SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 212

Stvaranje rasporeda sati genetskim algoritmima

Vinko Bedek

Zagreb, lipanj, 2008.

Sadržaj

1. Uvod 1

2. Genetski algoritmi 2

2.1 Osnove genetskog algoritma 3

2.2 Prikaz jedinki 4

2.3 Teorem shema 7

2.4 Selekcija 8

2.5 Križanje 10

2.6 Mutacija 12

2.7 Parametri 13

3. Problem stvaranja rasporeda sati 15

3.1 Definicija problema 15

3.2 Prikaz kromosoma 18

3.3 Genetski operatori 19

3.3.1 Selekcija 19

3.3.2 Križanje 20

3.3.3 Mutacija 20

3.4 Deterministično poboljšanje rješenja 21

3.5 Evaluacija dobrote 21

4. Programsko rješenje 23

4.1 Podešavanje genetskog algoritma 27

4.2 Ostale mogućnosti 28

5. Rezultati 29

5.1 Utjecaj vjerojatnosti mutacije 30

5.2 Utjecaj veličine populacije 31

5.3 Utjecaj vrste mutacije 32

5.4 Utjecaj vjerojatnosti i vrste križanja 34

6. Zaključak 37

7. Literatura 38

1. Uvod

Genetski algoritmi su postali popularnim oruđem u današnjem svijetu rješavanja problema iz različitih domena. Iako postoje i druge metode optimiranja poznate javnosti, genetski algoritmi dobivaju najviše pozornosti, pojavljuju se na televiziji i u novinama. Financiranje projekata sa genetskim algoritmima nadmašuje financiranja projekata na temelju drugih metaheurističkih metoda [2]. Jedan od razloga je zasigurno i sam naziv, genetski. Privlačnost se ne može osporiti. Genetski algoritmi se često predstavljaju javnosti s argumentom da će raditi zbog toga što preslikavaju mehanizme evolucije. Budući je evolucija s vremenom stvorila nas, prema nama vrhunac evolucije, takav argument se često prihvaća kao valjan bez obzira o kakvoj se kompleksnosti problema radi. Iako se razmišljanjem u argumentu brzo pronalaze rupe, recimo potrebno vrijeme za dolaženje do rješenja, genetski algoritmi su zaista moćno optimizacijsko oruđe kao što je pokazano u mnogim radovima ali i uspješnim primjenama izvan akademske zajednice.

U ovom su radu genetski algoritmi primijenjeni na problem stvaranja rasporeda sati. Kao i većina problema za koje se koriste genetski algoritmi, problem stvaranja rasporeda sati je NP-kompletan problem. Treba napomenuti da je stvaranje rasporeda koji će zadovoljiti sve uvjete najčešće nemoguće. Za rješenje koje će genetski algoritam dati je bitno da zadovoljava zahtjeve koji se ne smiju prekršiti. Kvaliteta tog rješenja će pak biti određena ostalim zadovoljenim zahtjevima.

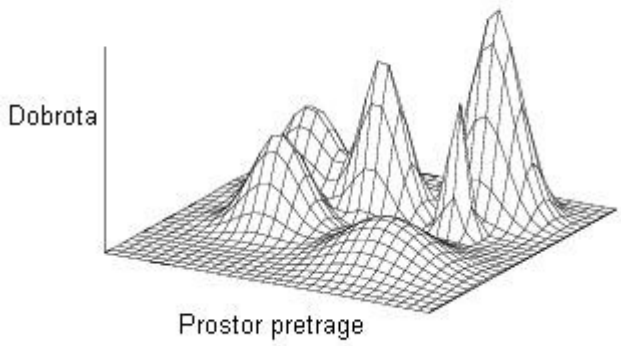
Rad se sastoji od 5 poglavlja. Nakon ovog, uvodnog, slijedi drugo poglavlje gdje se opisuju genetski algoritmi. Opisane su najvažnije karakteristike njihovog rada, a dan je i kratak uvod u teoriju koja stoji iza njihovog ponašanja. U trećem poglavlju se definira problem stvaranja rasporeda sati. Objašnjeni su zahtjevi koje raspored mora zadovoljiti, te je napravljen mali pregled postojećih metoda stvaranja rasporeda. U nastavku trećeg poglavlja je dan opis ostvarenja genetskog algoritma koji rješava problem stvaranja rasporeda sati. Četvrto poglavlje daje opis grafičkog sučelja, načina unosa podataka i mogućnosti koje programsko ostvarenje nudi. U petom poglavlju su prvo opisani podaci nad kojim je izvršeno testiranje, nakon čega se daje analiza postignutih rezultata za različite kombinacije parametara genetskog algoritma.

2. Genetski algoritmi

Genetski algoritmi su općeniti algoritmi pretrage i optimizacije inspirirani evolucijom. Zbog mogućnosti primjene na široki skup problema, kao i zbog jednostavnih principa, genetski algoritmi dobivaju sve više sljedbenika iz različitih područja. Metode inspirirane evolucijom su postojale i prije genetskih algoritama. Ideja upotrebe populacije rješenja za rješavanje praktičnih inženjerskih problema je promatrana nekoliko puta kroz 1950 i 1960. Ingo Rechenberg je 1960ih stvorio evolucijske strategije za potrebe optimizacije funkcija realnih varijabli. Algoritam se sastojao od jednog roditelja i jednog djeteta, dijete je nastajalo mutacijom roditelja. Fogel, Owens i Walsh su stvorili evolucijsko programiranje koje je imalo rješenja kandidate prikazane uz pomoć deterministčkih konačnih automata. Nad njima se primjenjivala mutacija, odnosno algoritam je nasumično mijenjao dijagrame prijelaza stanja. No za postojanje genetskih algoritama u današnjem smislu je odgovoran John Holland. 1960ih je proučavao mehanizme prilagodbe u prirodnim sustavima i načine na koje bi mogao te mehanizme prenijeti na računalne sustave. Godine 1975. u knjizi *Adaptation in Natural and Artificial Systems* predstavlja genetske algoritme. Holland je slijedio terminologiju biologije, kodirana rješenja je nazvao kromosomima, osnovne jedinice kodiranih rješenja genima, a vrijednosti koje te jedinice mogu poprimiti alelima, mjesto gena u kromosomu lokus [1]. Za razliku od ostalih metoda inspiriranih evolucijom koje su naglašavale selekciju i mutaciju, Holland je uveo i stavio naglasak na križanje koje je s razvojem genetike i razumijevanjem strukture DNA sve više dobivalo na važnosti. Na početku genetski algoritmi nisu izazvali veće zanimanje. Doktorskim radom Kena De Jonga, studenta kojem je Holland bio mentor to se promijenilo. On je u radu stavio naglasak genetskih algoritama na optimizacijske probleme, a ne na sposobnost prilagodbe, nešto što je kod Hollandovog rada na adaptivnim sustavima imalo veću važnost. Danas se većina radova vezanih uz genetske algoritme zasniva na optimizaciji. Kasnije je De Jong, za kojeg možemo reći da je glavni "krivac" za takav pogled na domenu upotrebe genetskih algoritama više puta objašnjavao da oni nisu zapravo optimizatori funkcija [8].

2.1 Osnove genetskog algoritma

Prema Hollandu genetski algoritam se sastoji od populacije jedinki nad kojima se izvršava selekcija za određivanje jedinki koje su zaslužile sudjelovati u stvaranju nove generacije. Početna se populacija dobiva slučajno, te je disperzija početne populacije najveća kroz cijeli tok genetskog algoritma. Nova generacija se stvara križanjem roditelja koji su za to odabrani prema svojoj dobroti, odnosno sposobnosti preživljavanja u svojem staništu kod stvarne evolucije. U svrhu odabira mora postojati neki način, odnosno funkcija vrednovanja dobrote jedinke. Na temelju te funkcije možemo dobiti izgled prostora pretrage. Bolje jedinke će se nalaziti na brdima, a gore u dolinama. Primjer prostora pretrage za funkciju sa dvije varijable se nalazi na slici 2.1. Prostori pretrage nisu uvijek jednostavnog oblika, čak i za jednostavnije probleme mogu postojati mnogobrojni vrhovi jedan pored drugog odijeljeni sa dubokim dolinama.



Slika 2.1 Prostor pretrage

Cilj je naravno pronaći najviši vrh koji se naziva globalni optimum, dok se ostali vrhovi nazivaju lokalnim optimumima. Na novostvorenim jedinkama može doći do mutacije. Preživljavanjem boljih jedinki kod stvarne evolucije dolazi do prenošenja genetskog materijala zaslužnog za preživljavanje na novu generaciju, što za posljedicu ima poboljšanje prilagođenosti cijele generacije. Ista se stvar primjećuje i kod genetskih algoritama.

Osnovne građevne jedinice genetskog algoritma su:

- Populacija jedinki
- Selekcija
- Križanje
- Mutacija

Četvrti element prema Hollandu, inverzija, se danas rijetko koristi. Vidljivo je da su genetski algoritmi u svojoj osnovi jednostavni algoritmi čija prava snaga proizlazi kao i kod evolucije u paralelizmu i raznolikosti jedinki. Jedinke su zapravo kromosomi i predstavljaju jedno moguće rješenje.

Jednostavan genetski algoritam radi na sljedeći način:

1. Generira se početna populacija sa n slučajno odabranih jedinki iz prostora pretrage
2. Izračuna se dobrota svake jedinke
3. Ponavlja se dok se ne stvori n jedinki
    - Izabere se par jedinki iz trenutne populacije, vjerojatnost odabira ovisi o dobroti jedinke. Jedna jedinka može biti više puta odabrana. S određenom vjerojatnošću dolazi do križanja, novonastale jedinke ili jedinka se stavlja u novu generaciju. Ako do križanja nije došlo djeca su kopija roditelja.
    - Mutirati djecu s vjerojatnošću križanja
4. Zamijeniti trenutnu generaciju s novostvorenom
5. Provjeriti uvjet završetka, ako nije zadovoljen nastaviti sa 2. korakom

Svaka iteracija se naziva generacijom. Uvjet završetka može biti provjera konvergencije, neko vremensko ograničenje ili broj generacija.

2.2 Prikaz jedinki

Prije nego se može početi sa izradom genetskog algoritma potrebno je odlučiti kakav će se prikaz koristiti. U najjednostavnijem i najčešćem slučaju radi se o binarnom prikazu, koji se najranije koristio i uz koji je vezana većina teorije o genetskim algoritmima. Iako

najzastupljenije, ovo kodiranje ima slabosti zbog postojanja takozvanih Hammingovih litica gdje bliske jedinke unutar prostora pretrage imaju veliku Hammingovu udaljenost [3], npr. euklidska udaljenost između jedinke kodirane sa 011111 i jedinke sa prikazom 100000 je 1, ali Hammingova udaljenost je 6. Genetski algoritam treba kroz evoluciju invertirati sve bitove da bi došao do rješenja koje je minimalno bolje od postojećih. Za većinu problema koji se susreću u industrijsko-inženjerskom svijetu binarna reprezentacija nije dovoljna zbog otežane prilagodbe problema prikazu. Kod odabira prikaza važno je da zauzima što manje mjesta uz uvjet da sadrži sve važne informacije potrebne za dobivanje rješenja. Očito je da odabir prikaza ima ogroman utjecaj na daljnji razvoj algoritma. Na temelju kakve vrijednosti gen može poprimiti prikazi se mogu podijeliti na:

- Binarni prikaz
- Prikaz brojevima s pomičnim zarezom
- Prikaz cijelim brojevima
- Prikaz objektima, odnosno strukturama

Prikaz brojevima s pomičnim zarezom se pokazao superiornijim od binarnih kod funkcijskih optimizacija. Prikaz cijelim brojevima, odnosno permutacijski prikaz se najviše koristi kod kombinatoričkih optimizacijskih problema budući se traži najbolja permutacija. Prema strukturi prikaza, prikazi se dijele na jednodimenzionalne i višedimenzionalne, u većini slučajeva se koristi jednodimenzionalni prikaz, ali za neke probleme je prirodnije koristiti višedimenzionalne. Genetski algoritmi rade sa dva tipa prostora. Jedan je prostor kodiranja, odnosno genotipski prostor, a drugi je prostor rješenja, odnosno fenotipski prostor. Mutacija i križanje rade nad genetskim prostorom, a selekcija i evaluacija jedinki rade nad prostorom rješenja. Između tih prostora postoji preslikavanje koje može biti 1:N, N:1 i 1:1. Kod preslikavanja se može ustvrditi da se neki kromosom iz genotipskog prostora ne može preslikati u fenotipski. Takav kromosom je ilegalan. Ilegalnost potječe iz prikaza i upotrebe genetskih operatora ako koristimo prikaz prilagođen problemu i operatore koji to nisu. U tom slučaju možemo ili odbaciti ilegalne jedinke, ili popraviti jedinke na neki način. Drugo rješenje je koristiti genetske operatore prilagođene problemu.

