SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Razvojno okruženje za genetske algoritme

Luka Humski

Voditelj: *doc. dr. sc. Marin Golub*

Zagreb, travanj, 2008.

Sadržaj

1. Uvod....4

2. Evolucija u prirodi....5

3. Genetski algoritmi....7

3.1 Populacija....7

3.2 Funkcija cilja ili dobrote....7

3.3 Odabir ili selekcija....7

3.4 Elitizam....8

4. Ostvarenje genetskih algoritama u Matlabu....9

4.1 Pozivanje funkcija za genetske algoritme iz linije naredbi (engl. command line) 9

4.1.1 Pokretanje genetskih algoritama s pretpostavljenim postavkama....9

4.1.2 Postavljanje mogućnosti za genetske algoritme....9

4.1.3 Pojašnjenja postavki....13

4.1.3.1 Postavke crtanja....13

4.1.3.2 Postavke populacije....14

4.1.3.3 Postavke selekcije....15

4.1.3.4 Postavke reprodukcije....16

4.1.3.5 Postavke mutacije....17

4.1.3.6 Postavke križanja....18

4.1.3.7 Postavke migracije....20

4.1.3.8 Postavke za zaustavljanje genetskog algoritma....21

4.2 Korištenje Optimization toola....21

4.3 Primjeri....23

4.3.1 Primjer 1., ...23

4.3.2 Primjer 2., ...25

4.3.3 Primjer 3., ...26

4.3.4 Primjer 4., Rastriginova funkcija....27

5. Zaključak....29

6. Literatura.....30

7. Sažetak.....31

1. Uvod

Genetski algoritmi, kao jedni od evolucijskih algoritama, heuristička su metoda optimiranja koja imitira prirodni evolucijski proces. Predstavljaju model strojnog učenja čije ponašanje potječe iz procesa evolucije koji se neprekidno odvija u prirodi.

Ideju evolucijskog računarstva 1960. godine u svom djelu „Evolucijske strategije“ donosi I. Rechenberg. Ideja je prihvaćena od strane istraživača na području računalne znanosti te se počinje intenzivno proučavati. Genetski su algoritmi rezultat istraživanja Johna Hollanda. Kao algoritmi koji se oslanjaju na ideje iz prirode, pogodni su za pretraživanje velikog prostora mogućnosti s ciljem pronalaska optimalnog rješenja.

Osnovna karakteristika evolucije u prirodi je prilagođavanje živih bića uvjetima u prirodi. Analogija evolucije kao prirodnog procesa i genetskog algoritma kao metode optimiranja očituje se u procesu **selekcije** i **genetskim operatorima**. Mehanizam odabira u prirodnom evolucijskom procesu čine okolina i uvjeti u prirodi. Kod genetskih algoritam ključ selekcije je **funkcija cilja** ili **funkcija dobrote** koja na odgovarajući način predstavlja problem koji se rješava. U prirodi jedinka koja je najbolje prilagođena uvjetima i okolini ima najveću vjerojatnost preživljavanja i parenja pa tako i prenošenja vlastitog genetskog materijala. Kod genetskih algoritama jednu jedinku predstavlja jedno rješenje. Selekcijom se odabiru dobre jedinke (rješenja) koje se prenose u sljedeću populaciju, a manipulacijom genetskog materijala stvaraju se nove jedinke. Takav ciklus **reprodukcije**, **selekcije** i **manipulacije** ponavlja se sve dok nije zadovoljen uvjet zaustavljanja genetskog algoritma.

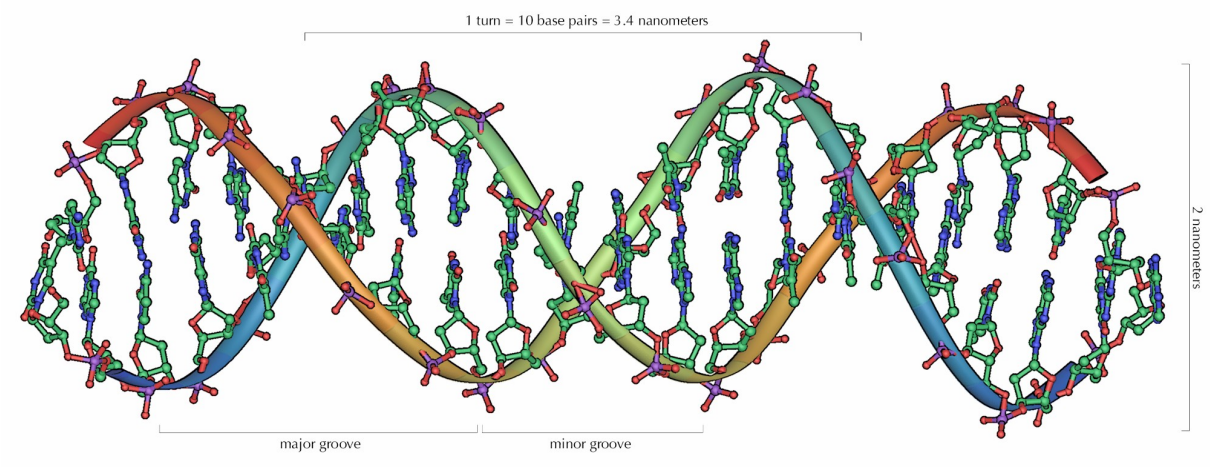
Po načinu djelovanja, genetski algoritmi ubrajaju se u **metode slučajnog pretraživanja prostora rješenja** u potrazi za globalnim optimumom. U tu skupinu možemo ubrojiti još i evolucijske strategije, simulirano kaljenje te genetsko programiranje. Snaga tih metoda, a ponajviše genetskih algoritama, leži u tome što su u stanju odrediti položaj globalnog optimuma u prostoru s više lokalnih ekstrema, u tzv. višedimenzionalnom prostoru. Klasične deterministčke metode tražit će lokalni ekstrem za koji ne znamo je li ujedno i globalni. Stohastičke metode, kojima pripadaju i genetski algoritmi, mogu s nekom vjerojatnošću odrediti globalni ekstrem funkcije cilja. Temeljna razlika između ova dva pristupa jest u tome da determinističke metode daju siguran rezultat, ali manje značajan (lokalni ekstrem), a uz rezultate dobivene stohastičkim metodama uvijek stoji postotak koji označava vjerojatnost da je to rješenje uistinu ono za koje ga se smatra, no, osim lokalnih, mogu dati i globalne optimume. Dakako, vjerojatnost ispravnosti rješenja kod stohastičkih metoda povećava se s brojem ponavljanja procesa rješavanja.

2. Evolucija u prirodi

Evolucija je neprekidan proces prilagođavanja živih bića na svoju okolinu, tj. na uvjete u kojima žive. Kako bi opstala, svaka nova generacija mora naslijediti dobra svojstva prethodne generacije, pronalaziti ih i mijenjati tako da ostanu dobra u neprekidno novim uvjetima. Jedinke koje naslijede loša svojstva bit će inferiorne u odnosu na jedinke s dobrim svojstvima te je velika vjerojatnost njihova izumiranja. S njihovim izumiranjem, nestaju i njihova (loša) svojstva.

Sva svojstva jedinke zapisana su u **kromosomima**. Kromosomi su lančaste tvorevine koje se nalaze u jezgri stanice. Skup svih informacija koje opisuju jedno svojstvo nalazi se u djeliću kromosoma koji nazivamo **gen**. Kromosomi uvijek dolaze u parovima (jedan od majke, a drugi od oca). Dakle, za svako svojstvo postoje dva gena. U genetskom paru geni mogu biti ravnopravni i neravnopravni, tj. jedan može biti dominantan, a drugi recesivan. U ravnopravnom paru gena novonastala jedinka svojstvo definirano tim parom gena ima negdje između svojstava oca i majke, dok u slučaju neravnopravnih gena u paru, rezultantno svojstvo određuje dominantni gen.

