Optimiranje nelinearnih funkcija cilja primjenom genetskih algoritama

Šprljan, Mario

Master's thesis / Diplomski rad

2010

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:627797

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

DIPLOMSKI RAD

Mario Šprljan

Zagreb, 2010.

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

OPTIMIRANJE NELINEARNIH FUNKCIJA CILJA PRIMJENOM GENETSKIH ALGORITAMA

Mentor:
prof.dr.sc. Nedeljko Štefanić

Student:
Mario Šprljan

Zagreb, 2010.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **MARIO ŠPRLJAN** Mat. br.:0023042855

Naslov: **OPTIMIRANJE NELINEARNIH FUNKCIJA CILJA PRIMJENOM GENETSKIH ALGORITAMA**

Opis zadatka:

Prilikom optimizacije odzivnih funkcija koje predstavljaju rezultat prilikom eksperimentalnih istraživanja prirode tehnoloških procesa, analitičke i numeričke metode koje se koriste u nelinearnom programiranju pokazuju značajna ograničenja kako se povećava složenost odzivne funkcije. Genetski algoritmi su jedna od mogućih metoda za pronalaženje optimalnog rješenja složenih nelinearnih odzivnih funkcija.

U radu je potrebno:

1. Razraditi matematički model nelinearnog programiranja.
2. Sistematizirati numeričke metode za optimizaciju nelinearnih funkcija cilja.
3. Detaljno objasniti osnove genetskih algoritama te pokazati način njihove primjene u nelinearnom programiranu.
4. Na odabranom realnom tehnološkom procesu iz strojarske prakse testirati pogodnost genetskih algoritama.
5. Procijeniti prednosti primjene genetskih algoritama pred klasičnim metodama optimiranja nelinearnih funkcija cilja.
6. Procijeniti moguće nedostatke i ograničenja primjene genetskih algoritama u strojarskoj praksi.

Zadatak zadan:
11. ožujka 2010.

Rok predaje rada:
Ožujak 2011.

Zadatak zadao:
Prof.dr.sc. Nedeljko Štefanić

Predsjednik Povjerenstva:
Prof. dr. sc. Franjo Cajner

Sažetak

Tema ovog DIPLOMSKOG RADA je primjena genetskih algoritama u optimizaciji nelinearnih funkcija cilja, uključujući probleme bez i sa ograničenjima. U prvom poglavlju prikazan je matematički model nelinearnog programiranja. Drugim poglavljem dan je kratak pregled klasičnih numeričkih metoda i sistematizacija kroz shematski prikaz. Trećim poglavljem detaljno su obrađene osnove jednostavnog genetskog algoritma, princip rada i prikaz izvedbe. Također je dotaknuta i tema prilagodljivih i paralelnih genetskih algoritama. Zatim je detaljno objašnjena izvedba genetskih algoritama u programskom paketu MATLAB. Četvrto poglavlje posvećeno je primjeni genetskih algoritama u strojarskoj praksi. Odabran je primjer optimizacije parametara procesa obrade metala glodanjem, u cilju postizanja minimalne hrapavosti površine. Petim poglavljem objašnjena je prednost genetskih algoritama u odnosu na klasične optimizacijske metode. Šestim poglavljem ukazano je na ograničenja genetskih algoritama. Na kraju je dan zaključak u sedmom poglavlju.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

Izjava i zahvale

Izjavljujem da sam ovaj rad izradio samostalno uz preporuke i savjete svog mentora prof. dr. sc. Nedeljka Štefanića, kojem se ovom prilikom zahvaljujem, te uz korištenje prostorija i računala na FSB u Zagrebu. Svaki dio ovog DIPLOMSKOG RADA smije se javno objaviti.

UVOD

Nakon završetka preddiplomskog studija strojarstva, smjera Industrijsko inženjerstvo i menadžment, na FSB-u sam nastavio studiranje na diplomskom studiju, također istog usmjerenja. Studirajući na preddiplomskom studiju susreo sam se sa osnovama strojarstva, odnosno proizvodnim tehnologijama u strojarstvu, zatim primjenom računala, te matematike i fizike u inženjerstvu. Traženje najboljeg rješenja proizvodnog ili nekog drugog procesa u odnosu na raspoloživa sredstva smatram jako korisnim i dobrim, uz uvjet da se ne našteti čovjeku i njegovoj okolini. Na diplomskom studiju susrećem se sa stručnim kolegijima na kojima se detaljno obrađuje ta problematika. Od tih kolegija izdvojio bih Operacijska istraživanja I i II, na kojima se izučava kako prema matematički postavljenim optimizacijskim problemima i zadanim ograničenjima pronaći optimalna rješenja. Zatim kolegij Multivarijatne statističke metode na kojem se izučava kako statističkim metodama dobiti matematički model promatranih problema u strojarskoj (ili nekoj drugoj) praksi. Ova područja je zanimljivo za istraživati kako meni osobno tako i uopćeno. Područje nelinearnog programiranja u sklopu kolegija Operacijska istraživanja II, a posebno upotreba heurističkih metoda, poput genetskih algoritama, odabrao sam za svoj DIPLOMSKI RAD. Genetski algoritmi su jedna od metoda koje se koriste za pronalaženje optimuma nelinearnih funkcija cilja. Rade na principu prihvaćanja najboljih, a odbacivanja onih lošijih rješenja u svakoj sljedećoj fazi (koraku) izvedbe algoritma. Genetski algoritam je razvijen na principima teorije evolucije. Različite vrijednosti varijabli funkcije cilja smatraju se jedinkama. Pripadajući geni jedinke koji ju isključivo određuju su binarne znamenke (0 i 1), njihovim kombiniranjem mijenjaju se vrijednosti varijabli.

1.0. MATEMATIČKI MODEL PROBLEMA NELINEARNOG PROGRAMIRANJA

1.1. Problem matematičkog programiranja [1]

Matematičko programiranje je disciplina koja je razvijena sa svrhom pronalaženja ekstremnih vrijednosti (minimum ili maksimum) funkcije uz zadana ograničenja. Funkcija čiji ekstrem se traži naziva se funkcija cilja, dok su ograničenja također funkcije ali se ne traži njihov ekstrem već one ograničavaju prostor mogućih rješenja zadanog problema. Sve funkcije zajedno predstavljaju matematički model određenog slučaja u praksi. Tako funkcija cilja može biti profit, troškovi ili nešto drugo u promatranoj proizvodnji. Ograničenja mogu biti određeni tehnološki (npr. normativi vremena), tržišni (npr. cijena jedinice proizvoda) i drugi uvjeti. Ako su spomenute funkcije ili samo neke od njih nelinearne to je problem nelinearnog programiranja.