Svojstva na temelju kojih se određuje kvaliteta prikaza [3]:

- Neredundantnost

  Preslikavanje između genotipskog i fenotipskog prostora bi trebalo biti 1:1. Ako imamo N:1 preslikavanje trošimo vrijeme na traženje među istovjetnim rješenjima, jedinke su različite u genotipskom prostoru pa genetski algoritam ne zna da su te jedinke jednake. Zbog takve situacije u algoritmu može doći do prerane konvergencije. Najgori je slučaj kada postoji 1:N preslikavanje jer treba odrediti koje će se rješenje iz fenotipskog prostora uzeti.

- Legalnost

  Bilo koja permutacija prikaza odgovara nekom mogućem rješenju. Ovim svojstvom osigurano je korištenje postojećih operatora. Jasno je da se ovo svojstvo na razini prikaza za složenije probleme teško zadovoljava.

- Kompletnost

  Ovo svojstvo govori da se do bilo koje točke prostora rješenja može doći pretragom genetskim algoritmom. Kada prikaz ne bi posjedovao takvo svojstvo postojala bi mogućnost da se kvalitetna rješenja nalaze upravo u nepristupačnom dijelu prostora pretrage.

- Lamarckianovo svojstvo

  Značenje alela za neki gen je neovisno o kontekstu. Tim svojstvom je osigurano da prinos kvaliteti rješenja zbog tog gena neće opasti mijenjanjem ostalih gena kroz proces evolucije.

- Kauzalnost

  Male varijacije nastale u genotipskom prostoru uslijed mutacije bi trebale rezultirati malim varijacijama u fenotipskom prostoru, drugim riječima mutacija bi trebala očuvati strukturu susjedstva (jedinki sličnih promatranoj po nekom kriteriju) u fenotipskom prostoru. Procesi pretrage koji ne uništavaju susjedstva posjeduju jaku kauzalnost, dok slaba kauzalnost znači veće promjene u fenotipskom susjedstvu jedinke malim promjenama u genotipskog prostora.

2.3 Teorem shema

Kod nekih ostalih heurističkih pristupa, npr. simuliranog kaljenja, postoji znatna količina teoretske analize. Osim što je važna za shvaćanje rada, teorija može dati smjernice u implementaciji i području uporabe [2]. Broj primjena genetskih algoritama raste sa njihovom popularnošću, no razvoj teorije genetskih algoritama je puno sporiji. Teorija o načinu rada genetskog algoritma je još uvijek fragmentirana i postoje različite perspektive. Najstarije teoretsko razmatranje koja se odnosi na originalni način rada genetskog algoritma dao je Holland. U svom radu Holland uvodi pojam shema. Sheme se odnose na bitovne nizove, a sastoje se od 0 , 1 i *. Npr. shema ***0**1 predstavlja sve jedinke koje imaju četvrti gen 0 i zadnji gen 1. Jedinke koje zadovoljavaju shemu se nazivaju instance sheme. Red sheme je broj nepromjenjivih bitova, a u slučaju prethodnog primjera iznosi 2. Najveća udaljenost između nepromjenjivih bitova unutar sheme se naziva definirajući razmak. Razlog uvođenja shema je bio u svrhu formalnog opisavanja ideje građevnih jedinica pomoću kojih genetski algoritam dolazi do rješenja. Na temelju shema i proučavanja rada genetskog algoritma nastao je teorem sheme. Teorem sheme kaže da broj jedinki koje sadrže shemu niskog reda, kratkog definirajućeg razmaka te iznadprosječne dobrote raste eksponencijalno [6]. Spomenuta dobrota sheme se računa implicitno kod računanja dobrote instanci te sheme unutar populacije. U teoremu shema su sadržana destruktivna svojstva mutacije i križanja. Zašto sheme niskog reda i kratkog razmaka? Ako imamo roditelje sa istom shemom, križanjem ćemo dobiti opet istu shemu, no ako su sheme različite postoji vjerojatnost da novonastala jedinka neće biti instanca sheme niti jednog roditelja. Što je definirajući razmak manji veća je vjerojatnost da će shema biti očuvana, odnosno da će se svi bitovi koji određuju shemu križanjem biti preneseni na dijete. Razlog za rast shema sa niskim redom je u primjeni mutacije. S većim brojem bitova koji moraju ostati nepromijenjeni kako bi shema ostala očuvana raste vjerojatnost da će jedan od njih biti promijenjen mutacijom. Na temelju teorema shema dolazi se do zaključka da genetski algoritam dolazi do rješenja sastavljajući kratke sheme, odnosno da kratke sheme s iznadprosječnom dobrotom predstavljaju građevne blokove. Navedeni zaključak se naziva hipoteza građevnih blokova. Točna definicija glasi: Genetski algoritam pretražuje prostor rješenja nizajući sheme niskog reda, kratke definirane dužine i iznadprosječne dobrote, zvane građevni blokovi [6]. Teorem shema je od svog nastanka

višestruko kritiziran. Michael Vose je proučavajući utjecaj mutacije kod genetskog algoritma pokazao da malena promjena u vjerojatnosti primjene mutacije može imati značajan utjecaj na daljnji put genetskog algoritma kroz prostor pretrage koji analiza preko shema ne može objasniti. Vose isto tako navodi da je fokus na broj instanci jedinki neke sheme neispravan. Za genetski algoritam je puno važnije koje instance sheme se pojavljuju u sljedećoj generaciji [2]. Pod provjerom se našla i hipoteza građevnih blokova. Goldberg je predstavio ideju *deceptivnih* funkcija, funkcija koje bi navele genetski algoritam na krivi put uskraćujući dobre građevne blokove. Očekivano, genetski algoritam radi loše, i daje dokaz o ispravnosti hipoteze. S druge strane ako se promatra rad genetskog algoritma nad "*Royal Road*" funkcijom može se doći do drugačijih zaključaka. "*Royal Road*" funkcija daje genetskom algoritmu sve potrebno da dođe do rješenja na način koji je očekivan prema hipotezi. Rezultati su bili porazni za genetski algoritam, na problemu gdje je trebao pokazati snagu građevnih blokova je po pitanju performansi poražen od najjednostavnijeg *hill-climbera* [2].

2.4 Selekcija

Kao i u prirodi uloga selekcije kod genetskog algoritma je odabrati jedinke koje će sudjelovati u stvaranju nove populacije. Cilj je stvoriti populaciju koja je bolja od stare, odnosno sastoji se od jedinki sa boljom dobrotom. Kako bi se to moglo postići, genetski materijal dostupan za stvaranje te generacije mora biti kvalitetan. Ta osobina se ostvaruje davanjem veće vjerojatnosti jedinkama sa većom dobrotom za sudjelovanje u stvaranju nove generacije. Treba naglasiti da se u svrhu dobivanja što bolje jedinke ne smiju zanemariti one sa lošijom dobrotom, jer postoji mogućnost da se u njima nalazi genetski materijal koji može pogodovati boljoj dobroti kod djeteta, slično kao i u prirodi. Isto kao što postoji mogućnost gubitka dobrog genetskog materijala koji se nalazi u lošim jedinkama zbog smanjene vjerojatnosti sudjelovanja u reprodukciji novih jedinki, postoji i mogućnost da izgubimo najbolju jedinku budući sve stare jedinke nestaju. Iako je genetski materijal prenesen, ne postoji garancija da u novoj generaciji imamo jedinku koja je po svojoj dobroti jednaka najboljoj u prijašnjoj generaciji. Dolazi se do ideje da prenesemo najbolju, ili nekoliko najboljih jedinki u novu generaciju. Taj mehanizam se naziva elitizam.

Selekcije dijelimo na generacijske i eliminacijske. Generacijske se odnose na zamjenu cijele generacije novom. Za vrijeme stvaranja nove generacije potrebno je zadržati staru za izbor roditelja. U eliminacijskoj selekciji izbrišemo određeni broj jedinki. Odabir jedinki koje se brišu je težinski. Hollandov genetski algoritam je koristio jednostavnu proporcionalnu selekciju. Vjerojatnost da će jedinka biti odabrana je proporcionalna njezinoj dobroti. Vizualno se metoda može predočiti kao kotač ruleta gdje svaka jedinka ima svoj udio na temelju dobrote. Implementacija je sljedeća:

- Sumirati dobrote svih jedinki, T
- Ponavljati za broj željenih jedinki
    - Izabrati nasumičan broj između 0 i T, R
    - U petlji prolaziti kroz sve jedinke i sumirati dobrote, kada suma postane veća od R , zadnja jedinka čiju smo dobrotu zbrojili je odabrana

U svrhu ubrzanja se kod jednostavne selekcije umjesto opisanog, linearnog načina pretrage može koristiti binarno pretraživanje. Statistički, kod ove metode svaka jedinka u generaciji ima broj djece određen dobrotom. No, veličina populacije kod genetskih algoritama često nije dovoljno velika kako bi se statistički model preslikao u stvarnu situaciju. Godine 1987. James Baker je predložio stohastičku univerzalnu metodu. Odabir roditelja se vrši samo jednom na temelju slučajnog broja, a *n* roditelja se odabire deterministički. Najveći problem kod proporcionalnih selekcija leži u tome što na početku genetskog algoritma postoji malen broj jedinki koje dobrotom odudaraju od prosjeka. Primjenom proporcionalne selekcije algoritam stavlja preveliki naglasak na pretraživanje prostora pretrage u kojem su te jedinke. Isto tako, kasnije kada jedinke postaju sve sličnije i razlike u dobroti su male, algoritam ne stavlja dovoljan naglasak na odabir boljih jedinki. Može se reći da brzina evolucije ovisi o varijanci dobrote u populaciji. U tu svrhu uvode se skalirajuće metode. Od vremena De Jongovih radova primjena skalirajućih metoda je postala široko prihvaćena [3]. Predloženo je nekoliko načina skaliranja koji se dijele na statičke i dinamičke skalirajuće metode. Ako je relacija transformacije konstantna tada se radi o statičkim metodama, ako je pak relacija podložna promjeni u ovisnosti o nekim vremenski promjenjivim faktorima govorimo o dinamičkim skalirajućim metodama.