Kemijsku strukturu prisutnu u kromosomima otkrili su 1953. godine Watson i Crick. Molekula DNK (Slika 2.1) je u obliku dvije spirale građene od fosforne kiseline i šećera, a mostovi između spiralnih niti građeni su od **dušićnih baza** koje mogu biti: adenin (**A**), gvanin (**G**), citozin (**C**) i timin (**T**). Komplementarni parovi, tj. parovi preko kojih su dvije spirale DNK vodikovim vezama međusobno povezane, su adenin i timin te citozin i gvanin. Informacija je zapisana upravo u tim dušičnim bazama. Kao što je u računalu najmanja jedinica informacije jedan bit, tj. jedna binarna znamenaka, tako je u prirodi najmanja jedinica informacije jedna dušična baza.



Slika 2.1 Molekula DNK

DNK informaciju kodira kroz niz dušičnih baza. Informacija sadrži šifru aminokiseline koja je potrebna za dobavljanje nekog proteina. Kako postoji 20 različitih aminokiselina, a jedan nukleotid može adresirati samo 4 različite aminokiseline, jasno je da će informacija morati biti zapisana u nekoliko nukleotida. S najmanje 3 nukleotida možemo adresirati barem 20 (43=64>20) različitih aminokiselina. Stoga se u prirodi i koriste tripleti nukleotida koje nazivamo **kodonom**.

Genetski se materijal u prirodi s generacije na generaciju prenosi križanjem. Prilikom križanja neki geni mutiraju, tj. postanu neki drugi gen.

3. Genetski algoritmi

Kao što sam već i u uvodu napomenuo, genetski su algoritmi temeljeni na evoluciji u prirodi. Kako u evoluciji preživljavaju najjači, tako će i kod genetskih algoritama preživjeti skup najjačih, tj. najboljih rješenja. Pri reprodukciji dolazi do izmjene gena. Isto tako u prijelazu s generacije na generaciju mijenjaju se i rješenja kod genetskih algoritama. Taj proces zovemo **križanjem**. Uz križanje postoji još i znatno rjeđi proces, mutacija. **Mutacija** je proces slučajne promjene genetskog materijala do kojeg u prirodi dolazi pod utjecajem vanjskih uzroka. Križanje i mutaciju u terminologiji GA-a nazivamo **genetskim operatorima**, a proces izdvajanja najboljih jedinki unutar svake generacije **odabirom** ili **selekcijom**.

3.1 Populacija

Populacija je skup jedinki iste vrste smještenih na nekom području. Kako dio populacije stari i umire, tako se razmnožavanjem stvaraju novi potomci i veličina populacije u svakoj generaciji ostaje približno konstantna. U genetskom algoritmu populacija je skup potencijalnih rješenja zadanog problema. Početna populacija može biti odabrana slučajnim odabirom ili nekim drugim optimizacijskim postupkom. Odumiranje slabijih jedinki, tj. lošijih potencijalnih rješenja koje se nisu uspjele prilagoditi novim životnim uvjetima i opstanak jedinki koje su to uspjele u genetskim se algoritmima određuje uporabom **funkcije cilja ili dobrote**. Razmnožavanje jedinki koje su, zahvaljujući svojim dobrim svojstvima, uspjele preživjeti provodi se operacijom križanja.

Svakim novim ponavljanjem jedinke u populaciji poprimaju sve bolja svojstva. Algoritam obično završava dosezanjem zadanog broja ponavljanja, tj. broja generacija ili istekom prethodno određenog vremena za rad algoritma. Kada je uvjet završetka ispunjen, iz dobivene populacije odabire se najbolja jedinka i ona predstavlja rješenje optimizacijskog problema.

3.2 Funkcija cilja ili dobrote

Funkcija cilja predstavlja prirodnu okolinu koja vrši selekciju nad jedinkama. Što je jedinka bolje prilagođena okolini u kojoj živi, tj. što je rješenje bolje, veće je vjerojatnost njenog preživljavanja, tj. prenošenja u novu generaciju. Odabir odgovarajuće funkcije cilja ključan je problem kod implementacije genetskog algoritma. Također, s obzirom na to da se radi o funkciji koja se u algoritmu najviše koristi, funkcija cilja trebala bi biti što je moguće jednostavnija i brža.

3.3 Odabir ili selekcija

Svrha selekcije je čuvanje i prenošenje dobrih svojstava na sljedeću generaciju te izbacivanje loših svojstva. Selekcijom se biraju dobre jedinke koje se onda prenose na novu populaciju. Najjednostavniji postupak selekcije bio bi takav da se jedinke sortiraju po dobroti te da se izbaci ranije definiran broj najlošijih jedinki. Na taj bi

način proces optimiranja vrlo brzo završio. Zašto ipak onda ne koristimo taj postupak? Zato što i one najlošije jedinke imaju neka dobra svojstva koja treba iskoristiti. Kad se iz populacije izbace najlošije jedinke, ostaju samo najbolje, no te najbolje ne predstavljaju ujedno i skup svih najboljih svojstava. Čak i najlošije jedinke neka svojstva imaju bolja od najboljih. Tu činjenicu ne smijemo zanemariti. No, nije dobro niti ako dozvolimo da je vjerojatnost preživaljvanja svih jedinki podjednaka. Na taj bismo način omogućili gubitak velikog broja dobrih jedinki. Prema tome, problem je jedino smisleno riješiti dodjeljivanjem niske vjerojatnosti preživljavanja (ali veće od 0) lošim jedinkama, a visoke (no manje od 1) dobrim jedinkama. Određenom malom broju jedinki s najboljim svojstvima, **elitnim jedinkama**, možemo dodijeliti vjerojatnost preživaljavanja 1, tj. osigurati da će one sigurno nepromijenjene preći u novu generaciju.

Genetske algoritme s obzirom na vrstu selekcije možemo podijeliti na **generacijske** i **eliminacijske**. Generacijski genetski algoritmi u jednom ponavljanju raspolažu s dvije populacije. Karakteristične vrste selekcija koje koriste generacijski genetski algoritmi su **jednostavna selekcija** i **turnirska selekcija**. Karakteristika eliminacijskog genetskog algoritma je **eliminacijska selekcija**.

3.4 Elitizam

Kako se trenutno najbolja rješenja ne bi izgubila uptrebom genetskih operatora ili eliminacijom, tijekom selekcije javlja se potreba za zaštitom najbolje ili nekoliko najboljih jedinki od izmjena ili eliminacije. Zaštita najboljih jedinki naziva se elitizam i osigurava da se svaka nova generacija kreće prema globalnom optimumu. Zaštićene jedinke zovemo elitnim jedinkama. Ukoliko je populacija velika, prilikom pretrage za najboljim jedinkama algoritam se usporava.

4. Ostvarenje genetskih algoritama u Matlabu

Sustav Matlab uz mnoge svoje alate sadrži i alat za genetske algoritme. U ovom ću poglavlju opisati kako koristiti taj alat.

4.1 Pozivanje funkcija za genetske algoritme iz linije naredbi (engl. *command line*)

Za pokretanje genetskih algoritama iz linije naredbi treba pozvati funkciju za genetske algoritme sa sintaksom:

[x fval] = ga(@fitnessfun, nvars, options), gdje je:

@fitnessfun – handle za funkciju cilja,

nvars – broj nezavisnih ulaznih varijabli u funkciju cilja, a

options – struktura koja sadrži postavke za GA. Ako se ovaj argument izostavi, koriste se pretpostavljene postavke.

S ciljem dobivanja detaljnijih informacija, možete koristiti sljedeću sintaksu:

[x fval exitflag output population scores] = ga(@fitnessfcn, nvars, options).

Rezultat je dan s:

x – trenutak kada je dostignuta konačna vrijednost funkcije cilja,

fval – konačna vrijednost funkcije cilja,

exitflag – opisuje razlog zašto je GA prekinut,

output – struktura sadrži informacije o obavljanju GA u svakoj generaciji,

population – konačna populacija,

scores – konačna rješenja.

4.1.1 Pokretanje genetskih algoritama s pretpostavljenim postavkama

Za pokretanje genetskih algoritama s pretpostavljenim mogućnostima, funkciju za genetske algoritme možemo pozvati u kraćem obliku, tj.

[x fval] = ga(@fitnessfun, nvars).

Značenje argumenata objašnjeno je na početku poglavlja.

4.1.2 Postavljanje mogućnosti za genetske algoritme

Kroz strukturu options u funkciju ga prenose se sve postavke.

Za pregled svih varijabli strukture options i vrijednosti koje one mogu poprimiti u liniji naredbi treba upisati **gaoptimset** bez argumenata ili pogledati Tablica 4.1.

