobna riješiti problem, samo nešto sporije od njene paralelne verzije.

Raspoređivanje laboratorijskih vježbi paralelnim evolucijskim algoritmima

Sažetak

Problem raspoređivanja vrlo je čest u svim područjima djelovanja, tako i u visokoškolskim ustanovama. Takvi problemi vrlo često su složeni, kako samom problematikom, tako i vremenskom složenošću. Također postoji mogućnost da potpuno rješenje koje zadovoljava sve uvjete niti ne postoji. Evolucijski algoritmi se upravo na takvoj vrsti problema pokazuju uspješnima zbog toga što pronalaze rješenje prihvatljive kvalitete u razumnom vremenu. U radu je dan kratak pregled evolucijskih algoritama, s naglaskom na genetski algoritam. Dana je definicija problema raspoređivanja laboratorijskih vježbi na Fakultetu elektrotehnike i računarstva te je opisan sustav za rješavanje tog problema temeljen na genetskom algoritmu. Rad je fokusiran na paralelni genetski algoritam kojim se nastoji ubrzati pronalazak zadovoljavajućeg rješenja. Na kraju se iznose rezultati ispitivanja učinkovitosti algoritma ovisno o vrijednosti odabranih parametara algoritma.

Ključne riječi: paralelni genetski algoritam, raspoređivanje, laboratorijske vježbe, migracija, evolucijski algoritam

University course timetabling using parallel genetic algorithm

Abstract

Scheduling problem is very common in all areas of human activities, including higher education institutions. Such problems are often complex, with definition problem, and time complexity. It is also possible that a complete solution that satisfies all the conditions do not exist. Evolutionary algorithms in this kind of problem shows successful because they find a solution acceptable quality in reasonable time. The paper gives a brief overview of evolutionary algorithms, with emphasis on genetic algorithm. Work defined the scheduling problem of laboratory practice at the Faculty of Electrical Engineering and Computer Science, and describes a system for solving this problem based on genetic algorithms. The work is focused on parallel genetic algorithm, which seeks to accelerate the finding satisfactory solutions. Finally, the results of testing the effectiveness of the algorithm are shown depending on the values of selected parameters of the algorithm.

Keywords: paralel genetic algorithm, timetabling, laboratory exercices, migration

LITERATURA

[1] Z. Bratković, T. Herman, V. Omrčen, M. Čupić, i D. Jakobović. University Course Timetabling with Genetic Algorithm: A Laboratory Excercises Case Study. *Evolutionary Computation in Combinatorial Optimization*, stranice 240–251.

[2] E.K. Burke i S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 127(2):266–280, July 2002. URL `http://ideas.repec.org/a/eee/ejores/v140y2002i2p266-280.html`.

[3] Marin Golub. Genetski algoritam - prvi dio. . URL `http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf`.

[4] Marin Golub. Genetski algoritam - drugi dio. . URL `http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf`.

[5] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 2007. doi: 10.1007/s00291-007-0097-0.

[6] B. McCollum. University timetabling: Bridging the gap between research and practice. U H Rudova E Burke, urednik, *PATAT 2006 — Proc. 6th Int. Conf. on the Practice And Theory of Automated Timetabling*, stranice 15 – 35. Masaryk University, 2006. URL `citeseer.ist.psu.edu/mccollum06university.html`.

[7] Andrea Schaerf. A survey of automated timetabling. U *115*, stranica 33. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 30 1995. URL `citeseer.ist.psu.edu/schaerf95survey.html`.

7. Dodatak

Aplikaciju možemo testirati pomoću programa koji se nalaze u *test* direktoriju na priloženom CD-u. postoje dvije aplikacije :

- TheServer.jar - posrednik, služi za definiranje parametara, logiranje rezultata i davanje podataka klijentima.
- ParalelClientMain.jar - klijent aplikacija, svaki proces pokreće jednu. Ulazni parametri aplikacije su IP adresa, te port servera. Primjer pokretanja procesa : *java - jar ParalelClientMain.jar "10.192.192.116" 2001*.

Zbog preciznijih izračuna potrebno je prvo pokrenuti klijent aplikacije, a tek onda pokrenuti server koji automatski tada šalje podatke procesima.