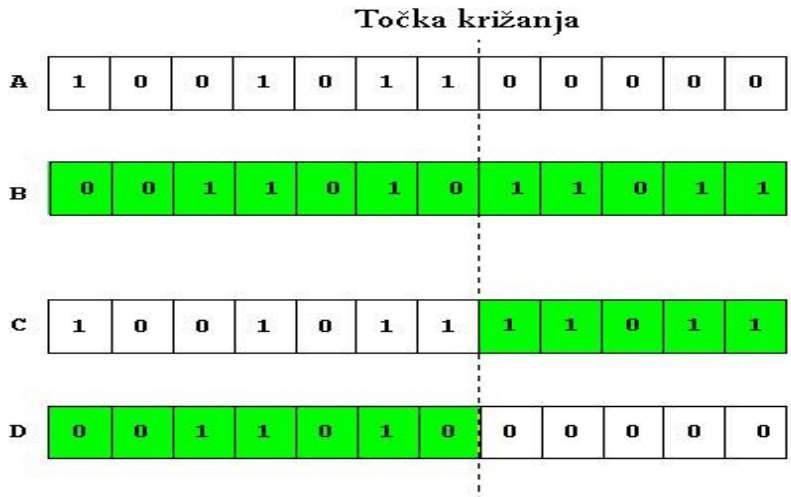
Primjer jedne dinamičke skalirajuće metode je sigma skaliranje. Jedna mogućnost ostvarivanja je prikazana preko jednadžbe (2.1) [1].

$$\text{ExpVal}(i,t) = \begin{cases} 1 + \dfrac{f(i) - \bar{f}(t)}{2\sigma(t)} & \text{if } \sigma(t) \neq 0 \\ 1.0 & \text{if } \sigma(t) = 0, \end{cases} \tag{2.1}$$

Iz formule je vidljivo da će selekcijski pritisak kroz tok genetskog algoritma ostati ujednačen. Budući se promatra razlika srednje dobrote i dobrote jedinke skalirana sa standardnom devijacijom na početku algoritma bolje jedinke neće biti jako udaljene od srednje vrijednosti, odnosno kod selekcije će vjerojatnost njihovog odabira biti manja. Kasnije, kada standardna devijacija opadne, bolje jedinke će imati veću vjerojatnost odabira od lošijih nego što bi to imale kod selekcije samo na temelju dobrote. Sljedeći način kojim se izbjegava prerana konvergencija je rangirajuća selekcija. Vjerojatnost odabira ovisi o rangu jedinke unutar populacije. Jasno je da za posljedicu algoritam može sporije doći do rješenja, ali isto tako zadržavanje raznolikosti unutar generacije može dovesti do kvalitetnijeg rješenja. Još jedan nedostatak rangirajućih selekcija je vrijeme potrebno za sortiranje cijele populacije. Metoda koja je efikasnija od rangirajućih, a zadržava dobra svojstva vezana uz selekcijski pritisak (odnosno koliko udio djece u generaciji će potjecati od roditelja sa visokom dobrotom) se naziva turnirska. Odabire se k jedinki te se najbolja uzima za roditelja. Kod eliminacijske turnirske selekcije se odabiru 3 ili više jedinki, a jedinka s najmanjom dobrotom se zamjenjuje sa djetetom dvije najbolje jedinke.

2.5 Križanje

Križanjem se stvaraju nove jedinke iz genetskog materijala roditeljskih kromosoma. Jednim križanjem se mogu stvoriti dvije nove jedinke ili samo jedna. Samo križanje je izvedeno na način da zadovoljava prikaz kromosoma. Najjednostavnije križanje je križanje s jednom točkom prekida prikazano na slici 2.2. Kromosom jednog roditelja se prekida u jednoj točki te se na prekinuti dio nadovezuje preostali dio iz drugog roditelja.



Slika 2.2 Križanje sa jednom točkom prekida

Ova vrsta križanja ima svojih nedostataka. Ne postoji mogućnost kombiniranja svih shema, npr. ako imamo shemu jednog roditelja 1****1 i drugog ***1** nikako križanjem ne možemo dobiti shemu 1**1*1. Isto tako sheme sa relativno dugim definirajućim razmakom u odnosu na ukupnu duljinu kromosoma će s vremenom nestati jer će ih operator križanja uništiti, jer prema teoremu shema algoritam sa jednostavnim križanjem daje prednost shemama sa manjim definirajućim razmakom. Naziv pojave je pozicijska sklonost i točna definicija glasi: sheme koje se mogu stvoriti ili uništiti križanjem ovise o poziciji bitova unutar kromosoma [1]. Davanje prednosti shemama sa manjim definirajućim razmakom ima još jednu posljedicu. Bitovi koji su blizu bitova koji čine željenu shemu imaju veću vjerojatnost za očuvanje kroz tijek algoritma. Osim toga primijećeno je da ovakvo križanje zadržava bitove na krajnjim pozicijama budući da je najmanji dio kromosoma koji se može izmjenjivati prvi ili zadnji bit. U svrhu eliminiranja ovih pojava Holland je u svom originalnom genetskom algoritmu koristio operator inverzije. Svrha inverzije nije bila stvaranje novog genetskog materijala, već samo preslagivanje gena na drugačiji način za izbjegavanje gore spomenutih efekata križanja sa jednom točkom prekida. Jedinka prije i poslije inverzije ne izgledaju isto sa stanovišta križanja i ako ih križamo sa nekom drugom jedinkom rezultati neće biti isti. Nakon operatora inverzije jedinka zadržava sva svoja svojstva. Način na koji se to postiže je pridjeljivanje svakom genu broj koji označava lokus na kojem se nalazi.

Zbog svojih nedostataka u očuvanju shema kao i ograničenju u stvaranju novih koristi se križanje s dvije točke prekida gdje se točke odabiru nasumično a segmenti unutar tih točaka se kod roditelja zamjenjuju. Osim križanja sa dvije točke prekida moguće je koristiti bilo koju vrijednost između 1 i n-1 gdje je n broj gena. Ako se koristi križanje sa n-1-om točkom prekida tada se radi o uniformnom križanju s maskom 1010101... odnosno naizmjence se uzimaju geni prvo iz jednog roditelja pa iz drugog. Križanje koje koristi masku sa jednakim vjerojatnostima se naziva p-uniformno križanje gdje je p vjerojatnost da se gen uzme iz prvog roditelja. Za općenitiju slučaj sa p=0.5 se koristi samo naziv uniformno križanje. Opisane vrste križanja se odnose na bitovne prikaze, i iako se mogu iskoristiti i za druge prikaze u većini slučajeva je potrebno pronaći ili stvoriti križanje koje je prilagođeno vrsti problema koji se pokušava riješiti.

2.6 Mutacija

Primjenom samo križanja nad jedinkama se kroz određeni broj generacija dolazi do prerane konvergencije, genetski materijal koji je algoritmu na raspolaganju postaje homogen, te nema raznolikosti između jedinki. Kao što vrste u prirodi ne bi preživjele bez mutacija nad genima, odnosno prilagodbi okolini, tako niti jedinke u genetskom algoritmu neće dosegnuti svoj potencijal jer će zaglaviti na nekom lokalnom optimumu unutar prostora pretrage. Bitovi na pojedinim mjestima unutar svih jedinki će postati isti i, ako rješenja sa boljom dobrotom na tim mjestima imaju drugačiji raspored nula i jedinica, ne postoji način da algoritam dođe do tih rješenja bez upotrebe mutacije. Iako je uloga križanja neupitna vidljivo je da bez mutacije efikasan genetski algoritam ne može postojati. Najjednostavnija mutacija kod binarnog prikaza je invertiranje bita. Evaluacija o izvršavanju operatora mutacije se kod takvog prikaza obavlja za svaki bit i radi se o vjerojatnostima od 0.005 do 0.01. Miješajuća mutacija za razliku od jednostavne mutacije radi nad kromosomima, a ne nad genima; odabiru se dvije pozicije unutar kromosoma i svi geni između se ispremiješaju, ako se geni nanovo generiraju radi se o potpunoj miješajućoj mutaciji. Potpuna mutacija izmiješa sve bitove nekog slučajno odabranog kromosoma; zapravo se radi o miješajućoj mutaciji sa fiksnim točkama početka i kraja intervala unutar koje će se izmiješati geni postavljenima na početak i kraj kromosoma. Ako se radi o permutacijskom prikazu za potrebe nekog kombinatoričkog problema, najjednostavnija

mutacija koja se može primijeniti je zamjena pozicija dvaju gena. Napredniji primjer mutacije kod permutacijskog prikaza bi bio PBM (*Position Based Mutation*). Uzima se gen s neke lokacije i stavlja se na neko drugo nasumično odabrano mjesto.

Kromosom prije PBM: 1 6 4 9 3 **7** 2 5 8 0

Kromosom poslije PBM: 1 **7** 6 4 9 3 2 5 8 0

2.7 Parametri

Za zadovoljavajuće rezultate i zadovoljavajuću brzinu pronalaska takvih rezultata potrebno je pronaći odgovarajuće vrijednosti parametara. Osnovni parametri koji se podešavaju kod svakog genetskog algoritma su veličina populacije, vjerojatnost križanja i vjerojatnost mutacije. Bez previše razmišljanja se dolazi do nekih osnovnih zaključaka o veličinama određenih parametara. Premalen broj jedinki nije dovoljan za održavanje potrebne količine genetskog materijala, prevelik broj s druge strane bespotrebno usporava izvođenje. Vjerojatnost mutacije jasno mora biti malih razmjera ako želimo zadržati konvergenciju, preveliki iznos tjera algoritam na nasumično kretanje prostorom pretrage, maleni iznosi vjerojatnosti križanja sprječavaju kombiniranje dobrih svojstava roditelja. Izvršeno je mnogo istraživanja o optimalnim vrijednostima parametara, no kako je svaki algoritam drugačiji i vrijednosti koje će davati najbolja rješenja će se razlikovati od slučaja do slučaja. Jedan od ranijih istraživanja o vrijednostima parametara je izvršio De Jong, a prema njegovim rezultatima najbolja veličina populacije je između 50 i 100 jedinki, vjerojatnost križanja oko 0.6, a vjerojatnost mutacije je odredio kao 0.001 po bitu. Ovi parametri su prihvaćeni u javnosti kao polazni za različite vrste problema. 1986 Grefenstette je uz pomoć genetskog algoritma odlučio pronaći optimalne vrijednosti parametara. Radio je nad De Jongovim setovima problema, te je kao parametre koristio De Jongove preporuke. Kao rezultat je dobio jedinku koja je postavila veličinu populacije na 30 jedinki, vjerojatnost križanja na 0.95, vjerojatnost mutacije na 0.01, generacijski jaz (udio populacije koji se mijenja novim jedinkama) je bio 1, a koristila je i elitizam [1]. Osim fiksnih iznosa moguće je napraviti i adaptivni genetski algoritam koji će mijenjati vrijednosti parametara u skladu sa situacijom u kojoj se pronašao. U slučaju da postoji prevelika raznolikost unutar populacije algoritam može smanjiti vjerojatnost mutacije i

povećati vjerojatnost križanja, a ako se nađe na području lokalnog optimuma, povećanjem vjerojatnosti mutacije će se preseliti na neki drugi podskup mogućih rješenja.

3. Problem stvaranja rasporeda sati

3.1 Definicija problema

Kod stvaranja rasporeda sati radi se o problemu alociranja resursa, podložnih ograničenjima, konačnom broju vremenskih odsječaka i mjesta u svrhu zadovoljavanja skupa ciljeva u najvećoj mogućoj mjeri [6]. Problemi stvaranja rasporeda sati spadaju u NP-kompletne probleme, prema tome ne postoji deterministički algoritam koji bi u polinomijalnom vremenu dao optimalno rješenje. Svake godine u akademskim ustanovama od osnovnih škola do sveučilišta stvara se raspored sati koji je često u početnim tjednima odvijanja aktivnosti privremen. S prolaskom vremena i uočavanjem nedostataka on dolazi do svoje konačne verzije. Kvaliteta rasporeda je subjektivna i osobe odgovorne za stvaranje rasporeda će uvijek imati prigovore od strane korisnika. Raspored (naravno relativno efikasan) u potpunosti stvoren od strane računala kod korisnika će izazvati manje negodovanja nego da ga je stvarala neka osoba, jer je teže kriviti računalo (za koje vjerujemo da je objektivno) nego ljudsko biće za koje znamo da je sklono greškama. Razvoj automatiziranih načina stvaranja rasporeda plijeni pozornost već više od 40 godina. Kroz vrijeme je razvijeno više metoda rješavanja problema stvaranja rasporeda sati [5].

- Pristup pomoću matematičkog programiranja

  Zbog velikih zahtjeva za računalnim resursima nije toliko popularno.

- Pristup temeljen na bojanju grafova

  Čvorovi predstavljaju događaje, odnosno predavanje, dok bridovi označavaju konflikte između događaja, npr. profesor održava dva predavanja odjednom. Stvaranje rasporeda bez konflikata se može poistovjetiti sa bojanjem čvorova na način da dva susjedna čvora nisu obojana istom bojom. Svaka boja predstavlja jedan vremenski period. Na temelju ovako definiranog problema se grade heurističke metode. Ovaj pristup se dosta koristi zbog svoje jednostavnosti u implementaciji, ali nije toliko moćan u usporedbi sa novijim pristupima.

- Metode klastera

  Događaji se grupiraju tako da su unutar grupa zadovoljeni strogi zahtjevi. Nakon toga se grupama dodjeljuju vremenski periodi na način da se pokušaju zadovoljiti blagi zahtjevi.