Prije korištenja strukture options kao argumenta funkcije ga, strukturu treba stvoriti. To se radi s naredbom:

options = gaoptimset(@ga).

Ovim će pozivom biti generirana struktura options s pretpostavljenim vrijednostima, tj:

PopulationType: 'doubleVector'
PopInitRange: [2x1 double]
PopulationSize: 20
EliteCount: 2
CrossoverFraction: 0.8000
ParetoFraction: []
MigrationDirection: 'forward'
MigrationInterval: 20
MigrationFraction: 0.2000
Generations: 100
TimeLimit: Inf
FitnessLimit: -Inf
StallGenLimit: 50
StallTimeLimit: Inf
TolFun: 1.0000e-006
TolCon: 1.0000e-006
InitialPopulation: []
InitialScores: []
InitialPenalty: 10
PenaltyFactor: 100
PlotInterval: 1
CreationFcn: @gacreationuniform
FitnessScalingFcn: @fitscalingrank
SelectionFcn: @selectionstochunif
CrossoverFcn: @crossoverscattered
MutationFcn: {[1x1 function_handle] [1] [1]}
DistanceMeasureFcn: []
HybridFcn: []
Display: 'final'
PlotFcns: []
OutputFcns: []
Vectorized: 'off'
UseParallel: 'never'.

Ove će postavke biti korištene i ako u funkciji GA izostavite argument options.

Do svake pojedine postavke u skupu postavki (options) možemo doći tako da upišemo options.tražena_postavka. Primjerice, ako napišemo option.PopulationSize, na ekranu će se ispisati vrijednost varijable PopulationSize (ako to pokrenemo kad su pohranjene pretpostavljene vrijednosti, ispisat će se 20). Želimo li unijeti novu vrijednost za npr. veličinu populacije, valja u liniji naredbi u Matlabu upisati:

Option.PopulationSize=nova_vrijednost.

Ukoliko pri generiranju postavki genetskih algoritama želite većinu postavki postaviti na pretpostavljene vrijednosti, no neke ipak želite promijeniti, to možete učiniti pozvavši funkciju:

options = gaoptimset('varijabla1', vrijednost1, 'varijabla2', vrijednost2,…).

Primjerice, options = gaoptimset('PopulationSize', 300, 'Vectorized', 'on'). Nakon izvođenja ove naredbe sve varijable strukture options poprimit će pretpostavljene vrijednosti osim varijabli PopulationSize i Vectorized koje će biti inicijalizirane s 300 i 'on'.

options = gaoptimset(oldopts,newopts) kombinira postojeću strukturu mogućnosti, oldopts, s novom strukturom mogućnosti, newopts. Svaki parametar u newopts s nepraznim vrijednostima pohranjuje se umjesto odgovarajućeg argumenta u oldopts.

Savjetujem vam da na kraj naredbe kojom mijenjate sadržaj strukture options ne stavljate znak „;“ (točka zarez) kako bi vam se, odmah nakon promjene, na zaslonu ispisale nove postavke.

Varijable strukture options, kratki opis i moguće vrijednosti nalaze se u Tablica 4.1.

Tablica 4.1 struktura options

Postavka (varijabla)	**Opis**	**Moguća vrijednost**
CreationFcn	Handle na funkciju koja kreira početnu populaciju	{@gacreationuniform}
CrossoverFcn	Handle na funkciju čiji se algoritam koristi za kreiranje djece pri križanju	@crossoverheuristic {@crossoverscattered} @crossoverintermediate @crossoversinglepoint @crossovertwopoint @crossoverarithmetic
CrossoverFraction	Dio populacije sljedeće generacije, ne uključuje djecu elitnih jedinki, koji je kreiran funkcijom križanja	Positive scalar | {0.8}
Display	Razina prikaza	'off' | 'iter' | 'diagnose' | {'final'}
DistanceMeasureFcn	Handle na funkciju koja računa udaljenost među jedinkama	{@distancecrowding, 'phenotype'}
EliteCount	Pozitivan cijeli broj koji nam govori koliko jedinki iz trenutne generacije sigurno prelazi u narednu generaciju, tj. koliko je elitnih jedinki	Pozitivan cijeli broj | {2}
FitnessLimit	Skalar. Kada funkcija cilja dosegne ovu vrijednost, algoritam staje	Skalar | {-Inf}
FitnessScalingFcn	Handle na funkciju koja mjeri vrijednosti funkcije cilja	@fitscalingshiftlinear @fitscalingprop @fitscalingtop {@fitscalingrank}
Generations	Pozitivan cijeli broj koji određuje	Pozitivan cijeli broj | {100}

	najveći mogući broj ponavljanja algoritma prije njegova prekidanja	
HybridFcn	Handle na funkciju koja nastavlja optimizaciju nakon što GA biva prekinut ili polje ćelija koje određuje hibridnu funkciju i mogućnosti njene strukture	Function handle | @fminsearch @patternsearch @fminunc @fmincon {[]} ili 1-2 polje ćelija| {@solver, hybridoptions}, gdje je solver = fminsearch, patternsearch, fminunc, or fmincon {[]}
InitialPenalty	Početna vrijednost parametra kazne	Pozitivan skalar | {10}
InitialPopulation	Početna populacija koja se koristi kao sjeme za genetski algoritam	Matrica | {[]}
InitialScores	Početna vrijednost koja određuje dobrotu; može biti djelomična	Jednostupčani vektor | {[ ]}
MigrationDirection	Smjer migracije	'both' | {'forward'}
MigrationFraction	Skalar između 0 i 1 koji određuje postotak jedinki u svakoj podpopulaciji koja migrira u različitu podpopulaciju	Skalar | {0.2}
MigrationInterval	Pozitivan cijeli broj koji određuje svakih koliko generacija dolazi do migracije	Pozitivan cijeli broj | {20}
MutationFcn	Handle na funkciju koja proizvodi mutiranu djecu	@mutationuniform @mutationadaptfeasible {@mutationgaussian}
OutputFcns	Polje handlova na funkcije koje crtaju podatke izračunate algoritmom	@gaplotbestf @gaplotbestindiv @gaplotdistance @gaplotexpectation @gaplotgeneology @gaplotselection @gaplotrange @gaplotscorediversity @gaplotscores @gaplotstopping | {[]}
PlotInterval	Pozitivan cijeli broj koji određuje svakih koliko generacija se vrši ispis	Pozitivan cijeli broj | {1}
PopInitRange	Matrica ili vektor koji određuje raspon jedinki u početnoj populaciji	Matrica ili vektor | [0;1]

PopulationSize	Veličina populacije	Pozitivan cijeli broj | {20}
PopulationType	String opisuje tip podataka populacije	'bitstring' | 'custom' | {'doubleVector'} Imajte na umu da linearna i nelinearna ograničenja nisu zadovoljena kad je PopulationType postavljeno na 'bitString' ili 'custom'.
SelectionFcn	Handle na funkciju koja bira roditelje djece koja nastaju križanjem i mutacijom	@selectionremainder @selectionuniform {@selectionstochunif} @selectionroulette @selectiontournament
StallGenLimit	Pozitivni cijeli broj. Algoritam staje ako u objektnoj funkciji nema napretka u narednih StallGenLimit generacija	Pozitivan cijeli broj |{50}
StallTimeLimit	Pozitivan skalar. Algoritam staje ako nema napretka u objektnoj funkciju StallTimeLimit sekundi	Pozitivan skalar | {Inf}
TimeLimit	Pozitivan skalar. Algoritam završava nakon što je pokrenut TimeLimit sekundi	Pozitivan skalar | {inf}
TolCon	Pozitivan skalar. TolCon se koristi za određivanje izvedivosti s obzirom na nelinearna ograničenja	Pozitivni skalar | {1e-6}
TolFun	Pozitivan skalar. Algoritam se izvodi sve dok je ukupna promjena u funkcije cilja kroz StallGenLimit manja od TolFun	Pozitivni skalar | {1e-6}
UseParallel	Računa funkciju cilja populacija paralelno	'always' | {'never'}
Vectorized	String određuje je li računanje funkcije prijelaza vektorizirano	'on' | {'off'}

4.1.3 Pojašnjenja postavki

4.1.3.1 Postavke crtanja

Postavke crtanja omogućuju vam crtanje podataka iz genetskog algoritma za vrijeme njegova izvođenja. Kada izaberete funkciju crtanja i pokrenete genetski algoritam, u prozoru za crtanje prikazuje se graf s dvije osi. Klikom na subplot otvara se veća inačica grafa u novom prozoru. Pritiskom na gumb **stop** u prozoru crtanja možete prekinuti izvođenje algoritma.