1.1.1. Problem nelinearnog programiranja [1]

Matematički model problema nelinearnog programiranja može se zapisati na slijedeći način:

1) Minimizacija funkcije cilja

$$\min \quad f(\mathbf{x}) \qquad (1.1)$$

uz ograničenja

$$g_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, m \qquad (1.2)$$

$$h_j(\mathbf{x}) \le 0, \quad j = 1, 2, \ldots, p \qquad (1.3)$$

te uz uvjet o nenegativnosti varijabli svih funkcija u matematičkom modelu

$$\mathbf{x} \ge \mathbf{0} \ . \qquad (1.4)$$

Argument ovih funkcija je vektor **x** koji sadrži *n* varijabli: $x_1, x_2, \ldots, x_n$. Izraz (1.1) predstavlja funkciju cilja s naznakom traženja minimuma. Izraz (1.2) predstavlja *m* ograničenja izraženih u obliku jednadžbi, dok izraz (1.3) predstavlja *p* ograničenja izraženih u obliku nejednadžbi. Sve tri funkcije pretvaraju *n*-dimenzionalni prostor u jedan realni broj. Ako se dakle radi o problemu nelinearnog programiranja neke ili sve od tih funkcija moraju biti nelinearne. Moguće rješenje ovog problema je skup *S* koji je određen na slijedeći način:

$$S = \left\{ \mathbf{x} \middle| g(\mathbf{x}) = \mathbf{0}, \ h(\mathbf{x}) \le \mathbf{0}, \ \mathbf{x} \ge \mathbf{0} \right\} \qquad (1.5)$$

gdje su $g: R^n \mapsto R^m$ i h: $R^n \mapsto R^p$ vektorske funkcije, moguće rješenje problema nelinearnog programiranja. Globalni minimum funkcije cilja na skupu S biti će neka točka **x*** iz tog skupa

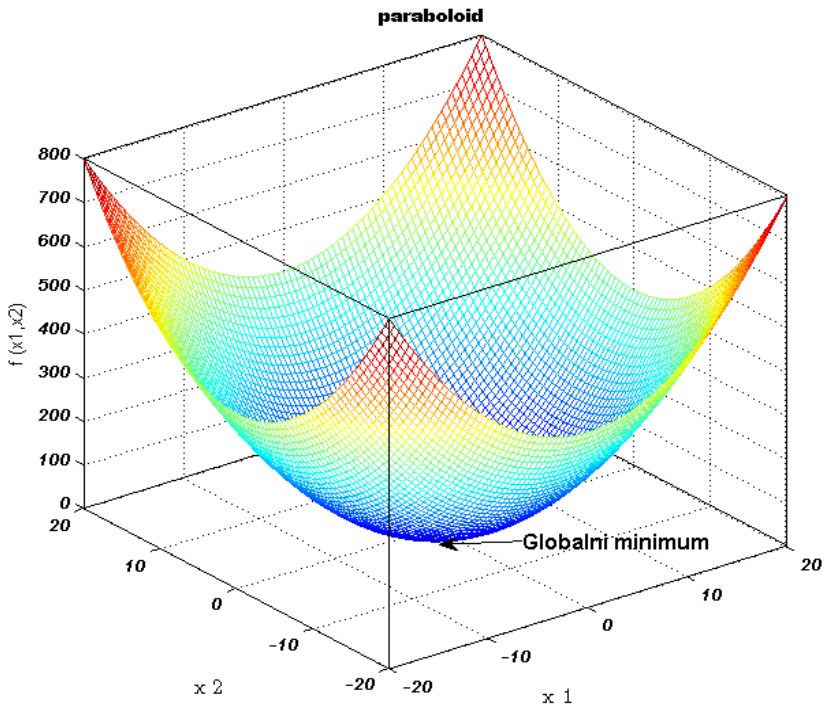
(ako takva točka postoji) sa slijedećim svojstvom:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in S \qquad (1.6)$$

Ako se promatra samo okolina točke **x*** tada se radi o lokalnom minimumu i vrijedi:

$$f(\mathbf{x}^*) \leq f(\mathbf{x} + \alpha\, \mathbf{s}) \qquad (1.7)$$

gdje je α po volji malen realni broj, a **s** predstavlja vektor smjera u *n*-dimenzijskom prostoru, odnosno koje se varijable x_i od mogućih *n* i u kojem smjeru mijenjaju za malen realni broj. Ako u izrazu (1.6) i (1.7) stoji znak < (bez dopuštenja jednakosti) radi se slijedom o strogom globalnom minimumu i strogim lokalnim minimumima (kojih može biti više na skupu S). Izrazi se mogu lakše shvatiti ako se promatra funkcija u prostoru, npr. paraboloid: $z = x_1^2 + x_2^2$ koji ima minimum u točki (0,0), što je prikazano na slici 1.1.



Slika 1.1. Globalni minimum paraboloida

2) Maksimizacija funkcije cilja

$$\max \quad f(\mathbf{x}) \qquad (1.8)$$

uz isti oblik ograničenja kao u (1.2) i (1.3), te uvjeta o nenegativnosti varijabli kao u (1.4) dobije se skup *S* koji također ovisi o postavljenim uvjetima. Globalni maksimum funkcije cilja na skupu *S* će biti točka **x*** iz istog skupa (ako takva točka postoji) sa svojstvom:

$$f(\mathbf{x}^*) \geq f(\mathbf{x}), \quad \forall \mathbf{x} \in S \qquad (1.9)$$

Ako se promatra samo okolina točke **x*** tada se radi o lokalnom maksimumu i vrijedi:

$$f(\mathbf{x}^*) \geq f(\mathbf{x} + \alpha\, \mathbf{s}) \qquad (1.10)$$

gdje α i **s** imaju iste karakteristike kao za slučaj minimizacije funkcije cilja. Također ako u izrazu (1.9) i (1.10) stoji znak > (bez dopuštenja jednakosti) radi se slijedom o strogom globalnom maksimumu i o strogim lokalnim maksimumima na skupu *S*.