- *Constraint based*

  Događaji u problemu se modeliraju skupom varijabli čije vrijednosti se dodjeljuju u skladu sa zahtjevima. Ako dodjela neke vrijednosti dovede algoritam do ilegalnog rješenja, algoritam se vraća do točke dodjele, uzima neku drugu vrijednost te nastavlja sa radom sve dok ne pronađe rješenje koje zadovoljava sve zahtjeve.

- Metaheurističke metode

  U ovu skupinu spada simulirano kaljenje, tabu pretraga, kao i genetski algoritmi. U ovom pristupu započinje se sa jednim ili više početnih rješenja nad kojima se vrši pretraga uz izbjegavanje zastoja na lokalnim optimumima. Ove metode često dovode do rješenja visoke kvalitete. Loša strana je potreba za pronalaskom parametara koji će takva rješenja dati. Osim toga imaju velike zahtjeve za računalnim resursima.

- Multikriterijski pristupi

  Svaki kriterij mjeri prekršaje pojedinog zahtjeva. Kriteriji imaju različite razine važnosti u različitim situacijama.

- Zaključivanje na temelju baze slučaja *(case-based reasoning)*

  Pristupi ove vrste koriste prošla rješenja kao građevne blokove za stvaranje rješenja za nove probleme stvaranja rasporeda. Svaki slučajem pohranjuje se problem i pripadno rješenje u bazu. Kada treba riješiti novi problem, iz baze pohranjenih rješenja se dohvaća najsličnije rješenje. Pretpostavka je da će dohvaćeno rješenje predstavljati dobru polaznu točku za rješavanje trenutnog problema. Pristup je inspiriran stvarnošću gdje se kao početni raspored u akademskim ustanovama uzima prošlogodišnji te prilagođava trenutnim zahtjevima. Važno svojstvo ovog pristupa je način određivanja sličnosti između dva rasporeda.

U ovom radu problem rasporeda sati se odnosi na srednjoškolski gimnazijski program. Raspored se sastoji od 5 dana s po sedam sati dnevno. Objekti povezani s rasporedom su profesori, razredi, predmeti i prostorije. Stvaranje rasporeda znači dodjeljivanje četvorke (profesor, razred, predmet, prostorija) određenom vremenskom razdoblju uz zadovoljavanje zahtjeva nametnutih od strane korisnika ili objekata kod stvaranja rasporeda. Profesor može predavati određen broj predmeta za koje je kvalificiran. Zahtjevi koji su nametnuti nad rasporedom sati spadaju u dvije kategorije, stroge i blage. Strogi zahtjevi moraju biti zadovoljeni te njihovim kršenjem raspored postaje neupotrebljiv. Blagi zahtjevi nisu obavezni, ali njihova zadovoljenost je poželjna. Dok strogi zahtjevi definiraju valjanost rasporeda, raspored u skladu s blagim zahtjevima osigurava zadovoljstvo korisnika, barem onoliko koliko su dobro definirani blagi zahtjevi. U primjerima problema iz stvarnog svijeta često je nemoguće zadovoljiti sve blage zahtjeve. Kvaliteta rješenja se dobiva na temelju zadovoljenosti blagih zahtjeva budući nema smisla ocjenjivati kvalitetu rješenja u kojem postoje prekršaji strogih zahtjeva, iako kod nekih problema njihova kompleksnost predstavlja zapreku zbog koje je teško pronaći rješenje koje bi uopće bilo moguće. Primjeri strogih zahtjeva su:

- Profesor ne može predavati dvama razredima odjednom
- Profesor može predavati samo predmete za koje je osposobljen
- Profesor može predavati ograničen broj sati tjedno
- Razred ne može imati dva predavanja odjednom
- U jednoj prostoriji se može istovremeno održavati samo jedno predavanje
- Slobodni sati se moraju pojavljivati samo na kraju dana
- Zahtjevi vezani uz tip prostorije

Primjeri blagih zahtjeva su:

- Predavanja istog predmeta ne smiju biti istog dana
- Predavanja istog predmeta ne smiju biti sljedeći dan
- Za predmete koji zahtijevaju veći broj sati tjedno stvarati u rasporedu blok satove
- Profesori nemaju previše rupa u svojem rasporedu

3.2 Prikaz kromosoma

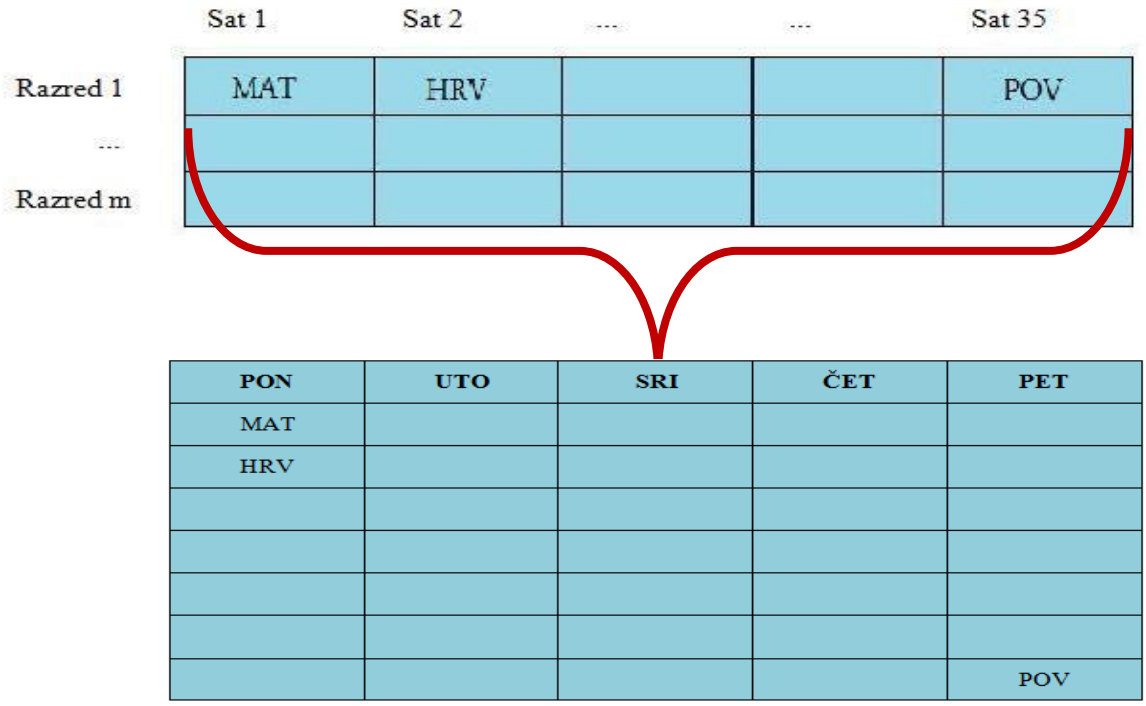
Kromosom je u ovom radu prikazan preko dvije matrice veličine *mxn* gdje je m broj razreda, a n broj raspoloživih sati u tjednu. Prva matrica sadrži profesore, te služi za raspoređivanje profesora, dok druga matrica sadrži prostorije. Matrica profesora je odabrana zbog činjenice da je strogi zahtjev kako razred ne može imati dva predavanja odjednom implicitno zadovoljen. Na isti način je odabrana i matrica prostorija jer na istom mjestu u matrici ne mogu biti dvije prostorije a time je zadovoljen još jedan strogi zahtjev. Na slici 3.1 se nalaze dva osnovna dijela kromosoma, matrica profesora i matrica prostorija.

	Sat 1	Sat 2	...	...	Sat n
Razred 1	Profesor	Profesor			Profesor
...					
Razred m					

	Sat 1	Sat 2	...	...	Sat 35
Razred 1	Prostorija	Prostorija			Prostorija
...					
Razred m					

Slika 3.1 Osnovne komponente kromosoma, matrica profesora i matrica prostorija

Fenotip na temelju kojeg se dolazi do dobrote jedinke se sastoji od ove dvije matrice i dodatne generirane matrice predmeta stvorene na temelju matrice profesora i prostorija te predmeta koje određeni profesor predaje. Na temelju te matrice tada možemo odrediti raspored za pojedini razred na način kako je prikazano na slici 3.2.



Slika 3.2 Dobivanje rasporeda za razred

3.3 Genetski operatori

3.3.1 Selekcija

Koristi se turnirska selekcija, gdje se odabire 6 nasumičnih jedinki te se dvije najbolje uzimaju kao roditelji. Razlog za odabir ovog načina umjesto *roulette-wheel* selekcije leži u velikim razlikama u dobroti između valjane i nevaljane jedinke koja može pridonijeti genetskom materijalu ali zbog svoje niske dobrote možda nikada neće biti odabrana. Kao što je rečeno, time se sprječava prerana konvergencija. Koristi se elitizam gdje najbolje 4 jedinke iz prijašnje generacije prelaze u novu.

3.3.2 Križanje

3.3.2.1 Križanje s fiksnom točkom prekida

Ovaj operator križanja uzima polovicu kromosoma iz jednog roditelja te ostatak iz drugog roditelja. Analogija je s križanjem s jednom fiksnom točkom prekida kod binarnog prikaza.

3.3.2.2 Slučajna zamjena rasporeda pojedinih razreda

Ova vrsta križanja nasumično uzima retke matrica iz jednog odnosno drugog roditelja s $p=0.5$. Inspiracija ovog križanja se nalazi u uniformnom križanju kod binarnog prikaza.

3.3.2.3 Pametno križanje

U ovom križanju do promjene dolazi kod maksimalno polovice jednog kromosoma. Odabir redaka koji će biti zamijenjeni s odgovarajućim redcima iz drugog roditelja je težinski te proporcionalan broju konflikata uvećanom za konstantu pomaka kako bi i redovi s 0 konflikata bili podložni zamjeni. Nad redcima se uzastopno izvršava jednostavna proporcionalna selekcija. Ako je redak već zamijenjen, algoritam nastavlja kao da je došlo do promjene te se zaustavlja nakon $n/2$ broja iteracija, gdje je n broj redaka u matrici kromosoma, tj. broj razreda.

3.3.3 Mutacija

3.3.3.1 Obična mutacija

U ovoj mutaciji za svaki sat kod nekog razreda algoritam s određenom vjerojatnošću zamjenjuje sat sa slučajno odabranim. Slobodni se satovi ostavljaju na mjestima gdje su se nalazili u inicijalnoj populaciji. Ovako će razredi u petak imati akumulirane slobodne satove. Ovo ograničenje se može iskoristiti na način da se u inicijalnoj populaciji postave slobodni satovi na željena mjesta. Moguće je u ovu mutaciju ugraditi logiku vezanu uz razmještaj slobodnih sati, ali pokazalo se da se ugradnjom nije dobilo na učinkovitosti. Zapravo performanse su se pogoršale, a osim toga za rezultate dostojne usporedbe sa pametnom mutacijom je bilo potrebno povećati vjerojatnost mutacije na 0.5, što ako se uzme u obzir da se radi o vjerojatnosti po genu je jednostavno previše.

3.3.3.2 Pametna mutacija

Kod pametne mutacije algoritam pokušava zamijeniti gen s slučajno odabranim, te ako bi nakon zamjene došlo do konflikata, ne zamjena se ne obavlja. Ako nakon određenog broja pokušaja nije došlo do zamjene, algoritam gleda trenutačni zbroj konflikata u stupcima koji sadrže gene i potencijalni zbroj nakon zamjene. U slučaju da je potencijalni zbroj konflikata u tim stupcima manji, dolazi do zamjene. Ako algoritam dođe do gena sa slobodnim satom, podešava poziciju zamjene na kraj dana u kojem se drugi, slučajno odabrani gen, nalazio te nalazi zadnji sat koji nije slobodan u tom danu, do zamjene će doći ako odabran slobodni sat ispred sebe ima popunjen sat jer inače bi u rasporedu nastala rupa.