PlotInterval određuje svakih koliko generacija se vrši ispis.

Postavka **OutputFcns** prihvaća polje koje se sastoji od 0, 1 ili svih sljedećih funkcija:

@gaplotbestf crta najbolje vrijednosti po generacijama,

@gaplotexpectation crta očekivani broj djece s obzirom na rezultat pojedine generacije,

@gaplotscorediversity crta histogram rezultata pojedine generacije,

@plotstopping crta kriterije zaustavljanja,

@gaplotbestindiv crta karakteristike najboljih jedinki u generacijama,

@gaplotgenealogy crte genealogiju jedinki. Crte iz jedne generacije u drugu su označene različitim bojama s različitim značenjima:

Crvene crte označavaju djecu nastalu mutacijom,

Plave crte označavaju djecu nastalu križanjem,

Crne crte označavaju elitne jedinke,

@gaplotscores crta rezultate jedinki u pojedinim generacijama,

@gaplotmaxconstr crta najveće nelinearno prekoračenje ograničenja u pojedinoj generaciji,

@gaplotdistance crta prosječnu udaljenost jedinki u pojedinoj generaciji,

@gaplotrange crta minimum, maksimum i prosječnu vrijednost funkcije cilja u pojedinim generacijama,

@gaplotselection crta histogram roditelja.

Možete napisati i vlastitu funkciju za crtanje.

Prvi redak funkcije za crtanje koju pišete mora imati oblik:

function state = plotfun(options, state, flag)

Ulazni argumenti u funkciju su:

options – struktura sadrži sve trenutne postavke

state – struktura sadrži informacije o trenutnoj populaciji

flag – string koji nam govori koji se dio algoritma trenutno izvodi

4.1.3.2 Postavke populacije

Postavke populacije omogućuju vam postavljanje postavki populacije koje onda GA koristi.

PopulationType određuje tip ulaznih podataka u funkciju cilja. PopulationType može biti:

'doubleVector' – koristiti ovu postavku ako su jedinke u populaciji tipa double. (ovo je ujedno i pretpostavljena postavka),

'bitstring' – koristite ovu postavku ako su jedinke u populaciji mali stringovi,

'custom' - koristite ovu postavku za kreiranje populacije čiji tip nije jedan od prethodno navedenih.

Ukoliko ste se odlučili koristiti vlastiti tip populacije ('custom'), morate napisati vlastite funkcije za mutaciju (MutationFcn), križanje (CrossoverFcn) i kreiranje početne populacije (CreationFcn) i postavkama navedenim u zagradi pridružiti te novo napisane funkcije. Primjerice:

options.MutationFcn=@moja_funkcija_za_mutaciju.

Kako napisati vlastite funkcije za mutaciju križanje i kreiranje početne populacije objašnjeno je kasnije u poglavljima posvećenim tim postavkama.

PopulationSize određuje koliko će jedinki biti u pojedinoj generaciji. Što je PopulationSize veći broj, genetski algoritam rješenje traži temeljitije čime se smanjuje mogućnost da algoritam vrati lokalni minimum koji ujedno nije i globalni, ali se i povećava vrijeme traženja.

Ukoliko ste za PopulationSize umjesto obične konstante postavili vektor, genetski algoritam će kreirati toliko podpopulacija kolika je duljina vektora. Veličina pojedine podpopulacije određena je pripadajućim elementom vektora. Npr. ako za ulazni vektor postavite [4 5 7 6] kreirat će se 4 podpopulacije (vektor ima 4 elementa) od kojih će prva podpopulacija imati 4 jedinke, druga 5, treća 7, a četvrta 6.

InitialPopulation određuje početnu generaciju u genetskom algoritmu. Pretpostavljena vrijednost je [] i u tom slučaju GA koristi ranije postavljenu funkciju za generiranje populacija (CreationFcn) za generiranje početne populacije. Ukoliko ste se odlučili sami zadati početnu populaciju, to možete učiniti tako da postavci InitialPopulation pridružite neprazan vektor. Pri tome valja paziti da broj redaka u tom vektoru ne smije biti veći o veličine populacije (PopulationSize), a broj stupaca mora biti jednak broju ulaznih varijabli u funkciju cilja. Ukoliko unesete manje jedinki početne populacije nego što je veličina populacije, preostale jedinke će generirati funkcija za kreiranje početne populacije (CreationFcn).

PopInitRange određuje domet vektora u početnoj populaciji koji su generirani u funkciji za kreiranje početne populacije (CreationFcn). PopInitRange možete postaviti kao matricu s dva retka i stupaca koliko ima varijabli funkcija cilja gdje svaki stupac ima oblik [dg, gg] . dg je donja granica, a gg gornja granica.

4.1.3.3 Postavke selekcije

Postavke selekcije određuju kako GA odabire roditelje preko kojih stvara jedinke sljedeće generacije.

SelectionFcn – ovom postavkom možete odrediti koju će funkciju GA koristiti za selekciju. Možete izabrati:

@selectionstochunif – Pretpostavljena funkcija selekcije, stohastička uniformna selekcija, postavlja liniju u kojoj svaki roditelj odgovara dijelu linije duljine proporcionalne svojoj skaliranoj vrijednosti. Algoritam se pomiče uzduž linije u jednakim razmacima. U svakom koraku, algoritam alocira jednog roditelja iz sekcije koju očitava. Prvi korak jest da stvori proizvoljan broj manji od broja koraka.

@selectionremainder – pripisuje roditelje deterministički od cijelog dijela svake individualne skalirane vrijednosti i onda koristi rulet selekciju na ostalom decimalnom

dijelu. Npr., ako je skalirana vrijednost nekog pojedinca 2.3, taj pojedinac se izlistava 2 puta kao roditelj jer je cijeli dio 2. Nakon što su roditelji pripisani prema njihovim cijelim dijelovima, ostatak roditelja se bira stohastički. Vjerojatnost da roditelj bude odabran u ovom koraku je proporcionalna parcijalnom dijelu njegove skalirane vrijednosti.

@selectionuniform – Uniformna selekcija odabire roditelje koristeći očekivanja i broj roditelja. Uniformna selekcija je korisna za testiranje i debagiranje, ali nije isplativa kod strategije pretraživanja.

@selectionroulette – Rulet selekcija odabire roditelje oponašajući vrtnju ruleta, u kojem područje dijela kotača ruleta koji odgovara jedinki proporcionalno s očekivanjem jedinke. Algoritam koristi slučajno odabrane brojeve za spajanje područja s odgovarajućom vjerojatnošću.

@selectiontournament – Turnirska selekcija bira svakog roditelja birajući igrače turnirske veličine nasumično i onda birajući najboljeg pojedinca od toga koji će postati roditelj. Turnirska veličina mora biti barem 2. Pretpostavljena turnirska veličina je 4.

Da biste promijeniti pretpostavljenu turnirsku veličinu, valja koristiti sljedeću sintaksu:

options = gaoptimset('SelectionFcn',...{@selecttournament, size}).

Moguće je napisati i vlastitu funkciju selekcije. Funkcija se poziva kao @ime_funkcije. Ta se funkcija postavlja kao funkcija selekcije naredbom:

options = gaoptimset('SelectionFcn', @ime_funkcije);

Vaša funkcija selekcije mora imati prototip:

function parents = myfun(expectation, nParents, options), gdje je

Expectation – očekivani broj djece za svakog člana populacije,

nParents – broj roditelja za odabir,

options – GA options struktura.

Funkcija vraća roditelje, retčani vektor duljine nParents koji sadrži popis odabranih roditelja.

4.1.3.4 Postavke reprodukcije

Postavke reprodukcije određuju kako genetski algoritam stvara djecu za novu generaciju.

EliteCount određuje broj elitnih jedinki, tj. jedinki koje sigurno prelaze u novu generaciju. EliteCount treba postaviti na pozitivnu cjelobrojnu vrijednost koja je manja od veličine populacije (PopulationSize). Pretpostavljena vrijednost je 2.