2.0. SISTEMATIZACIJA NUMERIČKIH METODA OPTIMIRANJA NELINEARNIH FUNKCIJA CILJA [2]

Ovim poglavljem DIPLOMSKOG RADA prikazati će se osnovna sistematizacija numeričkih metoda za optimizaciju nelinearnih funkcija cilja. Detaljan uvid u način rada ovih metoda, korake algoritma, korištene izraze i drugo, dan je u literaturi.

2.1. Metode nelinearnog programiranja za funkcije jedne varijable

Newtonova metoda. Potrebno je odabrati prvu točku iz područja definicije, intuitivno ili slučajno. Ta točka može, ali i ne mora biti blizu optimuma. Ako se pogodi prava točka algoritam brzo konvergira prema optimumu. Također je za ovu metodu bitno da funkcija cilja bude derivabilna i neprekidna na području definicije. Metoda se koristi za optimizacijske probleme bez ograničenja. Metoda također ije prikladna za upotrebu kada je funkcija cilja većeg stupnja nelinearnosti, zbog određivanja derivacija i aproksimacijskog (Taylorovog) polinoma.

Metoda aproksimacije polinomom. Optimum funkcije cilja dobiva se aproksimacijom iste sa polinomom drugog ili trećeg stupnja. Postupno se tijekom koraka smanjuje interval pretraživanja i konvergira prema optimumu.

Metoda uniformnog traženja. Metoda radi na principu podjele početno odabranog intervala na *n* jednakih dijelova. Tom podjelom dobivaju se točke za koje se ispituju vrijednosti funkcije cilja. Ona točka za koju se dobije minimalna vrijednost funkcije cilja predstavlja potencijalni optimum, odnosno potrebno je još dodatno ispitati interval oko nje. Metoda zahtijeva kvazi konveksnost funkcije cilja.

Metoda račvanja. Također se razmatraju kvazi konveksne funkcije cilja na odabranom intervalu, kao kod metode uniformnog traženja.

2.2. Metode nelinearnog programiranja za funkcije više varijabli

Metoda kaznenih funkcija. Pretražuje se područje definicije ograničeno zadanim skupom ograničenja u obliku $g_i(\mathbf{x}) = 0$, uz *i*=1, 2,…, *n*. Također se javlja potreba za derivabilnošću i neprekidnošću funkcije cilja jer se priroda ekstrema određuje pomoću derivacija, prva i druga derivacija.

Metoda najbržeg uspona. Kreće se od početnog, slučajno odabranog rješenja, vektora $\mathbf{x} = (x_1, x_2, \dots, x_n)$. U slijedećim koracima određuje se poboljšano rješenje. Također se javlja potreba za derivabilnošću i neprekidnošću funkcije cilja jer se upotrebljava gradijent.

Metoda najokomitijeg uspona. Metoda zahtijeva neprekidnost i derivabilnost funkcije cilja jer se upotrebljava gradijent kao pokazatelj u kojem smjeru je najokomitiji uspon (najstrmiji nagib). Kreće se od jedne početne točke, vektora $\mathbf{x} = (x_1, x_2, \dots, x_n)$, te se u slijedećim koracima pomoću gradijenta određuju bolja rješenja.

Metoda ograničenog gradijenta. Koristi se za traženje optimuma nelinearnih funkcija cilja sa zadanim ograničenjima u obliku $g_i(\mathbf{x}) \leq 0$, uz *i*=1, 2,…, *n*. Također se kreće od početne, slučajno odabrane točke područja mogućih rješenja. Funkcija cilja treba biti derivabilna i neprekidna.

Metoda konveksnih kombinacija. Spada u skupinu gradijentnih metoda, dakle koriste se derivacije i funkcija cilja mora biti neprekidna i derivabilna. Metoda se koristi kad je uz nelinearnu funkciju cilja i skup ograničenja dan nelinearnim vezama u obliku $g_i(\mathbf{x}) \leq 0$.

Newton-Raphsonova metoda. Metoda koristi Taylorov red za potrebe aproksimacije točke optimuma funkcije cilja.

Kvazi-Newtonova, Fletcher-Powel-Davidonova i Fletcher-Reevesova metoda. Sve tri spomenute metode koriste konjugirane vektore smjerova i pretražuju *n*-dimenzionalne kvadratne funkcije cilja. *Fletcher-Reevesova* metoda koristi samo prvu derivaciju zato pripada metodama prvog reda. Metode su osjetljive na izbor duljine koraka algoritma.