3.4 Deterministično poboljšanje rješenja

Za potrebe daljnjeg poboljšanja jedinki nakon završetka genetskog algoritma stvoren je deterministički algoritam koji poboljšava jedinke. Algoritam radi nad konfliktima u matrici profesora. Algoritam bi se mogao opisati kao *hill-climber*. Započinje sa listom konfliktnih pozicija unutar matrica profesora i prostorija. Uzima prvi konflikt, te pravi susjedstvo za promatranu jedinku na temelju svih mogućih pozicija na koje se konfliktni profesor može smjestiti za razred kod kojeg je nastao konflikt. Iz susjedstva uzima najbolju jedinku, te nad njom ponavlja postupak ili dok konflikti u matrici prekršaja nestanu ili ne može naći bolju jedinku. Iako radi i nad konfliktima prostorija pokazalo se u situaciji sa 0 prekršaja u matrici profesora nije sposoban pretjerano smanjiti konflikte kod prostorija, ako je testni slučaj teži što se tiče raspoloživog prostora.

3.5 Evaluacija dobrote

Dobrota se računa na temelju tri svojstva jedinke:

- Broju konflikata u matrici profesora, K_{prof}
- Broju konflikata u matrici prostorija, K_{pros}
- Težinskoj sumi prekršenih blagih zahtjeva, K_{blag}

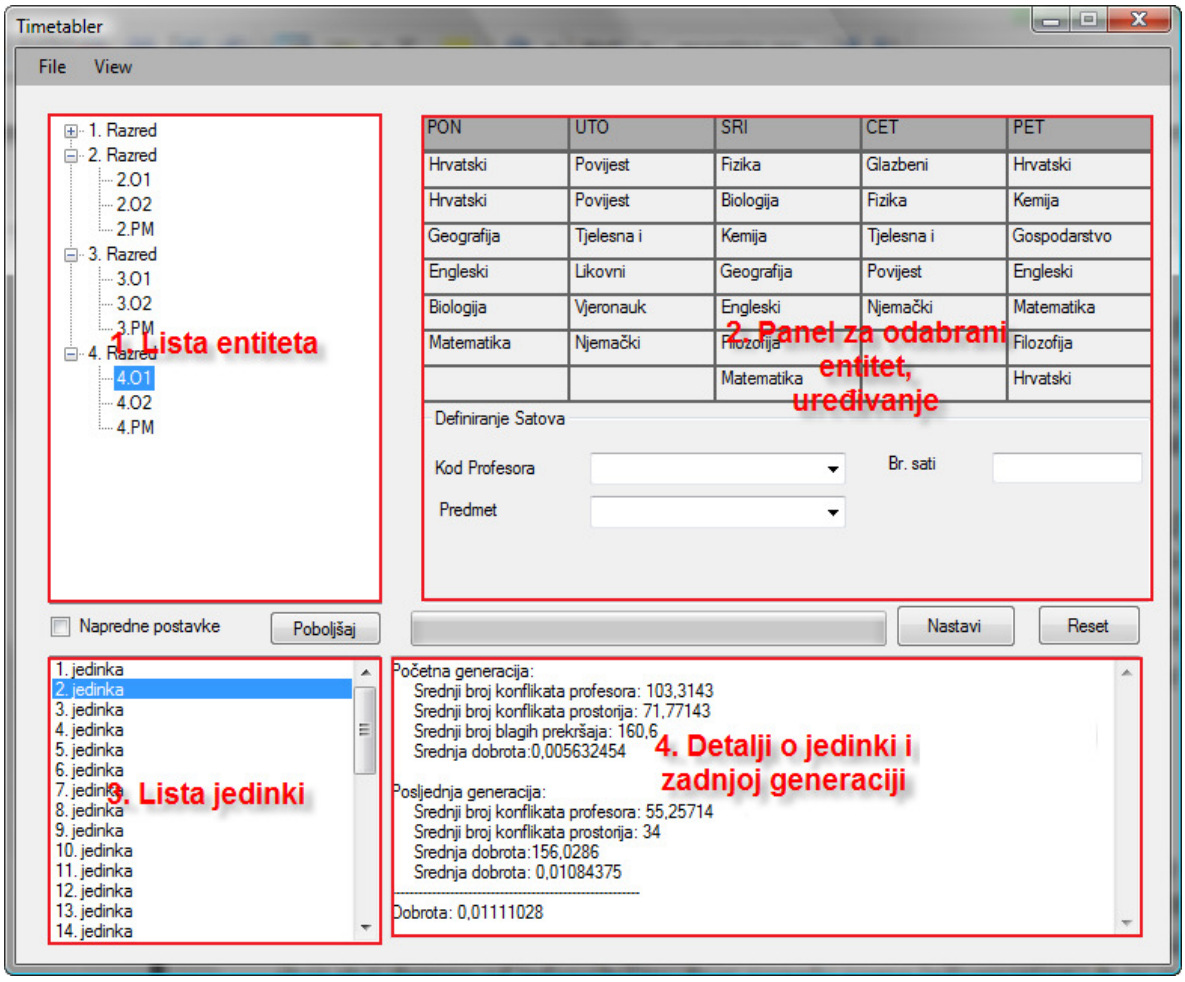
Prvo se evaluira kvaliteta rasporeda za svaki razred, kvaliteta se određuje na temelju prekršenih blagih zahtjeva. Svaki prekršeni blagi zahtjev ima određenu težinu u sumi za taj razred. Nakon što dobijemo sumu težinski zbrojenih blagih prekršaja računa se broj prekršaja strogih zahtjeva vezane uz pojavu profesora u isto vrijeme kod dva razreda. Na kraju se još računa broj prekršaja u matrici prostorija. Konačno iz tih triju komponenti se na temelju jednadžbe (3.1) dobiva dobrota.

$$f = \frac{1}{1 + K_{prof} + K_{pros} + \frac{K_{blag}}{DBZ}} \tag{3.1}$$

DBZ označava djelitelja kojim podešavamo utjecaj blagih zahtjeva na dobrotu. Ovaj parametar, osim što statički djeluje na algoritam, bi se mogao iskoristiti za dinamičko vođenje algoritma prema kvalitetnijim rješenjima. Na početku, kada imamo veliki broj prekršaja strogih zahtjeva, može se staviti veći djelitelj, a kasnije kada udio rješenja sa zadovoljenim strogim zahtjevima poraste na određeni postotak, možemo spustiti djelitelj i povećati pritisak na blage zahtjeve.

4. Programsko rješenje

Grafičko sučelje koje upravlja genetskim algoritmom je napravljeno u svrhu olakšanja unošenja ulaznih podataka i lakšeg pregledavanja rezultata. Glavna zaslonska maska se nalazi na slici 4.1.

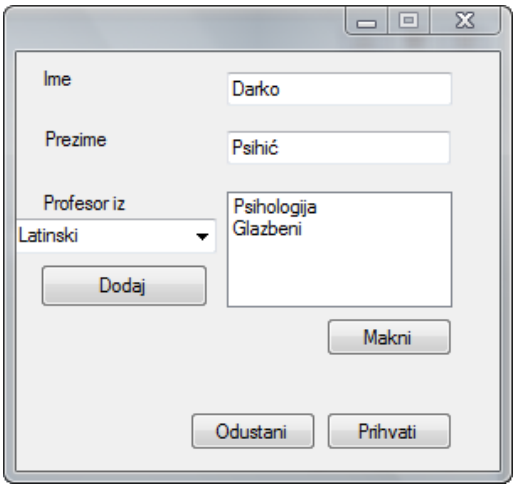


Slika 4.1 Glavna zaslonska maska

Glavni elementi grafičkog sučelja su:

1. Lista entiteta

   Prikazuje se lista entiteta zavisno o odabranom pogledu, moguće je pregledavati prema profesorima, razredima, prostorijama, predmeti, za sve entitete osim predmeta se prikazuje raspored. Novi entitet se dodaje preko kontekstnog menija klikom na Add, klikom na Remove odabrani entitet se briše. Kod dodavanja razreda oznaka razreda prvi znak mora označavati godinu kojoj razred pripada. Za profesore i razrede postoji i dodatna opcija Edit, kod razreda služi samo za mijenjanje imena, a za profesore je to jedini način uređivanja. Zaslonska maska za uređivanje profesora se nalazi na slici 4.2.



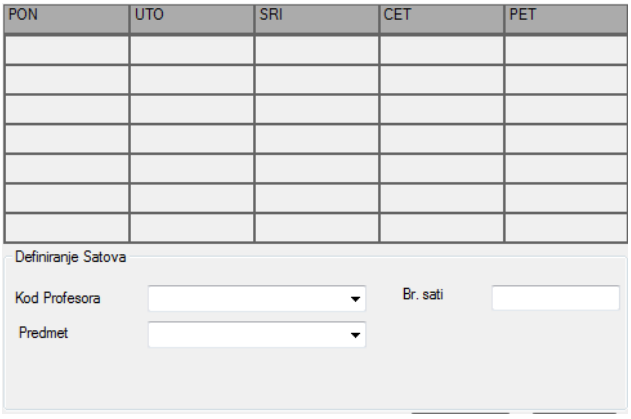
Slika 4.2 Zaslonska maska za uređivanje profesora

2. Panel za odabrani entitet

   Ovisno o odabranom pregledu, prikazuju se detalji za entitet, ovdje se vrši većina uređivanja ulaznih podataka.

Panel poprima sljedeće izglede:

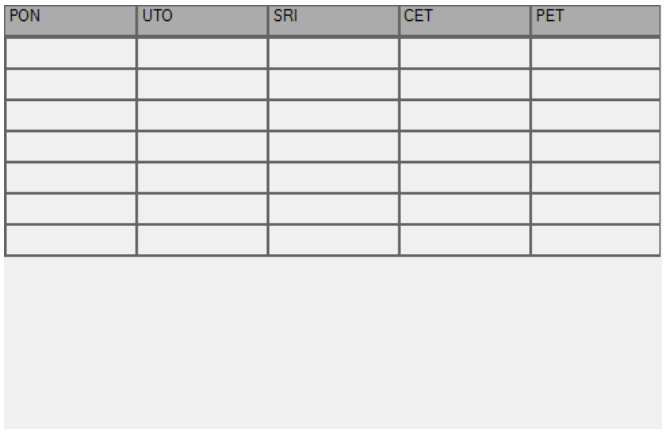
- Za razrede



Slika 4.3 Izgled panela za pregled razreda

Definiranje satova je jednostavno, preko padajućih izbornika na slici 4.3 se odabire profesor, predmet između onih koje taj profesor predaje, a zatim se upiše broj sati tjedno koje taj profesor održava trenutno odabranom razredu.

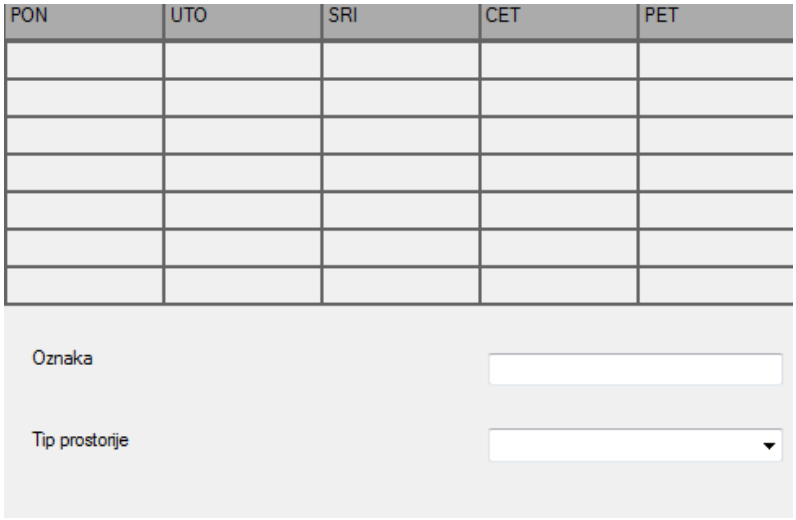
- Za profesore



Slika 4.4 Izgled panela za pregled profesora

Kod profesora ne postoji mogućnost direktnog uređivanja podataka na glavnom zaslonu, već se do te funkcionalnosti dolazi na ranije opisan način. Prikazuje se samo raspored, kao što je vidljivo na slici 4.4.