CrossoverFraction određuje dio nove generacije (u koji ne ulaze elitne jedinke) koji nastaje križanjem. Postavite CrossoverFraction na vrijednost između 0 i 1. Pretpostavljena vrijednost je 0.8.

4.1.3.5 Postavke mutacije

Postavke mutacije određuju kako genetski algoritam radi male slučajne promjene jedinki populacije s ciljem izrade mutirane djecu.

Mutacija omogućava genetsku raznolikost i genetskom algoritmu širi prostor traženja rješenja. Za odabir načina mutacije koristi se postavka MutationFcn. Postavka MutatioFcn može poprimiti sljedeće vrijednosti:

@mutationgausian – pretpostavljena funkcija za mutaciju, Gaussova, dodaje slučajan broj dobiven Gaussovom razdiobom s prosječnom 0 za svaki ulaz roditeljskog vektora. Standardno odstupanje (devijacija) raspodjele određeno je parametrima **Scale** and **Shrink** i preko postavke **PopInitRange** u strukturi options.

Paramerar Scale određuje standardno odstupanje u prvoj generaciji. Ako PopInitRange podesite da bude vektor v dimenzija 2×1, početno standardno odstupanje jednako je za sve koordinate roditeljskog vektora, a određeno je kao Scale*(v(2) - v(1)).

Ako postavite PopInitRange tako da bude vektor v s dva retka i stupaca koliko ima varijabli funkcija cilja, početno standardno odstupanje vektora roditelja u koordinati dano je izrazom Scale*(v(i,2) - v(i,1)).

Parametar Shrink nadzire kako se standardno odstupanje smanjuje s novim generacijama. Ako PopInitRange podesite da bude vektor dimenzija 2×1, standardno odstupanje k-te generacije, σ_k, jednako je za sve koordinate roditeljskog vektora i određeno je rekurzivnim izrazom:

$$s_k = s_{k-1}\left(1 - Shrink\frac{k}{Generations}\right).$$

Ako postavite PopInitRange tako da bude vektor v s dva retka i stupaca koliko ima varijabli funkcija cilja, standardno odstupanje vektora roditelja u koordinati i u k-toj generaciji, $\sigma_{i,k}$ dano je rekurzivnim izrazom:

$$s_{i,k} = s_{i,k-1}\left(1 - Shrink\frac{k}{Generations}\right).$$

Ako za vrijednost parametra Shrink postavite 1, algoritam linearno smanjuje standardno odstupanje pojedine koordinate sve dok u posljednjoj generaciji standardno odstupanje ne postane 0. Negativna vrijednost parametra Shrink rezultira porastom standardnog odstupanja iz generacije u generaciju.

Vrijednosti parametara Scale i Shrink možete preko linije naredbi postavljati pomoću naredbe oblika:

options = gaoptimset('MutationFcn', ... {@mutationgaussian, scale, shrink}).

@mutationuniform – Uniformna ili jednostavna mutacija je dvostupanjski proces. Prvo, algoritam odabire dijelove vektora zapisa jedinki za mutaciju, gdje svaki zapis ima vjerojatni **omjer** mutacije. Pretpostavljena vrijednost omjera je 0.01. U drugom koraku, algoritam zamjenjuje pojedine označene zapise sa slučajno generiranim brojevima jednoliko odabranim unutar raspona zapisa.

Vrijednosti parametara Scale i Shrink možete preko linije naredbi postavljati pomoću naredbe oblika:

options = gaoptimset('MutationFcn', {@mutationuniform, omjer})

@mutationadaptfeasible – nasumično stvara upute koje su prilagodljive redom do zadnjeg uspješnog ili neuspješnog stvaranja generacije. Moguće područje je omeđeno ograničenjima i neujednačenim ograničenjima. Duljina koraka se bira uz svaku uputu tako da linearna ograničenja i granice budu zadovoljeni.

Kao i za sve do sada obrađene postavke, i način mutacije možete odrediti vlastitom M-funkcijom. Vaša funkcija za mutaciju treba imati ovakav prototip:

function mutationChildren = vaša_funkcija_za_mutaciju(parents, options, nvars, FitnessFcn, state, thisScore, thisPopulation),

pri čemu ulazni argumenti imaju sljedeća značenja:

parents – retčani vektor s roditeljima izabranim pomoću funkcije selekcije,

options – struktura options,

nvars – broj varijabli,

fitnessFcn – funkcija cilja,

state – struktura koja sadrži informacije o trenutnoj generaciji. Struktura state opisuje okruženje,

thisScore – vektor s bodovima za trenutnu populaciju,

thisPopulation – matrica s jedinkama trenutne populacije.

Naravno, vašu funkciju za mutaciju postavit ćete s naredbom:

options = gaoptimset('MutationFcn', @vaša_funkcija_za_mutaciju);

4.1.3.6 Postavke križanja

Postavke križanja određuju kako genetski algoritam križa dvije jedinke ili dva roditelja s ciljem izrade „križanac“ u novoj generaciji.

CrossoverFcn predstavlja funkciju koja provodi križanje. Možete izabrati jednu od sljedeći funkcija:

@crossoverscattered, pretpostavljena funkcija križanja, bira slučajni binarni vektor i tamo gdje vektor ima vrijednost 1 stavlja gene prvog roditelja, a gdje ima vrijednost 0, gene drugog roditelja. Primjerice, kada bi prvi roditelj bio zadan vektorom r1, drugi vektorom r2 i kada bi slučajni binarni vektor bio sbv, vektor djeteta bi bio d.

r1 = [a b c d e f g h]

r2 = [1 2 3 4 5 6 7 8]

sbv = [1 1 0 0 1 0 0 0]

d = [a b 3 4 e 6 7 8]

@crossoversinglepoint bira slučajan cijeli broj n s intervala (1, broj ulaznih varijabli funkcije cilja) i nakon toga:

Označava zapise vektora prvog roditelja na pozicijama manjim ili jednakim n

Označava zapise vektora drugog roditelja na pozicijama većim od n

Spaja označene zapise u vektor dijete

Primjerice, ako je r1 prvi roditelj, r2 drugi, a n je poprimio vrijednost 3, dijete će biti d.

r1 = [a b c d e f g h]

r2 = [1 2 3 4 5 6 7 8]

d = [a b c 4 5 6 7 8]

@crossovertwopoint odabire dva slučajna pozitivna cijela broja m i n s intervala (1, broj ulaznih varijabli funkcije cilja) i nakon toga:

Označava zapise vektora prvog roditelja na pozicijama manjim ili jednakim m te većim od n

Označava zapise vektora drugog roditelja na pozicijama većim od m, a manjim ili jednakim od n

Spaja označene zapise u vektor dijete

Primjerice, ako je r1 prvi roditelj, r2 drugi, m je poprimio vrijednost 3, a n 6, dijete će biti d:

r1 = [a b c d e f g h]

r2 = [1 2 3 4 5 6 7 8]

d = [a b c 4 5 6 g h]

@crossoverintermediate kreira djecu uzimanjem težinskog prosjeka roditelja. Težine možete odrediti preko parametra **omjer** koji može biti skalar ili retčani vektor duljine broja ulaznih varijabli funkcije cilja. Pretpostavljeni vektro je s 1 na svim mjestima. Funkcija kreira dijete od roditelja1 i roditelja2 s formulom:

dijete = roditelj1 + rand * omjer * (roditelj2 - roditelj1).

Ako sve vrijednosti parametra omjer leže na intervalu [0, 1], nastala djeca su unutar hiperkocke definirane poretkom roditelja na suprotnim vrhovima. Ako omjer nije unutar tog intervala, djeca se mogu nalaziti izvan hiperkocke. Ako je omjer skalar, onda sva djeca leže na bridu koji spaja dva roditelja.

Kako biste promijenili pretpostavljenu vrijednost parametra omjer, koristite sljedeću naredbu:

options = gaoptimset('CrossoverFcn', ... {@crossoverintermediate, ratio});

@crossoverheuristic vraća dijete koje leži na crti koja spaja dva roditelja. Dijete je bliže roditelju s boljim svojstvima. Preko parametra **omjer** možete odrediti koliko je dijete udaljeno od boljeg roditelja. Pretpostavljena vrijednost parametra omjer je 1,2. Ako su roditelj1 i roditelj2 roditelj pri čemu je roditelj1 roditelj s boljim svojstvima, dijete se dobiva pomoću formule:

Dijete = roditelj2 + omjer* (roditelj1-roditelj2).