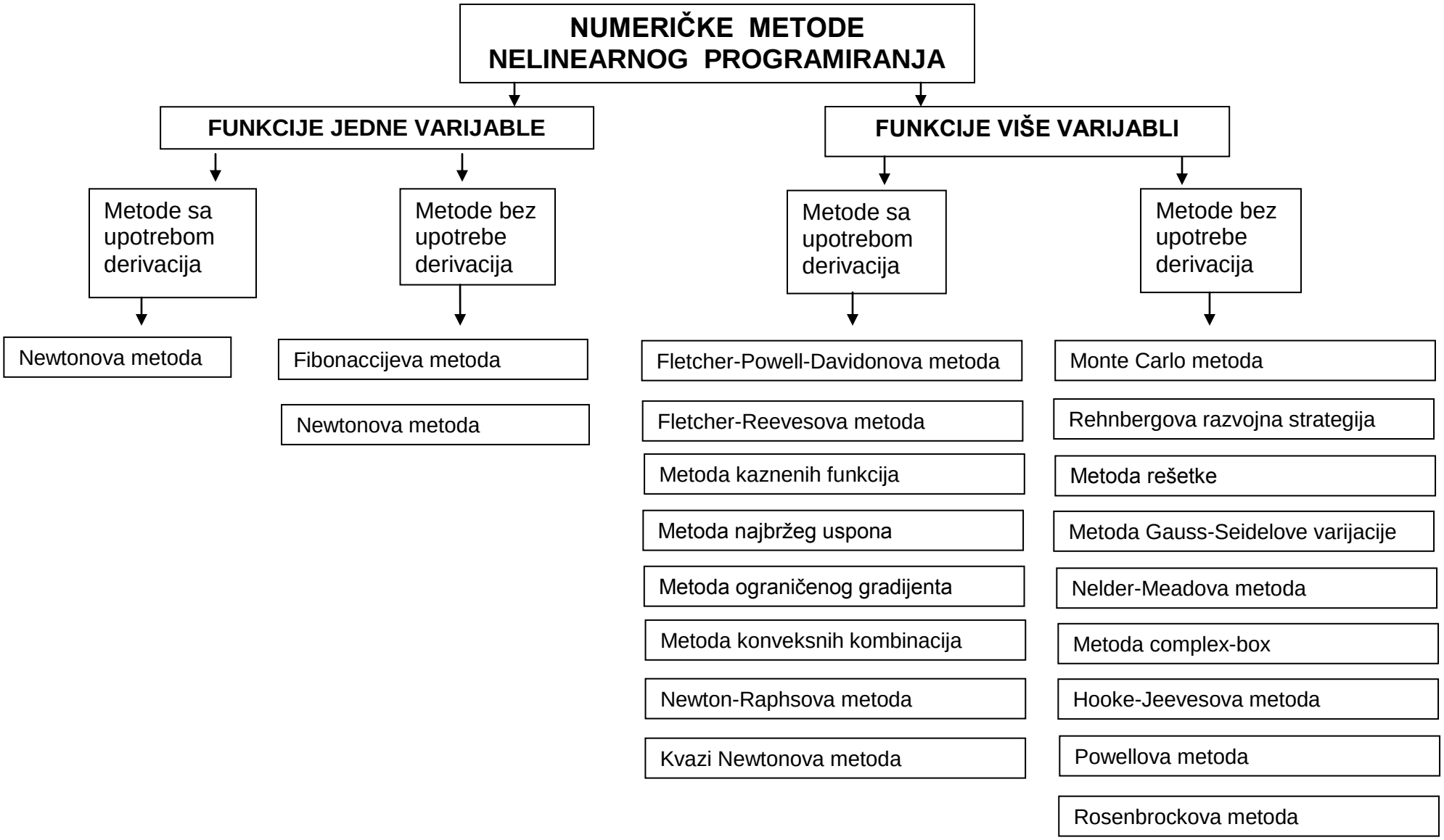
Monte Carlo metoda. Metoda radi na principu slučajnog pretraživanja vrijednosti funkcije cilja na području definicije, koje može, ali i ne biti ograničeno. Iz više slučajno dobivenih vrijednosti funkcije moguće ju je aproksimirati i tako odrediti ekstrem.

Rehnbergova razvojna strategija. Metoda se odvija na principu prethodnik sljedbenik. Sljedbenik uvijek ima veću vrijednost funkcije cilja od prethodnika. Ako to nije slučaj onda se ona sljedbenik sa manjom vrijednošću stavlja kao prethodnik. Kad se dostigne stanje da su nekoliko uzastopnih sljedbenika različiti u vrijednostima funkcije cilja unutar propisanih granica (tolerancija) algoritam se zaustavlja.

Metoda rešetke. Ova metoda je determinističkog karaktera i rješenja se pretražuju simultano. Duljina i smjerovi koraka su unaprijed zadani. Pokušaji pronalaska boljih rješenja se izvode istodobno (simultano), te se i usporedba rezultata može izvršiti odjednom.

Hooke-Jeevesova metoda. Metoda u kojoj se koraci algoritma mogu ubrzavati. Rješenja se neposredno pretražuju, na temelju pokušaja i pogrešaka, u četiri smjera. Pogreška nastupa kada slijedeće rješenje nije bolje od prethodnog, tada se vraća u prethodnu točku i izabire drugi smjer pretrage. Ako su rješenja uzastopno bolja jedna od drugog ubrzava se korak algoritma.

Iz prikazanog pregleda numeričkih metoda vidljivo je da se one razlikuju prema tome koliko varijabli ima funkcija cilja, da li koriste ili ne koriste derivacije, te da li su predviđene za rad bez ili sa ograničenjima. U nastavku na slikama 2.1, 2.2, i 2.3 je prikazano nekoliko shematskih podjela numeričkih metoda.



Slika 2.1. Shema podjele numeričkih metoda nelinearnog programiranja

3.0. OSNOVE GENETSKIH ALGORITAMA [3]

3.1. Način rada jednostavnog genetskog algoritma

Može se pretpostaviti samo problem maksimizacije. Ako se radi o problemu minimizacije neke funkcije *f* tada je potrebno uvesti novu funkciju *g = -f,* te je maksimizacija funkcije *g* ekvivalentna minimizaciji funkcije *f* :

$$\min \ f(\mathbf{x}) = \max \ g(\mathbf{x}) = \max \{- f(\mathbf{x})\} \qquad (3.1)$$

također se pretpostavlja da funkcija *g* poprima pozitivne vrijednosti na području definicije (cijeloj domeni), ako to nije slučaj dodaje se pozitivna konstanta *C.*

Neka funkcija kojoj se želi naći maksimum ima svojstvo $f(x_1,..., x_k) : R^k \mapsto R$. Argument te funkcije je dakle vektor **x** sa *k* varijabli i svaka od tih varijabli x_i može poprimiti vrijednosti iz $D_i = [A_i, B_i] \subseteq R$. Također je $f(x_1,..., x_i,..., x_k)$ >0 za sve $x_i \in D_i$.

Jednostavni genetski algoritam će raditi sa varijablama x_i funkcije *f* samo ako ih se prevede u za njega prikladan zapis realnih brojeva, a to je zapis u binarnom sustavu.

3.1.1. Binarni zapis prirodnog broja *N*

Binarni zapis broja koristi samo dvije znamenke, 0 i 1 kojih može biti po volji mnogo i na različitim mjestima (pozicijama) u zapisu. Zapis prirodnog dekadskog broja *N* (uključujući i nulu) u binarnom sustavu (koristi se baza 2) i njegovo izračunavanje, odnosno prevođenje u dekadski sustav (koristi se baza 10) izvodi se prema izrazu (2.2):

$$N = a_n a_{n-1} ... a_1 a_0 (2) = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + ... + a_1 \cdot 2 + a_0 \qquad (3.2)$$

uz *n* = broj znamenki – 1. Znamenke $a_n, a_{n-1},..., a_1, a_0$ su cijeli brojevi iz skupa $\{0,1\}$.

<u>Primjer 3.1.</u>: Binarni zapis s osam znamenki broja 147 je u obliku 10010011(2), a broja 2 s istom duljinom zapisa je u obliku 00000010, što se računa kako slijedi:

10010011(2) = $1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 147$,

00000010(2) = $0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 2$.