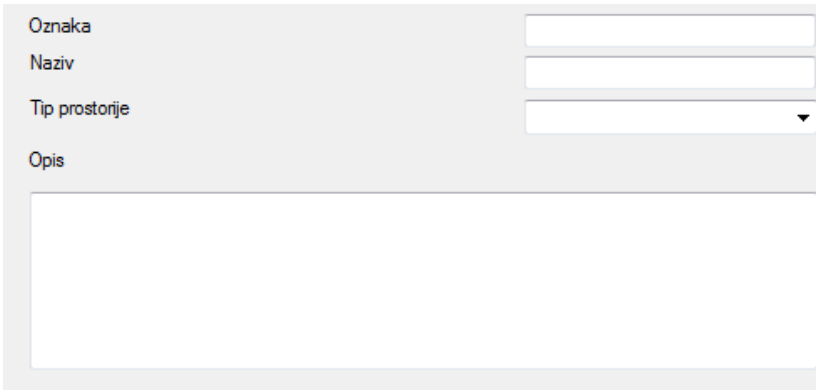
- Za prostorije



Slika 4.5 Izgled panela za pregled prostorija

Za prostorije se definira oznaka i tip prostorije, kao i kod profesora i razreda, može se pregledavati raspored. Izgled panela je prikazan na slici 4.5.

- Za predmete



Slika 4.6 Izgled panela za pregled predmeta

Za predmete se evidentira njihova oznaka koja će se koristiti za prikaz rasporeda, puni naziv tog predmeta, tip prostorije koju zahtjeva taj predmet te opcionalno opis. Na slici 4.6 je prikazan izgled panela.

3. Lista jedinki

   Služi za odabir jedinke iz koje se prikazuju podaci za ostatak zaslona

4. Detalji o jedinki i zadnjoj generaciji

   Prikazuju se detalji o inicijalnoj i posljednjoj generaciji: srednji broj konflikata u matrici profesora, srednji broj konflikata u matrici prostorija, srednji broj prekršenih blagih zahtjeva, kao i srednja dobrota. Prikazuje se i detalji o jedinki: njezina dobrota, broj prekršaja u strogih zahtjeva za profesore, broj prekršaja blagih zahtjeva, broj prekršaja strogih zahtjeva za prostorije, za svakog profesora i prostoriju, ukoliko za njih postoje konflikti ispisuje koliko ih je za tog profesora odnosno prostoriju.

4.1 Podešavanje genetskog algoritma

Podešavanje se vrši klikom na napredne postavke. Parametri koje možemo mijenjati su:

- Veličina populacije
- Broj generacija
- Vrsta mutacije
- Vrsta križanja
- Vjerojatnost mutacije
- Vjerojatnost križanja
- Elitizam

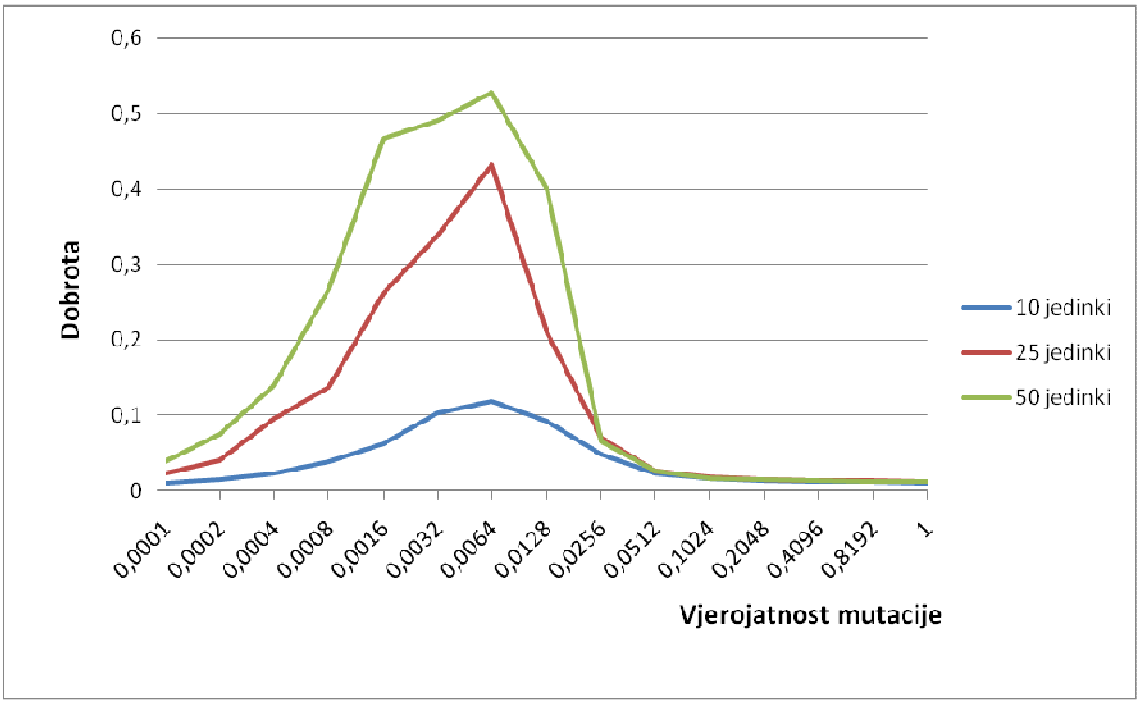
4.2 Ostale mogućnosti

Program podržava pohranjivanje rezultata zajedno s ulaznim podacima nad kojima onda kasnije možemo nastaviti izvođenje genetskog algoritma ili započeti novo izvođenje. Treba napomenuti da program ne pamti na kojoj je generaciji stao tako da će kod nastavljanja proći kroz onaj broj koji je definiran kao broj generacija. Osim što se dobiveni rezultati mogu pregledavati i spremati, program podržava ispis rasporeda za trenutno odabrani entitet trenutno odabrane jedinke ili ispis za sve entitete trenutno odabranog pregleda za trenutnu jedinku. Isto tako možemo napraviti izvoz podataka za trenutnu jedinku u CSV ili HTML.

5. Rezultati

Skup podataka nad kojim se izvršavalo testiranje je potekao iz zamišljene gimnazije. Sastoji se od 13 razreda, 23 profesora i 15 prostorija od kojih je sedam regularnih, a ostalih osam su za potrebe predmeta koji imaju posebne zahtjeve kod izvođenja nastave, npr. prostorija za geografiju. Razredi imaju od 31 do 33 sata tjedno nastave od mogućih 35. Npr. razred sa oznakom 1.PM ima 33 sata tjedno, od toga 15 sati nastave predmeta sa posebnim zahtjevima nad prostorijama. Odabrani broj prostorija je minimalan, tj. nastava se oduzimanjem jedne regularne prostorije više ne bi mogla održavati, odnosno genetski algoritam nikad ne bi došao do rješenja. Minimalan je i odabrani skup profesora. Ako se fiksira broj profesora i prostorija moguće je dodati još jedan razred, te će genetski algoritam doći do rješenja, ali za potrebe testiranja i pokazivanja utjecaja parametara na izvođenje odabran je slučaj sa 13 razreda. Testiranje nad nekim skupom parametara se ponavljalo 5 puta. Većina grafova se odnosi na prosjek najboljih jedinki, osim ako nije posebno napomenuto.

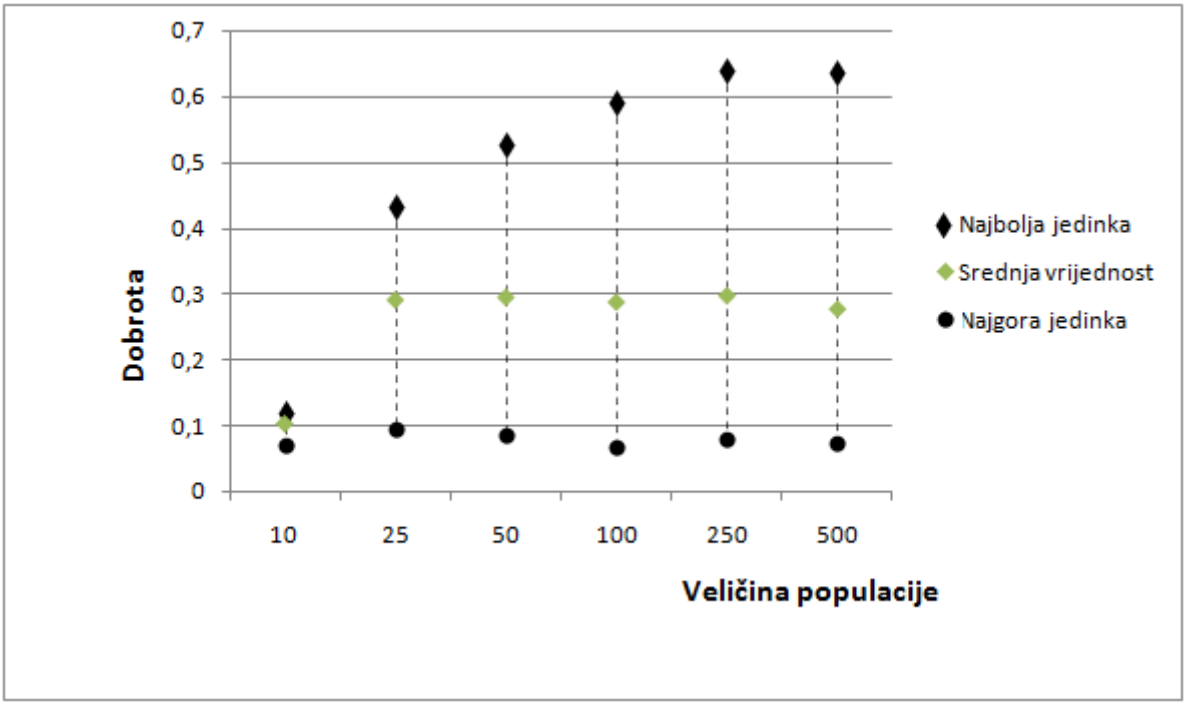
5.1 Utjecaj vjerojatnosti mutacije



Slika 5.1 Utjecaj vjerojatnosti mutacije na dobrotu

Slika 5.1 prikazuje utjecaj mutacije na rad genetskog algoritma za veličinu populacije od 10, 25, 50 jedinki. Vjerojatnost mutacije je povećavana eksponencionalno do vrijednosti 1 kako bi se pokazao destruktivni učinak prevelike mutacije na rad algoritma. Rezultati se odnose na pametnu mutaciju u kombinaciji sa križanjem slučajnom zamjenom redaka za 700 iteracija. Vidljivo je da se mutacija ponaša u skladu sa očekivanjima. Slabu kvaliteta jedinki za niske vjerojatnosti mutacije pripisujemo tome što je algoritam zastao na nekom lokalnom optimumu. Za veće vrijednosti mutacije algoritam za svaku generaciju radi nad jedinkama koje su previše rasprostranjene po prostoru pretrage da bi konvergirao. Čak i da se križanjem dobije jedinka dobre kvalitete, efektivna vjerojatnost mutacije nad nekim genom je dvostruko veća od samog parametra mutacije budući mutacija radi nad genima njihovom zamjenom. Naravno, ta procjena važi za slučaj kad se ne promatra utjecaj slobodnih satova na efektivnu mutaciju, budući oni mogu doći samo na određena mjesta (na kraj dana prije ostalih slobodnih satova). Iz grafa je isto tako vidljivo da porastom veličine populacije raste dobrota najbolje jedinke.