Kako biste promijenili pretpostavljenu vrijednost parametra omjer, koristite sljedeću naredbu:

options=gaoptimset('CrossoverFcn',... {@crossoverheuristic, omjer});

@crossoverarithmetic stvara djecu koja su težinska aritmetička sredina roditelja. Djeca su uvijek ostvariva s obzirom na linearna ograničenja i granice.

Ukoliko ste odlučili napisati vlastitu funkciju križanja, ona mora imati prototip:

xoverKids = myfun(parents, options, nvars, FitnessFcn, unused, thisPopulation)

pri čemu ulazni argumenti imaju sljedeća značenja:

parents – retčani vektor s roditeljima izabranim pomoću funkcije selekcije

options – struktura options

nvars – broj varijabli

fitnessFcn – funkcija cilja

unused – rezervirano mjesto se ne korisit

thisPopulation – matrica s jedinkama trenutne populacije. Broj redaka jednak je PopulationSize, a broj stupaca broju ulaznih varijabli funkcije cilja

Funkcija vraća xoverKids – plod križanja – kao matricu čiji redci predstavljaju novonastalu djecu. Broj stupaca jednak je broju ulaznih varijabli funkcije cilja.

4.1.3.7 Postavke migracije

Postavke migracije određuju kako jedinke putuju među podpopulacijama. Naravno, da bi se migracija mogla dogoditi, nužno je ostvariti osnovne preduvjete za postojanje podpopulacija, tj. broj jedinki u populaciji (PopulationSize) mora biti veći od 1. Kada dođe do migracije, najbolje jedinke iz jedne podpolulacije zamjenjuju najlošije jedinke u drugoj podpopulaciji. Jedinke koje migriraju iz jedne podpopulacije u drugu samo su kopirane u drugu podpopulaciju, tj. nisu uklonjene iz vlaste podpolulacije.

Način na koji će se vršiti migracija možete odrediti postavljenjem sljedećih postavki:

MigrationDirection – omogućava određivanje smjera migracije. Migracija se može odvijati u 1 ili u oba smjera. MigrationDirection može poprimiti vrijednosti:

'forward' – migracija se odvija prema naprijed, tj. n-ta podpopulacija migrira u n+1. podpopulaciju,

'both' – n-ta podpopulacija migrira u n+1. i n-1. podpopulaciju.

Skup podpopulacija možemo si predstaviti kružnicom, tj. prva i zadnja podpopulacija su susjedne pa su moguće i migracije među njima.

MigrationInterval – određuje svakih koliko generacija dolazi do migracije. Npr. ako je MigrationInterval 20, migracija će nastupiti nakon svakih 20 generacija.

MigrationFraction – određuje koliko se jedinki putuje iz jedne u drugu podpopulaciju. Broj jedinki koje migriraju se određuje tako da se MigrationFraction pomnoži s brojem jedinki u manjoj od podpopulacija među kojima se odvija migracija. Primjerice, kada bi se migracija odvijala među populacijama od 50 i 100 jedinki, a

MigrationFraction bio postavljen na 0.1, broj jedinki koje migriraju bio bi jednak 0,1*50=5.

4.1.3.8 Postavke za zaustavljanje genetskog algoritma

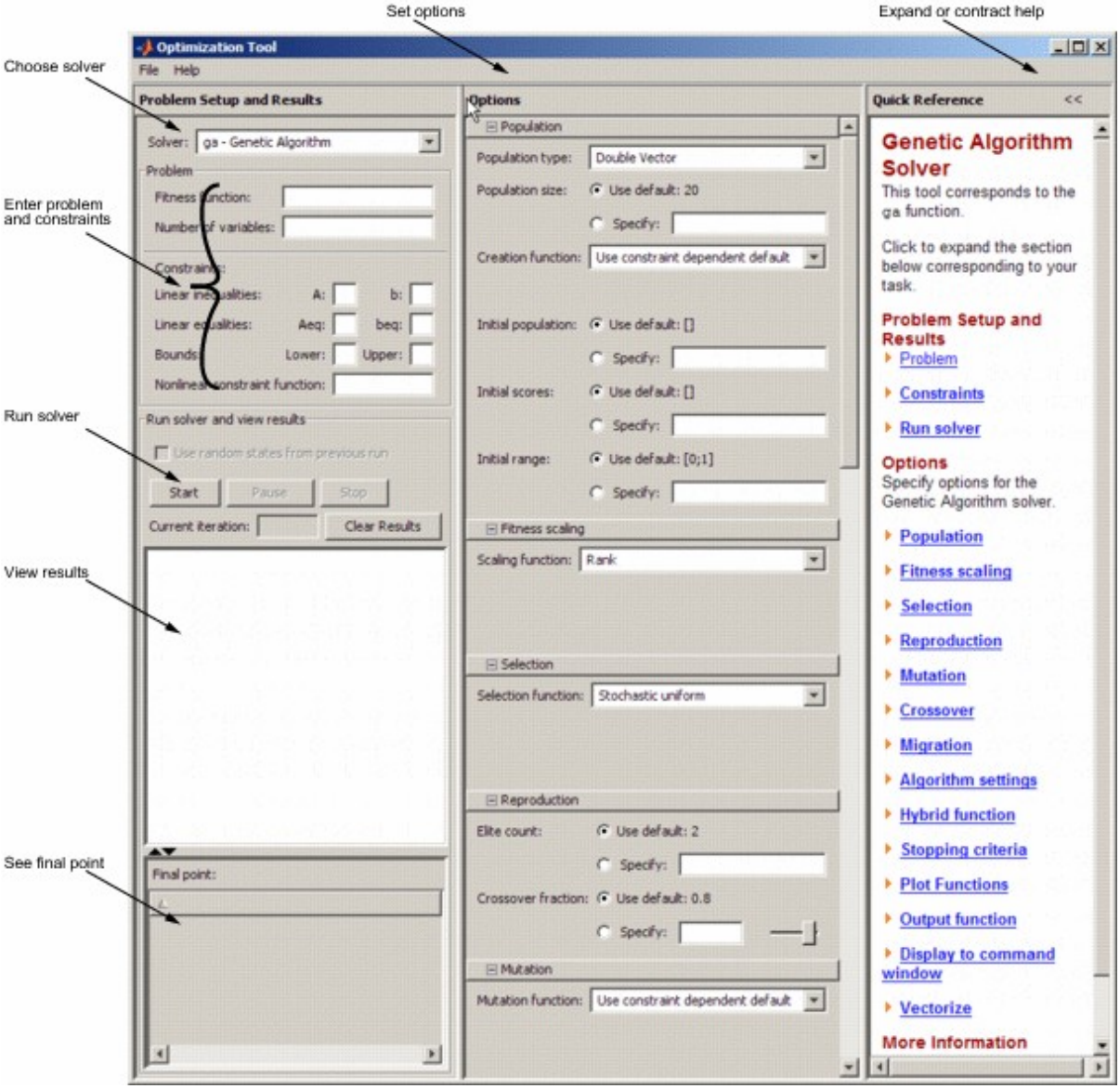
Generations – određuje najveći broj ponavljanja algoritma, tj. koliko najviše generacija može biti. Pretpostavljena vrijednost je 100.

TimeLimit – određuje maksimalno vrijeme u sekundama nakon kojeg GA prestaje s radom.

4.2 Korištenje Optimization toola

Optimization toolom možete napraviti sve što možete napraviti i pozivom funkcije ga iz linije naredbi. Postavke koje možete postavljati u Optimization toolu istovjetne su onima koje se u liniji naredbi postavljaju u strukturi options.

Za pokretanje Optimization toola valja u liniji naredbi upisati **optimtool('ga')**.



Slika 4.2 Izgled Optimization toola

Prije početka korištenja Optimization toola trebate unijeti sljedeće podatke:

Funkciju cilja (Fitness function) – funkcija koju želimo minimizirati. Funkciju cilja treba unijeti u formatu @ime_funkcije_cilja, gdje je fitnessfun.m M-funkcija koja računa funkciju cilja.