Vidljivo je da je maksimalni eksponent ovih binarnih brojeva *n* =7. Također prvi zapis ima prvu znamenku 1, što znači da će prirodni broj kojeg predstavlja taj zapis odmah biti određen kao

velik jer potencija 2^7 ne će biti neutralizirana množenjem sa nula. Drugi zapis ima prvih šest znamenki 0, što znači da se prvih šest potencija neutralizira i taj broj je određen kao malen. Ako se promatra interval kojeg može pokriti binarni zapis sa osam znamenki tada je binarni broj 00000000(2) = 0 početak intervala, a broj 11111111(2) = 255 kraj intervala. Obuhvaćeno je 2^8 =256 brojeva, tj. skup $\{0,..,255\}$.

Odavde slijedi zaključak da veća duljina zapisa broja (veći *n*) pokriva i veći interval, odnosno može se zapisati više prirodnih dekadskih brojeva (uključujući nulu). Također i da razmještaj jedinica i nula određuje veličinu dekadskog broja.

Primjer 3.2.: Slijedeće pitanje koje se postavlja je npr.: kako u binarnom sustavu zapisati realni broj 147,078 koji se nalazi unutar intervala [100, 200]?.

Broj 147,078 ima tri decimale, dakle najveća zahtijevana preciznost za prikaz tog broja je na treću decimalu. Odavde slijedi da između dva broja 147 i 148 postoji 999 brojeva spomenute preciznosti, a to su brojevi iz skupa $\{147{,}001;147{,}002;\ldots;147{,}999\}$.

Ako se u promatranom intervalu želi doći od broja 147,000 do broja 148,000 potrebno je 1000 brojeva (147,000 + 1,000 = 148,000), ili tisuću jednakih (ekvidistantnih) duljina. Ne uključuje se broj 147,000, a uključuje se broj 148,000.

Ovim primjerom je dan uvid u način rješavanja ovog problema, a za potpuno rješenje (zapis spomenutog broja u binarnom sustavu) potrebno je odrediti kako binarni zapis prirodnog dekadskog broja *N* (uključujući i nulu) određuje smještaj realnog broja na promatranom intervalu.

3.1.2. Potrebna duljina binarnog zapisa – potreban broj znamenaka

Iz primjera 3.2. se može zaključiti da o zahtijevanoj preciznosti varijabli x_i na D_i, te o duljini same domene D_i na kojoj se traži rješenje ovisi i veličina skupa prirodnih dekadskih brojeva (proširenog sa nulom) koji je potreban za prikaz promatranog intervala, odnosno ovisi duljina binarnog zapisa (zaključak iz primjera 3.1.). Potreban broj mjesta (ili duljina) binarnog zapisa za prikaz promatranog intervala realnih brojeva određene preciznosti određuje se prema izrazu (3.3):

$$(B_i - A_i) \cdot d \leq 2^{m_i}, \quad i = 1, 2, \ldots, k \qquad (3.3)$$

B_i je kraj, a A_i početak intervala (domene D_i); *d* je zahtijevana preciznost, odnosno veličina skupa brojeva (ili broj ekvidistantnih duljina) kojima se prikazuje interval $(A_i, B_i]$; m_i je minimalni

eksponent koji zadovoljava postavljeni uvjet (3.3) za varijablu x_i i domenu D_i; brojač i = 1, 2,…,k je indeks promatrane varijable.

3.1.3. Dekodiranje binarnog zapisa

Binarni zapis prirodnog dekadskog broja (i nule) određuje mjesto realnog broja na promatranom intervalu točno u željenu preciznost samo ako se izvrši pravilno prevođenje ili dekodiranje. Dekodiranje se vrši prema izrazu:

$$x_i = A_i + decimal\ (a_n a_{n-1} ... a_1 a_{0(2)}) \cdot \frac{B_i - A_i}{2^{m_i} - 1} \qquad (3.4)$$

gdje je $decimal\ (a_n a_{n-1} ... a_1 a_{0(2)})$ dekadski (decimalni) zapis binarnog broja u zagradi i izračunava se prema izrazu (3.2). $\frac{B_i - A_i}{2^{m_i} - 1}$ je veličina jednakih (ekvidistantnih) dijelova i proizlazi iz podjele promatrane domene D_i na onoliko jednakih dijelova koliko zahtijeva željena preciznost realnih brojeva i duljina same domene.

Primjer 3.3.: Sada je moguće riješiti problem binarnog zapisa realnog broja 147,078 unutar promatranog intervala (ili domene) [100, 200]. Preciznost realnog broja je na treću decimalu, a duljina intervala je 100 cijelih brojeva. Svaki cijeli broj dakle ima po 1000 osnovnih jedinica (jednakih dijelova) intervala. Stoga je za prikaz promatranog intervala sveukupno potrebno 100x1000=100 000 jednakih dijelova (brojeva). Sada je potrebno odrediti duljinu (ili broj mjesta) binarnog zapisa. Prema izrazu (3.3) je:

$$2^{16} = 65\ 536 < (200 - 100) \cdot 1000 = 100\ 000 < 2^{17} = 131\ 072$$

i uvjet je zadovoljen tek za 2^{17}. Dakle binarni zapis imati će 17 mjesta što daje mogućnost prikaza 131 072 broja (varijacije binarnih brojeva). Kako je potrebno 100 000 brojeva onih 31 072 su višak. To je neizbježno jer eksponent *m* mora biti cijeli broj jer je potreban i cijeli broj mjesta za upis znamenaka 0 i 1. Ovaj višak zapravo znači da će ponekad dva različita zapisa pripadati istom ekvidistantnom dijelu u intervalu. Ali ne mora se strogo ovako gledati. Može se promatrana domena podijeliti i na 131 072 dijela što znači da preciznost svih brojeva ne će biti na treću decimalu nego će približno svaki četvrti davati preciznost na četvrtu decimalu npr. za razlučivost 0,0005 (za pola treće decimale više). Također se može realni broj zapisati sa npr. šest decimala ali će njegova stvarna preciznost biti na treću decimalu. Time se svjesno radi pogreška. Binarni zapis broja 147,078 nakon određivanja eksponenta *m* određuje se prema izrazu (3.4):

$$decimal \quad (a_{16}a_{15}...a_1a_{0(2)}) = N = \frac{(x_i - A_i)\cdot(2^m - 1)}{B_i - A_i} = \frac{(147,078 - 100)\cdot 131071}{200 - 100} \cong 61\ 706 \ .$$