5.2 Utjecaj veličine populacije



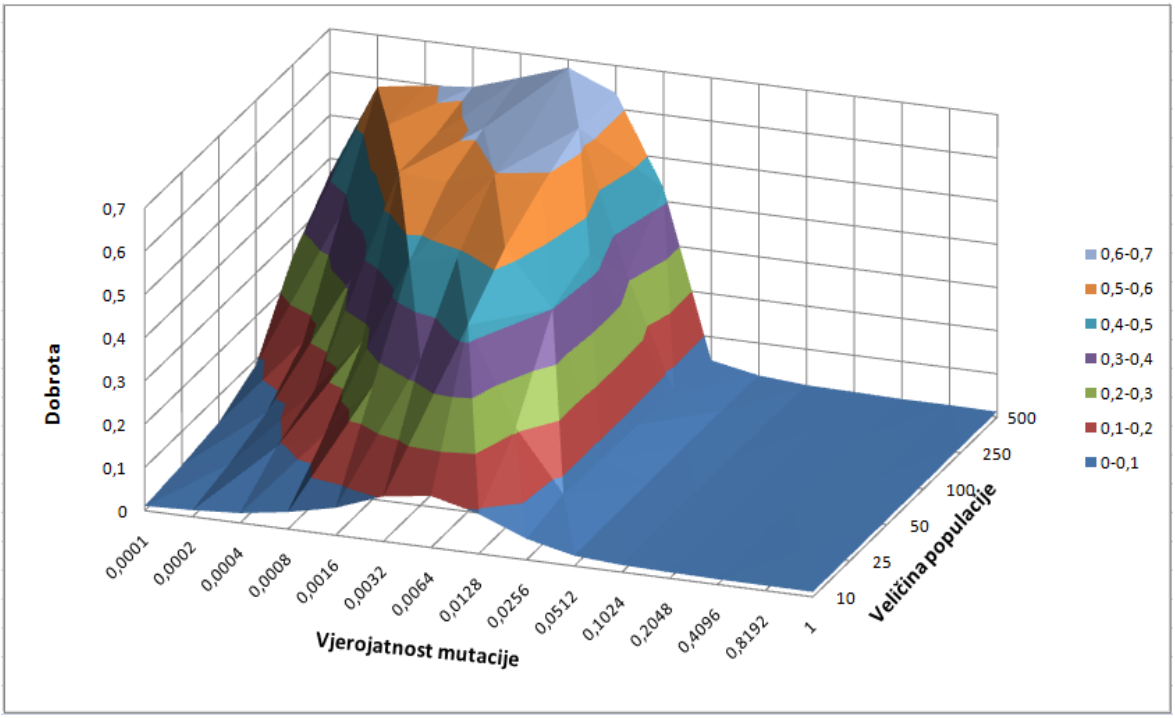
Slika 5.2 Utjecaj veličine populacije

Iz slike 5.2 je vidljivo da porastom broja jedinki unutar populacije genetski algoritam za isti broj iteracija dolazi do kvalitetnijih rješenja. No promatrajući srednju vrijednost unutar pojedinog slučaja za veličinu populacije vidimo da i nema neke razlike osim kod slučaja sa 10 jedinki, što se je pokazalo kao premalen broj s kojim bi algoritam mogao efikasno raditi. Iako se čini primamljivim povećati veličinu populacije vidljivo je da taj rast u kvaliteti nije konstantan te će doći do gornje granice kao što se ovdje vidi za slučaj od 500 jedinki. Ne treba zaboraviti da povećanje veličine populacije za rezultat ima i odgovarajuće povećanje vremena izvođenja genetskog algoritma, što je vidljivo iz tablice 5.1.

Tablica 5.1 Vrijeme izvođenja uz različite veličine populacije za 700 generacija

Veličina populacije	10	25	50	100	250	500
Vrijeme	15.20 s	53.41 s	119.04 s	250.43 s	660.11 s	1366.54 s

Isti učinak se sigurno može postići i podešavanjem parametara i puštanjem algoritma kroz veći broj generacija u manjem vremenskom roku. No za slučaj da se algoritam odluči paralelizirati, postoje očigledne prednosti veće populacije. Na slici 5.3 se nalazi 3d graf ovisnosti dobrote o veličini populacije i vjerojatnosti mutacije.

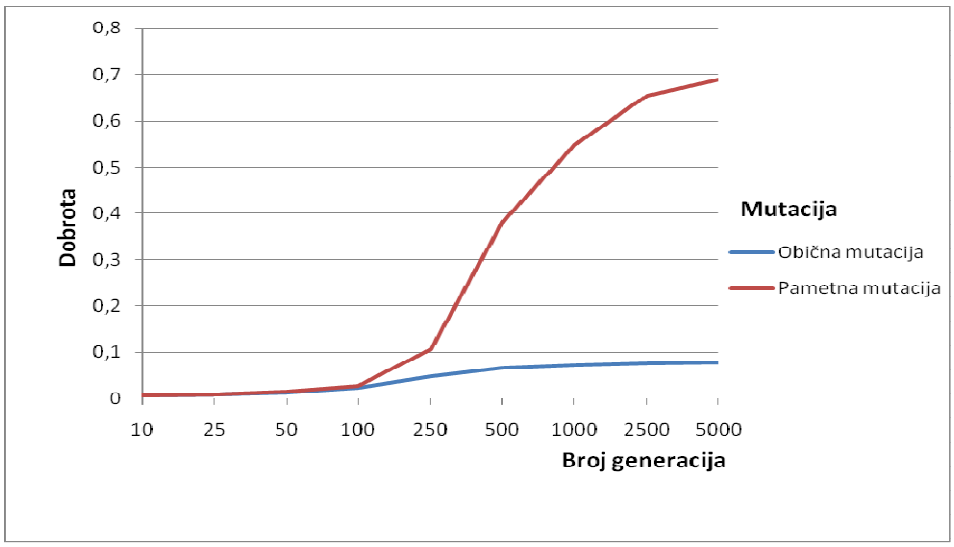


Slika 5.3 3D prikaz utjecaja vjerojatnosti mutacije i veličine populacije na dobrotu

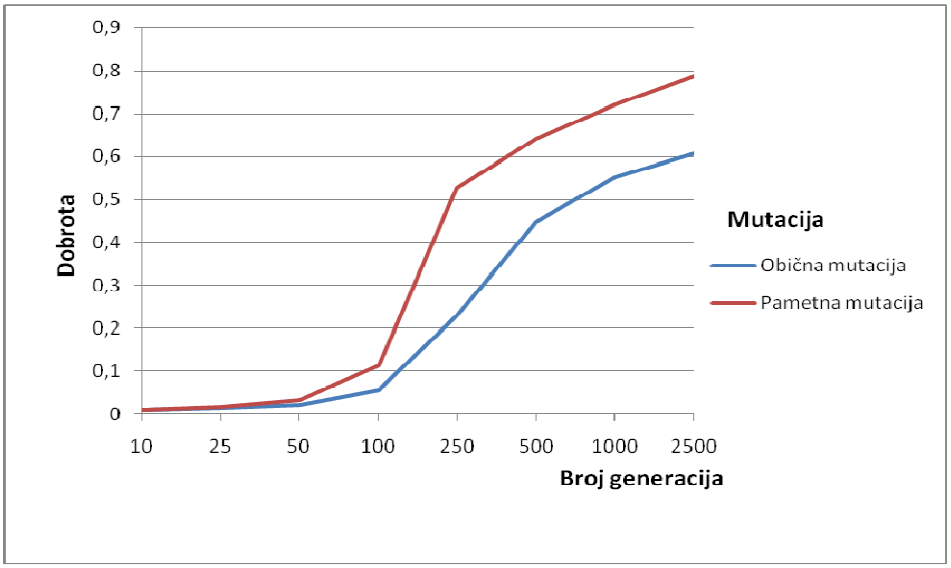
5.3 Utjecaj vrste mutacije

Odabrana vrsta mutacije ima ogromni utjecaj na ponašanje genetskog algoritma. Obična mutacija nije dovoljna, barem za ovaj testni slučaj, zadovoljiti stroge zahtjeve. Na slici 5.4 vidimo razliku u upotrebi pojedine vrste mutacije. Vjerojatnost mutacije je postavljena na 0.003, a veličina populacije na 40 jedinki. Odabrano je križanje nasumičnom zamjenom redaka. Ako se promotri broj konflikata za posljednju generaciju vidimo da kod regularne mutacije algoritam ima probleme sa stvaranjem rasporeda prostorija, što nije čudno ako se uzme u obzir da je u testnom slučaju upotrijebljen minimalni broj prostorija. Od petstote pa do 5000. generacije se srednja vrijednost broja konflikata kod prostorija za najbolje jedinke nalazila oko 11. Kod rasporeda profesora se obična mutacija ipak snalazi malo bolje jer u satnici profesora postoji dovoljno prostora za definiranje još jednog razreda, pa je i manja

vjerojatnost da će doći do konflikta. Osim što ne dolazi do mogućeg rješenja, algoritam za ovaj testni slučaj nije sposoban nadmašiti upotrebu pametne mutacije niti kod broja prekršenih blagih zahtjeva. Srednja vrijednost najboljih jedinki u 5 prolaza algoritma iznosi 80.2, dok je upotrebom pametne mutacije algoritam došao do srednjeg broja od 58.2 prekršaja blagih zahtjeva. Osim navedenih ograničenja postoji još jedan razlog loših rezultata kod obične mutacije. Kao što je rečeno obična mutacija zanemaruje slobodne satove, odnosno ostavlja ih na mjestima gdje su bili u inicijalnoj populaciji.



Slika 5.4 Utjecaj vrste mutacije



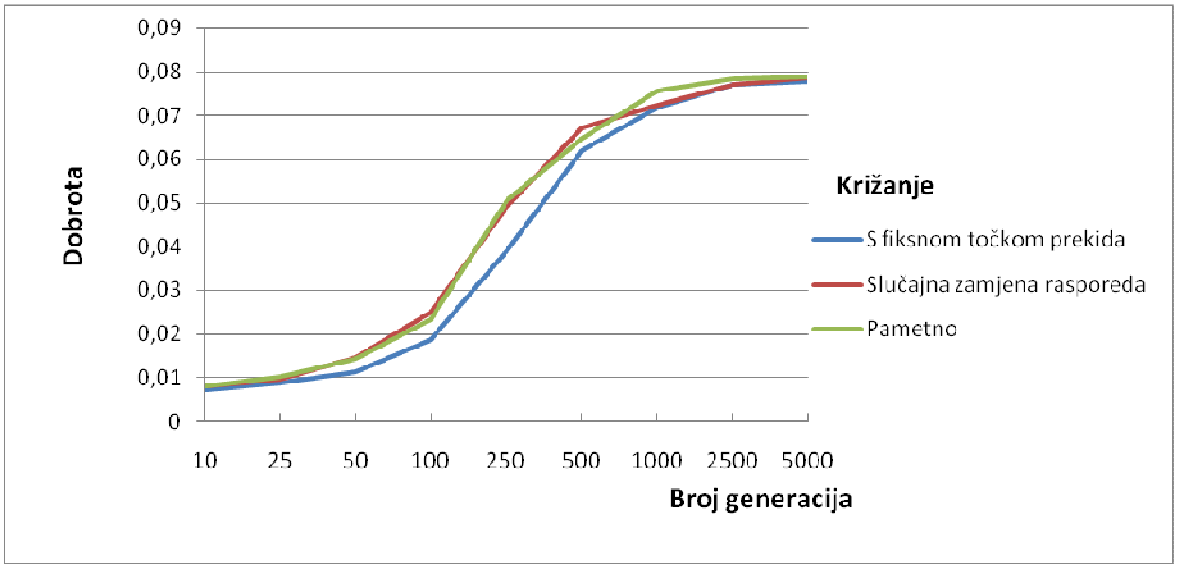
Slika 5.5 Utjecaj vrste mutacije za lakši problem

Za testni slučaj gdje je jedan razred manje te jedna prostorija više obična mutacija je konkurentna pametnoj. Na slici 5.5 je vidljivo ponašanje algoritma nad spomenutim testnim slučajem. U 250toj generaciji pametna mutacija dolazi do mogućeg rješenja, običnoj treba nešto dulje, ali do 500 generacije obje mutacije dovode genetski algoritam do rješenja koje zadovoljava stroge zahtjeve. U svim kasnijim generacijama postoje moguća rješenja, no pametna mutacija ih stvara više te bez obzira što ona gleda samo stroge zahtjeve, algoritam će prije doći do rješenja sa manjim brojem prekršaja blagih zahtjeva nego u slučaju da koristi regularnu mutaciju. Na kraju 2500. generacije korištenjem pametne mutacije se dolazi do prosječnog broja prekršaja blagih zahtjeva od 41.4 za najbolju jedinku, dok kod korištenja obične mutacije taj broj iznosi 78. Ako se gleda po razredima, pametna mutacija u prosjeku u ovom testnom slučaju za jedan razred daje 3.05 manje prekršaja od obične.