Broj varijabli (Number of variables) – broj elemenata ulaznog vektora, tj. broj ulaznih varijabli u funkciju cilja.

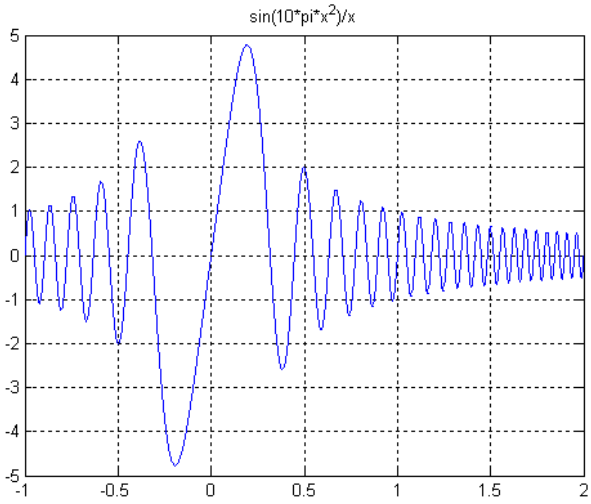
U okviru Constraints moguće je unositi ograničenja ili linearnu funkciju ograničenja. Ako problem nije ograničen, polja navedenog okvira mogu se ostavitri prazna.

Za pokretanje genetskog algoritma valja pritisnuti gumb Start. Rezultati otpimizacije ispisuju se u okviru Run solver and view results.

U okviru Options možete mijenjati postavke.

4.3 Primjeri

4.3.1 Primjer 1., $f_1(x) = \frac{\sin(10px^2)}{x}$



Slika 4.3 funkcija $f_1(x) = \frac{\sin(10px^2)}{x}$

Budući da funkcija ga vraća minimum funkcije cilja i da u Matlabu ne postoji funkcija koja pronalazi maksimum funkcije cilja, maksimum ćemo tražiti tako da tražimo minimum funkcije cilja suprotnog predznaka, tj. za tražit ćemo minimum funkcije –f(x).

```
function min=f1(x)
%prva ispitna funkcija za GA
pom=sin(10*pi.*x.^2)./x;
min=pom;
```

Slika 4.4 M-funkcija f1(x)

No, za početak, funkciju s kojom radimo (objektnu funkciju) moramo nekako prikazati u Matlabu. Time objektna funkcija postaje funkcija cilja, a prikazat ćemo je kao M-funkciju. Kod M-funkcije koja predstavlja funkciju iz ovog primjera nalazi se na Slika 4.4.

Prije no što započnem s traženjem ekstrema, moram stvoriti strukturu options. To ću napraviti s naredbom:

options = gaoptimset(@ga)

Na ovaj sam način stvorio strukturu options s pretpostavljenim vrijednostima.

Kako bih pronašao minimum zadane funkcije, u liniju naredbi u Matlabu ću upisati

[x fval] = ga(@f1, 1, options).

Ne bi li se uvjerio u točnost rješenja, naredbu ću pokrenuti nekoliko puta. Svakim novim pokretanjem mijenjaju se znamenke iza druge decimale varijable x. To mi se ne sviđa jer sam odlučio pronaći položaj minimuma s preciznošću na 4 decimale. Kako bih to postigao, moram popraviti postavke genetskog algoritma. Odlučio sam povećavati veličinu populacije i broj generacija. Rezultat stabilan na 4 decimale u 10 uzastopnih poziva funkcije ga dobio sam nakon što sam populaciju povećao s 20 na 2000 jedinki, a broj generacija sa 100 na 4000. To sam napravio s naredbom:

options = gaoptimset('PopulationSize', 2000, 'Generations', 4000).

Za poziciju minimuma dobio sam točku (0,1926, -4,7712).

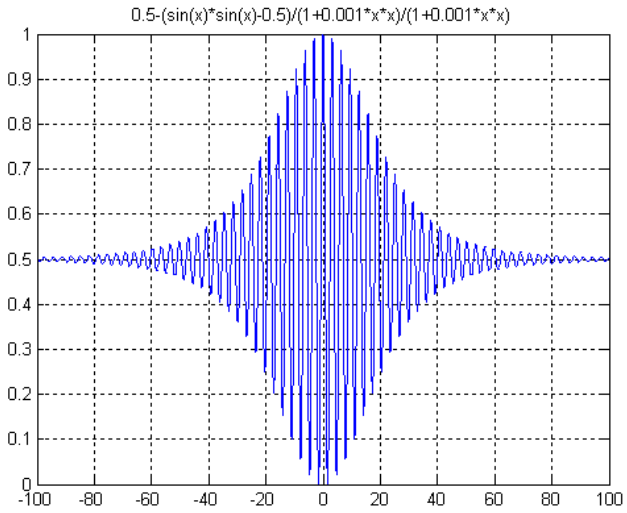
Kako bih pronašao globalni maksimum, moram promijeniti funkciju cilja. Tražit ću minimum funkcije –f1(x). Funkcija cilja nakon promjene prikazana je na Slika 4.5.

```
function max=f1(x)
%prva ispitna funkcija za GA
pom=-sin(10*pi.*x.^2)./x;
max=pom;
```

Slika 4.5 M-funkcija $-f_1(x)$

Funkcija ga mi nakon poziva promijenjene funkcije javlja da se minimum nalazi u točci (0,1926, -4,7712). Iz ovog rezultata odmah mogu iščitati x koordinatu maksimuma tražene funkcije. To je 0,1926. Budući da sam tražio minimum za suprotnu funkciju, predznak vrijednosti funkcije, tj. y koordinata bit će suprotnog predznaka od dobivene vrijednosti. Znači, maksimum tražene funkcije nalazi se u točci (0,1926, 4,7712).

4.3.2 Primjer 2., $f_2 = 0.5 - \frac{\sin^2(x) - 0.5}{(1 + 0.001x^2)^2}$



Slika 4.6 funkcija $f_2 = 0.5 - \frac{\sin^2(x) - 0.5}{(1 + 0.001x^2)^2}$

Kod M-funkcije kojom u Matlabu predstavljamo ovu funkciju nalazi se na Slika 4.7.

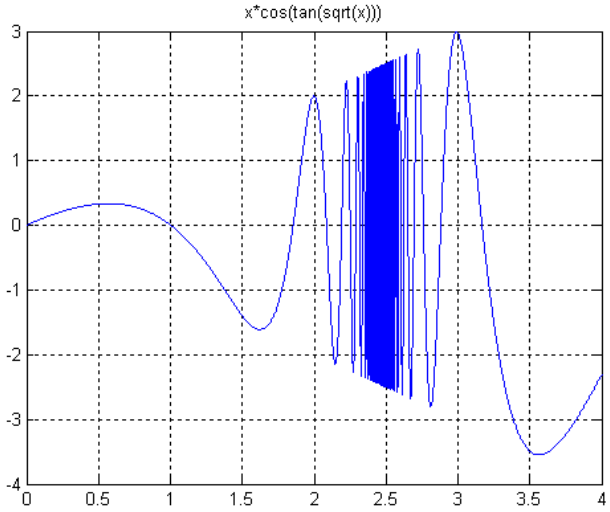
```
function min=f2(x)

%druga ispitna funkcija za GA

pom=0.5-
(1./((1+0.001.*x.*x).*(1+0.001.*x.*x)).*(sin(x).*sin(x)-
0.5));

min=pom;
```

Slika 4.7 M-funkcija $f_2(x)$

Kako bih odmah dobio preciznije rezultate, odmah će veličinu populacije postaviti na 2000, a broj generacija na 4000. S tim postavkama dobivam da se minimun nalazi u točkama (-1.5692, 0.0025) i (1.5692, 0.0025).

Maksimum ću ponovo tražiti tako da tražim minimum suprotne funkcije. Dobivam da se maksimum nalazi u točci (0, 1).

4.3.3 Primjer 3., $f(x) = x\cos(tg(\sqrt{x}))$



Slika 4.8 funkcija $f(x) = x\cos(tg(\sqrt{x}))$

Kod M-funkcije kojom je predstavljena ova funkcija nalazi se na Slika 4.9.

```
function min=f3(x)
%treća ispitna funkcija za GA
pom=x.*cos(tan(sqrt(x)));
min=pom;
```

Slika 4.9 M-funkcija $f_3(x)$

Skoro sa svakim novim pozivom funkcije ga dobivam novu vrijednost minimuma. Neke od točaka koje sam dobio kao rješenje su: (-23,8830, -36.8501), (-22,2442, -34.3203), (3,5718, -3.5482), (-745.1753, -1149,9).