Dobiven je broj u dekadskom sustavu (prirodan broj) koji određuje traženi realni broj u promatranom intervalu. Ako je interval podijeljen na 131 072 jednaka (ekvidistantna) dijela, broj 147,078 pomaknut je za skoro cijelih 61 706 jednakih dijelova od početnog broja 100. Dok je za cijelih 61 706 jednakih dijelova od početka intervala pomaknut broj 147,0783 koji ima više izraženu preciznost (do četvrte decimale). Broj se mogao zapisati i do dvanaeste decimale ali to nije njegova stvarna preciznost. Binarni zapis broja *N* se može lako pogoditi zbrajajući odgovarajuće potencije baze 2 i izgleda kako slijedi:

$$0111100010\ \ 0001010_{(2)} \Rightarrow decimal\ (0111100010\ \ 0001010) = 2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{8} + 2^{3} + 2 = 61706$$

Zaključak iz ovog primjera je da vrijednost (ili veličina) dekadskog broja koji se dobije iz oblika binarnog broja utječe na smještaj (poziciju ili vrijednost) realnog broja unutar promatranog intervala (ili domene), što je određeno dekodiranjem.

Primjerom 3.3. je prikazano traženje binarnog zapisa realnog broja. Takvi oblici zapisa brojeva su prikladni za pretraživanje i način na koji radi genetski algoritam. U genetskim algoritmima prvo se zadaju binarni zapisi brojeva dok se sami realni brojevi, koji su zapravo argumenti funkcije, dobivaju dekodiranjem. Također je ovim primjerom promatrana samo jedna varijabla dok funkcije cilja obično imaju više varijabli.

3.1.4. Kromosom i populacija

Ukupna duljina binarnog zapisa za sve varijable vektora **x**: $x_1, x_2, ..., x_i, ..., x_k$ koje se nalaze u promatranim intervalima, odnosno domenama: $D_1, D_2,..., D_i,..., D_k$ jednaka je zbroju potrebnih mjesta binarnih zapisa brojeva za prikaz svih *k* domena. Ovakav ukupan binarni zapis se naziva kromosom. Dakle znamenke imaju sličnu ulogu kao geni jer vrsta znamenke i smještaj određuju vrijednost realnih brojeva, odnosno smještaj unutar promatranih intervala. Kada se ukupni binarni zapisi odrede (dekodiraju) u realne brojeve određene su i sve varijable funkcije, te se izračunavaju vrijednosti funkcije u točkama. Algoritmom se jednostavno izmjenjuju nule i jedinice u ukupnom binarnom zapisu broja na različite pozicije, te se tako dobivaju i različite vrijednosti funkcija. Tako se ispituju (ocjenjuju ili evaluiraju) rješenja i traže nova koja rješavaju zadani problem.

Više ukupnih binarnih zapisa brojeva (ili više kromosoma) zajedno čine skup koji se naziva populacija. Tako je početni skup binarnih brojeva ili kromosoma početna populacija i može biti proizvoljno velik u rasponu od 1 do *pop_size*. Daljnji odabir kromosoma vrši se na temelju ocjenjivanja (evaluacije) svih kromosoma u populaciji. Ocjenjivanje se vrši funkcijom

$eval(v_i)$ koja izračunava vrijednost funkcije za kromosom v_i nakon dekodiranja ukupnog binarnog zapisa broja (izračunavanja varijabli funkcije). Algoritam postupka odabira kromosoma je slijedeći:

- Izračunati vrijednost funkcije $f(\mathbf{x})$, odnosno $eval(v_i)$ za sve odabrane kromosome u populaciji $v_i (i = 1,..., pop\_size)$.
- Odrediti ukupnu vrijednost ili jačinu populacije (ili dobrotu funkcije cilja):
  $F = \sum_{i=1}^{pop\_size} eval(v_i)$.
- Izračunati vjerojatnost odabira p_i svakog kromosoma $v_i (i = 1,..., pop\_size)$:
  $p_i = eval(v_i) / F$.
- Izračunati kumulativnu vjerojatnost q_i za svaki kromosom $v_i (i = 1,..., pop\_size)$:
  $q_i = \sum_{j=1}^{i} p_j$.

Nakon što se izračunaju vjerojatnosti odabira za svaki kromosom i slože kumulativne vjerojatnosti, te tako dobiju intervali vjerojatnosti za svaki kromosom pojedinačno pridružuju im se slučajni brojevi. Odavde slijedi da će neki kromosomi biti odabrani nekoliko puta, a neki ni jednom što je izravna posljedica vjerojatnosti odabira kromosoma. Postupak nalikuje na okretanje kola ruleta *pop_size* puta koje ima toliko utora koliko ima kromosoma u populaciji, ali su neki utori širi, a neki uži, pa će kuglica lakše pasti u šire utore. Postupak je slijedeći:

- Ispisati (generirati) slučajne brojeve *r* u rasponu [0…1] *pop_size* puta.
- Ako je $r < q_1$ odabrati prvi kromosom v_1, ako je drukčije odabrati *i*-ti kromosom v_i $(2 \le i \le pop\_size)$ takav da vrijedi $q_{i-1} < r \le q_i$.

3.1.5. Osnovne operacije jednostavnog genetskog algoritma

3.1.5.1. Križanje

Jedna od osnovnih operacija u jednostavnom genetskom algoritmu je **operacija križanja**. Križanjem dva kromosoma nastaju novi kromosomi koji imaju gene oba. Tako se i znamenke dvaju binarnih brojeva mogu križati i nastaju novi binarni brojevi koji imaju znamenke jednog i drugog. Razmještaj znamenki u binarnom broju određuje veličinu dekadskog broja, pomoću kojeg se dekodiranjem određuje smještaj realnog broja unutar promatranog intervala (zaključak iz prethodnih primjera). Slijedi da će nakon križanja ukupnih binarnih brojeva nastajati

novi realni brojevi različiti od prethodnih, odnosno mijenjati će se vrijednosti varijabli $x_1, x_2, ..., x_i, ..., x_k$ unutar domena $D_1, D_2, ..., D_i, ..., D_k$.