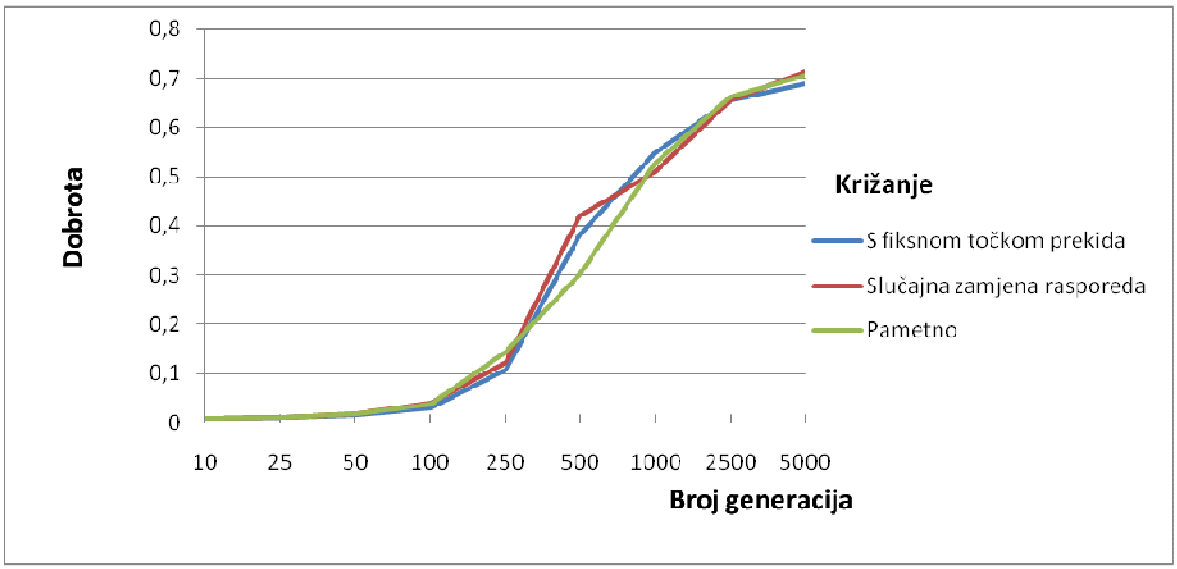
5.4 Utjecaj vjerojatnosti i vrste križanja

Utjecaj vrste križanja je obavljen za obje mutacije uz vjerojatnost mutacije 0.003, a vjerojatnost križanja 0.8.

Rezultati se nalaze na slikama 5.6 i 5.7.

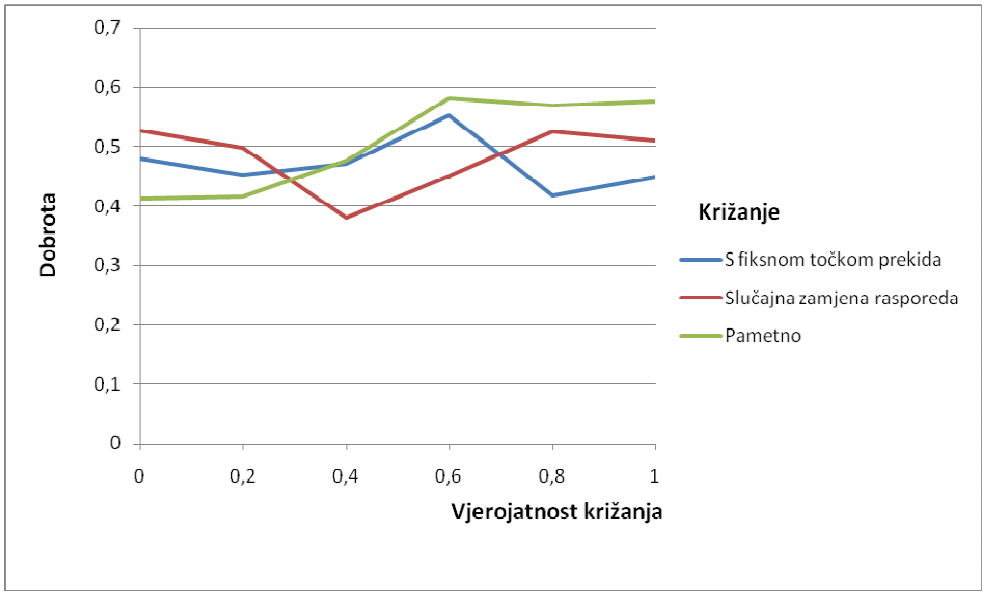


Slika 5.6 Ponašanje algoritma za različita križanja uz običnu mutaciju

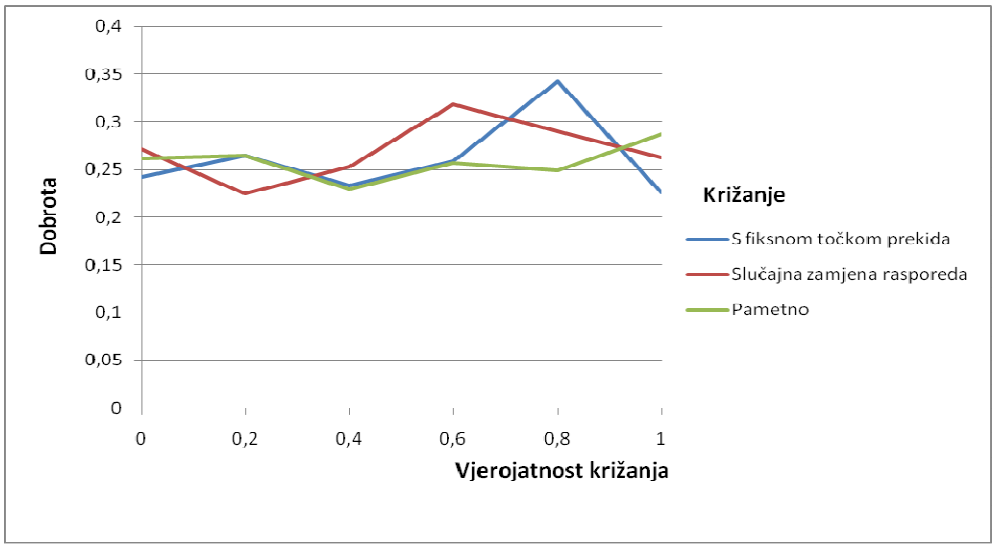


Slika 5.7 Ponašanje algoritma za različita križanja uz pametnu mutaciju

Promatrajući slike vidimo da križanje nema veliki utjecaj na performanse genetskog algoritma. Iznenađujući rezultati, pa su testiranja provedena još jednom, ali do promjene nije došlo. Osim što se čini da vrsta križanja nema veliki utjecaj na dobrotu, slike 5.8 i 5.9 pokazuju da je takav slučaj i sa vjerojatnošću sa kojom se križanje primjenjuje. Iako izgleda da postoje određene razlike istina je drugačija, varijacije u kvaliteti rješenja su puno manje nego što to slika daje naslutiti, te su one posljedica načina računanja dobrote.



Slika 5.8 Ponašanje algoritma za promjenu vjerojatnosti križanja uz vjerojatnost mutacije 0.005



Slika 5.9 Ponašanje algoritma za promjenu vjerojatnosti križanja uz vjerojatnost mutacije 0.002

Ovakvi rezultati, iako iznenađujući, pojavljivali su se i kod drugih radova. U radu [4] koji koristi isti ideju prikaza kromosoma (ne uzimaju u obzir prostrije) križanje se uopće ne koristi jer dolazi do slučaja da se kvaliteta rješenja srozava sa povećanjem vjerojatnosti mutacije. Njihovo objašnjenje je kako križanje radi prevelike korake u prostoru pretrage kao rezultat epistaze (utjecaj promjene u genima na prinos dobrote drugog gena) postignute prikazom.

6. Zaključak

U ovom radu je prikazana primjena genetskih algoritama na područje stvaranja rasporeda sati. Genetski algoritmi kao metoda optimiranja su prokušan način za rješavanje mnogih problema, a u zadnjih nekoliko godina porastom snage računala postaju sve popularniji. No puno toga još nije razjašnjeno o genetskim algoritmima. Zašto uopće rade? Intuitivno možemo doći do nekih zaključaka, ali teorija vezana uz genetske algoritme nije razvijena do te mjere da se može dati formalan odgovor na to pitanje. Istraživanje rada genetskog algoritma, osim što stvara teoretsku pozadinu, je važno jer nam može dati i načine predviđanja performansi za neko područje problema. U trenutku odluke o upotrebi genetskog algoritma ne smije se zaboraviti činjenica da genetski algoritam ne dolazi s jamstvom pronalaska najboljeg rješenja. U većini slučajeva nam ono nije niti potrebno; potrebno je rješenje koje je dovoljno kvalitetno za upotrebu. Takvo razmišljanje vrijedi i za stvaranje rasporeda sati. Pokazano je da genetski algoritam dolazi do rješenja bez prekršaja strogih zahtjeva (što je uvjet da se raspored može koristiti), a kasnije prolaskom kroz sve veći broj generacija dolazi do kvalitetnijih rješenja sa stajališta blagih zahtjeva. Rezultati jasno pokazuju da bez dodatne prilagodbe operatora nije moguće dobivati kvalitetna rješenja. Obična mutacija je dovoljna za jednostavnije slučajeve, ali smanjivanjem udjela mogućih rješenja u prostoru otežavanjem problema pretrage ona dolazi do svojih granica. Izostanak većeg učinka izbora operatora križanja te njegove vjerojatnosti, kako na najbolju jedinku generacije, tako i srednju vrijednost dobrote iznenađuje, ali kao što je rečeno ovo nije jedini rad u kojem se došlo do takvih zaključaka.

7. Literatura

[1] Mitchell, M. *An Introduction to Genetic Algorithms*, peto izdanje. Cambridge, Massachusetts: The MIT press, 1999.

[2] Reeves, Colin R., Rowe, Jonathan E. *Genetic Algorithms-Principles and Perspectives-A Guide to GA Theory.* Dordrecht: Kluwer Academic Publishing, 2003

[3] Gen, M., Cheng, R.. *Genetic Algorithms and Engineering Optimization*. Kanada: John Wiley & Sons, Inc, 2000

[4] Beligiannis, Grigorios N., Moschopoulos, Charalampos N., Kaperonis, Georgios P., Likothanassis, Spiridon D. *Applying evolutionary computation to the school timetabling problem: The Greek case*. Computers & Operations Research, Volume 35, Issue 4, (2008), str. 1265-1280

[5] Petrović, Sanja, Burke, Edmund.*The Handbook of Scheduling:Algorithms, Models, and Performance Analysis: University Timetabling,* CRC Press, 2004

[6] Golub, M. *Genetski algoritmi-Prvi dio*. Fakultet elektrotehnike i računarstva. Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave, 2004

[7] Wren, A. *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling : Scheduling, Timetabling and Rostering - A Special Relationship?*, Lecture Notes In Computer Science, Edinburgh, (1995), str. 46 – 75

[8] De Jong, K. A. *Foundations of Genetic Algorithms 2:Genetic Algorithms are NOT Function Optimizers,* San Mateo Ca, USA: Morgan Kaufmann Publishers, Inc., 1993

Stvaranje rasporeda sati genetskim algoritmima

Sažetak

U ovom radu je dan kratki uvod u rad genetskih algoritama. Zatim su opisani problemi stvaranja rasporeda sati i kratki opis različitih načina rješavanja. Ostvaren je i opisan genetski algoritam stvoren za potrebe rješavanja tog problema sa dvije mutacije i tri križanja. Stvoreno je grafičko sučelje za lakši unos podataka i prikaz rješenja. Prikazani su rezultati ostvareni programskim ostvarenjem. Proučen je utjecaj različitih operatora i vjerojatnosti njihove primjene.

Ključne riječi

Genetski algoritmi, stvaranje školskog rasporeda sati, kombinatorička optimizacija

Solving Timetabling Problems using Genetic Algorithms

Abstract

In this document a short introduction in genetic algorithms is given. Timetabling problems are described and a list of different approaches of finding a solution are presented. A genetic algorithm for solving school timetabling problems is implemented and described with two mutation, and three crossover operators. For the purpose of easier data input and result presentation, a graphical user interface was created. Usage of different operators and the influence of probabilities applied to them are analyzed.

Keywords

Genetic algorithms, school timetabling problem, combinatorial optimization