Pri traženju maksimuma s pretpostavljenim postavkama i veličinom populacije od 2000 jedinki i ponavljanjem algoritma kroz 1000 generacija dobivam da se maksimum nalazi u točci (12,6473, 11.4413). Ovu sam vrijednost dobio čak desetak puta za redom. No, kada se nacrta graf funkcije na širem intervalu, postaje jasno da ta točke nije globalni maksimum. Zato sam odlučio smanjiti populaciju na 4 jedinke i definirati početnu populaciju kao [1000; 2000; 3000; 4000] kako bih vidio što će tada ga vratiti za maksimum funkcije. Na ovaj sam način područje traženja genetskog algoritam proširio i dobio sam da se maksimum nalazi u točci (4000, 3620,6). Nakon toga sam početnu populaciju odlučio smjestiti puno dalje i postavio sam je kao [10000; 20000; 30000; 40000]. Sada dobivam da je maksimum u točci (30000, 27097). Početnu populaciju postavio sam s naredbom:

options.InitialPopulation=[10000; 20000; 30000; 40000].

Iz ovih rezultata i iz grafa funkcije zaključujem da nema niti globalni minimum, niti globalni maksimum, tj. da u oba smjera ide u beskonačno.

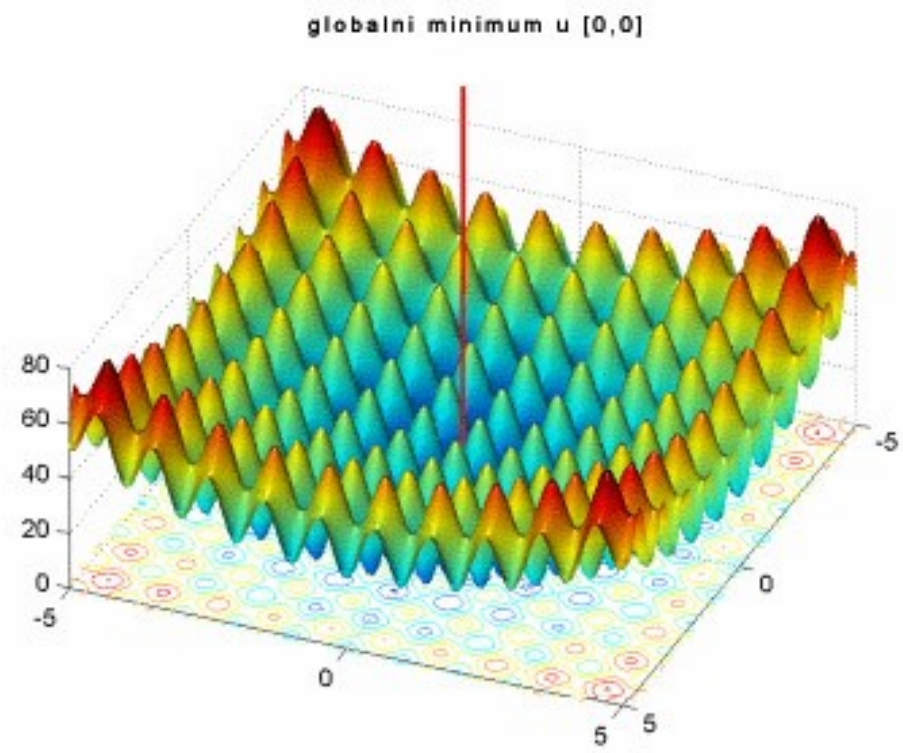
4.3.4 Primjer 4., Rastriginova funkcija

U primjeru ću pokazati kako pronaći minimum Rastriginove funkcije koja se često korisiti za provjeru genetskih algoritama.

Rastriginova funkcija je funkcija dvije varijable i definirana je ovako:

$$Ras(x) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_1).$$

Slika 4.10 predstavlja graf ove funkcije.



Slika 4.10 Rastriginova funkcija

Matlab sadrži M-funkciju rastriginsfcn.m koja računa vrijednosti Rastriginove funkcije.

```
function scores = rastriginsfcn(pop)
%RASTRIGINSFCN Compute the "Rastrigin" function.

%   Copyright 2003-2004 The MathWorks, Inc.
%   $Revision: 1.3.4.1 $  $Date: 2004/08/20 19:50:22 $


    % pop = max(-5.12,min(5.12,pop));
    scores = 10.0 * size(pop,2) + sum(pop .^2 - 10.0 * cos(2 * pi .*
pop),2);
```

Slika 4.11 M-funkcija za Rastriginovu funkciju

Kao što možete vidjeti na grafu, funkcija ima mnogo lokalnih minimuma. Međutim, funkcija ima samo jedan globalni minimum koji se nalazi u točci (0,0). Na bilo kojem drugom mjestu vrijednost lokalnog minimuma je veća od 0.

Kako biste pronašli minimum Rastringove funkcije u liniji naredbi treba upisati:

[x fval] = ga(@rastriginsfcn, 2, options)

Primijetite da je u ovom primjeru drugi argument u funkciji ga 2, a ne 1 kao u prva tri primjera. Drugi argument predstavlja broj ulaznih varijabli u funkciju cilja. U prva smo tri primjera radili s funkcijama 1 varijable pa je taj argument bio 1, no kako je Rastriginova funkcija funkcija 2 varijable, drugi po redu argument u funkciji ga mora biti 2.

S pojačavanjem postavki, rješenja koja ćemo dobivati pomoću funkcije ga bit će sve bliža posve točnom rješenju, no i vrijeme izvođenja funkcije će se povećati.

5. Zaključak

Genetski su algoritmi odličan alat za rješavanje mnogih problema za koje ne postoje deterministički algoritmi ili je njihova složenost prevelika. Čest primjer problema za koji je deterministički algoritam velike složenosti, a moguće ga je riješiti primjenom genetskih algoritama, problem je trgovačkog putnika. Vrijeme izvođenja determinističkog algoritma već je za dvadesetak gradova toliko veliko da je tim algoritmom besmisleno uopće i započeti potragu za najkraćim putem.

Kada biste sami pisali cijeli genetski algoritam za rješavanje nekog problema, bilo bi vam potrebno mnogo vremena. Zato postoje različite gotove funkcije koje rade glavninu genetskog algoritma. Programer koji želi riješiti određeni problem treba napisati funkciju cilja koja predstavlja taj njegov problem i pozvati već gotove funkcije genetskog algoritma nad tom svojom funkcijom cilja. Mnogo je različitih alata koji omogućuju ovakav rad s genetskim algoritmima. Jedan od tih alata je i Matlab. U ovom sam vam radu pokušao prezentirati kako pomoću gotovih funkcija u Matlabu riješiti problem korištenjem genetskih algoritama. Nadam se da sam u tom naumu uspio.

6. Literatura

[1] Golub, M. Genetski algoritam. Verzija 2.3. 27. rujna 2004.

[2] Čeri, M., Donđivić, L., Kokan, I., Lučić, M., Malović, M., Mišljenčević, N., Pađen, I., Radanović, G., Spasojević, B. Evolucijski algoritmi. Projekt. FER, 2007.

[3] Genetic Algorithm and Direct Search Toolbox, *Genetic Algorithm and Direct Search Toolbox, http://www.mathworks.com/access/helpdesk/help/toolbox/gads,* 15. travnja 2008.

7. Sažetak

Uz mnoge svoje mogućnosti, Matlab ima i mogućnost rješavanja optimizacijskih problema primjenom genetskih algoritama. Glavni zadatak programera koji rješava neki optimizacijski problem jest da taj problem prikaže kao funkciju cilja. Za tu funkciju cilja u Matlabu poziva funkciju ga koja pronalazi njen minimum. Kako bi pronašao maksimum objektne funkcije, treba tražiti minimum suprotne funkcije. Način rada genetskog algoritma, tj. njegove postavke zapisane su u strukturi options. Postavke genetskog algoritma mogu se mijenjati tako da se mijenjaju dijelovi strukture options.