Za operaciju križanja određuje se točka križanja *pos* koja predstavlja slučajni broj unutar raspona [1…*m* –1] (*m* je ukupna duljina binarnog broja, ili ukupan broj znamenki).
Operacija križanja se izvodi tako da su dva kromosoma

$(a_1 a_2 ... a_{pos}\, a_{pos+1} ... a_m)$ i $(b_1 b_2 ... b_{pos}\, b_{pos+1} ... b_m)$,

gdje su a_{pos} i b_{pos} točke križanja kromosoma, nakon operacije zamijenjeni sa izdancima

$(a_1 a_2 ... a_{pos}\, b_{pos+1} ... b_m)$ i $(b_1 b_2 ... b_{pos}\, a_{pos+1} ... a_m)$.

Razmještanjem nekoliko prvih znamenki binarnog broja određuje se «makro smještaj» (ili pozicija) realnog broja (vrijednost varijable x_i). Razmještanjem zadnjih nekoliko znamenki binarnog broja određuje se «mikro smještaj» ili preciznost realnog broja. Tako će nakon operacije križanja novonastali binarni brojevi (izdanci), ovisno o tome gdje je točka križanja, davati vrijednosti različite od prvotnih brojeva (roditelja). Te vrijednosti biti će mnogo različite ako se promijene uglavnom prve, a manje različite ako se promijene uglavnom posljednje znamenke.

<u>*Primjer 3.4.*</u>: U intervalu [100, 200] broj 197,100 je blizu kraja promatranog intervala, a broj 100,737 je blizu početka intervala. Ovi realni brojevi imaju svoj zapis u binarnom sustavu, a to je:

$197{,}100 = 1111100010\quad 0100110_{(2)}$ i

$100{,}737 = 0000000111\quad 1000110_{(2)}$.

Nakon križanja ova dva broja od točke *pos* = 8, nastaju nova dva:

$197{,}222 = 11111000\quad \mathbf{111000110}_{(2)}$ i

$100{,}615 = 00000001\quad \mathbf{100100110}_{(2)}$.

Prvi izdanak, broj 197,222 ima «makro smještaj» kao broj 197,100 jer ima (je naslijedio) prvih osam znamenki iz binarnog zapisa tog broja (prvog roditelja). Drugi izdanak, broj 100,615 ima (je naslijedio) prvih osam znamenki binarnog zapisa broja 100,737 (drugog roditelja) pa u skladu s njim ima i približno jednak smještaj unutar intervala.

Parametar genetskog sustava koji određuje operaciju križanja je vjerojatnost križanja p_c. Tako umnožak $p_c \cdot pop\_size$ predstavlja broj kromosoma koji će biti odabrani za operaciju križanja. Postupak odabira izvodi se prema slijedećem postupku:

- Ispisati (generirati) slučajne brojeve *r* iz raspona [0…1] za svaki kromosom;

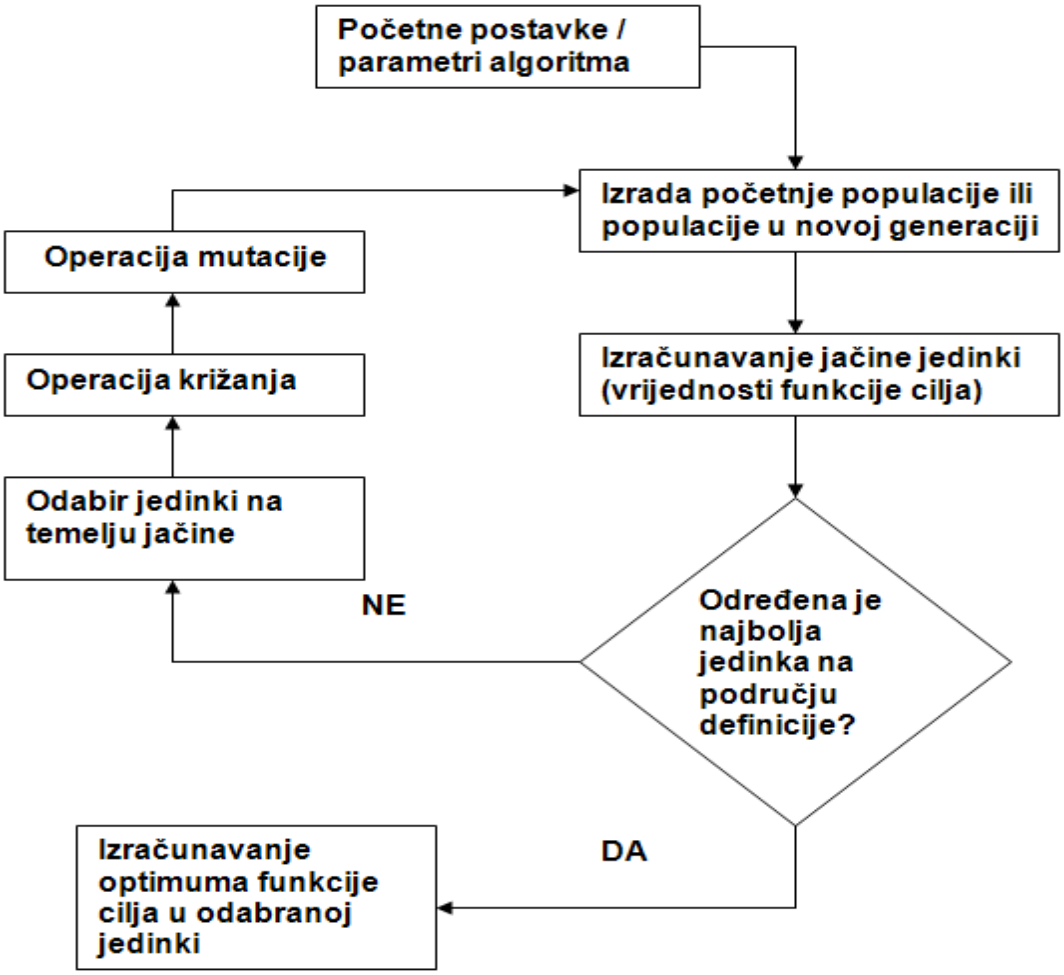
- Ako je $r < p_c$ odabrati spomenuti kromosom za križanje.

3.1.5.2. Mutacija

Slijedeća osnovna operacija u genetskom algoritmu je **operacija mutacije**. Mutacija je zapravo promjena jedne znamenke u drugu, sebi suprotnu jer binarni sustav ima samo dvije moguće znamenke. Parametar koji određuje operaciju mutacije je vjerojatnost mutacije, p_m i preko njega se određuje broj znamenki u ukupnoj populaciji koje mutiraju: $p_m \cdot m \cdot pop\_size$. Operacija mutacije obično se izvodi nakon križanja prema slijedećem postupku:

- Ispisati (generirati) slučajne brojeve *r* iz raspona [0…1];
- Ako je $r < p_m$ znamenka kojoj je pridružen slučajni broj *r* će mutirati.

Korisno je prikazati dijagram tijeka genetskog algoritma, slika 3.1.



Slika 3.1. Dijagram tijeka genetskog algoritma

3.2. Način rada prilagodljivih genetskih algoritama [4]

Prilagodljivi genetski algoritam (*AGA – Adaptive genetic algorithm*) namijenjen je također za rad u okruženju generacijskog genetskog algoritma. Odabiru se kromosomi (jedinke) početne populacije, koja predstavlja prvu generaciju, te se genetskim operatorima, križanjem i mutacijom, izraženim parametrima vjerojatnost križanja p_c i vjerojatnost mutacije p_m, u slijedećim generacijama mijenja populacija rješenja. Parametri genetskih operatora, p_c i p_m, također se mijenjaju ovisno o konvergenciji algoritma prema nekom optimumu (lokalnom ili globalnom. Izraz preko kojeg se može uočiti takva konvergencija algoritma je u obliku:

$$eval_{\max} - \overline{eval} \qquad (3.5)$$

Gdje je $\overline{eval} = \sum_{i=1}^{N} eval_i$, aritmetička sredina svih ocjena u populaciji (generaciji). Ako se razlika izražena izrazom (3.5) smanjuje tada algoritam konvergira prema nekom od optimuma, te je potrebno mijenjati parametre p_c i p_m. Također je potrebno sačuvati dobra rješenja u koracima algoritma. Čuvanje se može ostvariti smanjenjem parametara p_c i p_m za bolje jedinke, a povećanjem za slabije. Povećanjem spomenutih parametara povećava se dakle vjerojatnost križanja i mutacije jedinki, što mijenja njihov genetski materijal. To nije poželjno za jedinke koje daju dobra rješenja. Slabije jedinke služe za osiguranje od konvergencije algoritma prema lokalnom optimumu. Matematički izrazi za izračunavanje parametara genetskih operatora su:

$$p_c = \frac{k_1(eval_{\max} - eval')}{eval_{\max} - \overline{eval}} \text{ , uz } k_1 \leq 1.0 \qquad (3.6)$$

$$p_m = \frac{k_2(eval_{\max} - eval)}{eval_{\max} - \overline{eval}} \text{ , uz } k_2 \leq 1.0 \qquad (3.7)$$

gdje su *eval* dobrota (ocjena) kromosoma (vrijednost funkcije cilja) odabranog za mutaciju,
eval' dobrota (ocjena) boljeg od dvaju kromosoma odabranih za križanje.

Parametri p_c i p_m su jednaki nuli za najbolje rješenje, odnosno za elitne jedinke. To je u skladu sa operatorom u genetskom algoritmu koji se naziva elitizam. Taj operator ne dozvoljava ikakvu provedbu križanja i mutacije nad odabranim jedinkama koje se nazivaju elitne.

3.3. Paralelni genetski algoritmi [4]

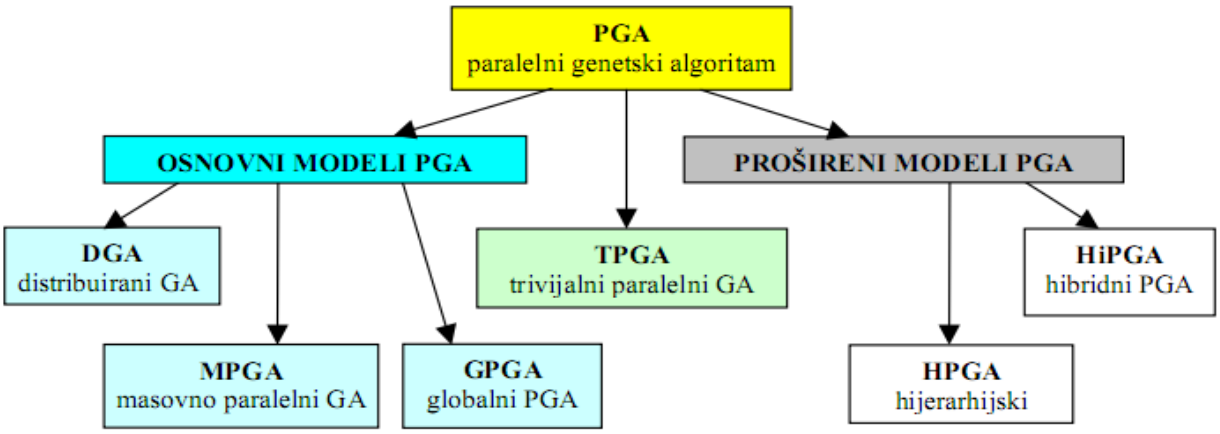
Paralelni genetski algoritmi koriste se za rješavanje težih optimizacijskih problema. Za njihovo izvođenje potrebna su računala sa više-jezgrenim procesorima ili umrežena računala. Cilj paraleliziranja je skraćenje vremena izvedbe algoritma, a ideja je zasnovana na sekvencijskom rješavanju pojedinih dijelova algoritma, odnosno podzadataka. Postoje dva pristupa paraleliziranja genetskih algoritama:

- Standardni pristup. Paralelizacija genetskih operatora i izračunavanje vrijednosti funkcije cilja paralelno. Paralelizira se samo posao evaluacije (izračunavanje vrijednosti funkcije cilja za odabrane jedinke). Model se naziva jedno-populacijski model.

- Dekompozicijski pristup. Podijeliti populaciju na manje dijelove (pod-populacije) i obavljati cijeli genetski algoritam nad njima. Model se naziva više-populacijski.

Paralelizacija genetskih algoritama je moguća na razini populacije, na razini jedinki,te na razini evaluacije. Podjela paralelnih genetskih algoritama prikazana je na slici 3.2. Prema razini paralelizacije postoje tri osnovna načina podjele na podzadatke:

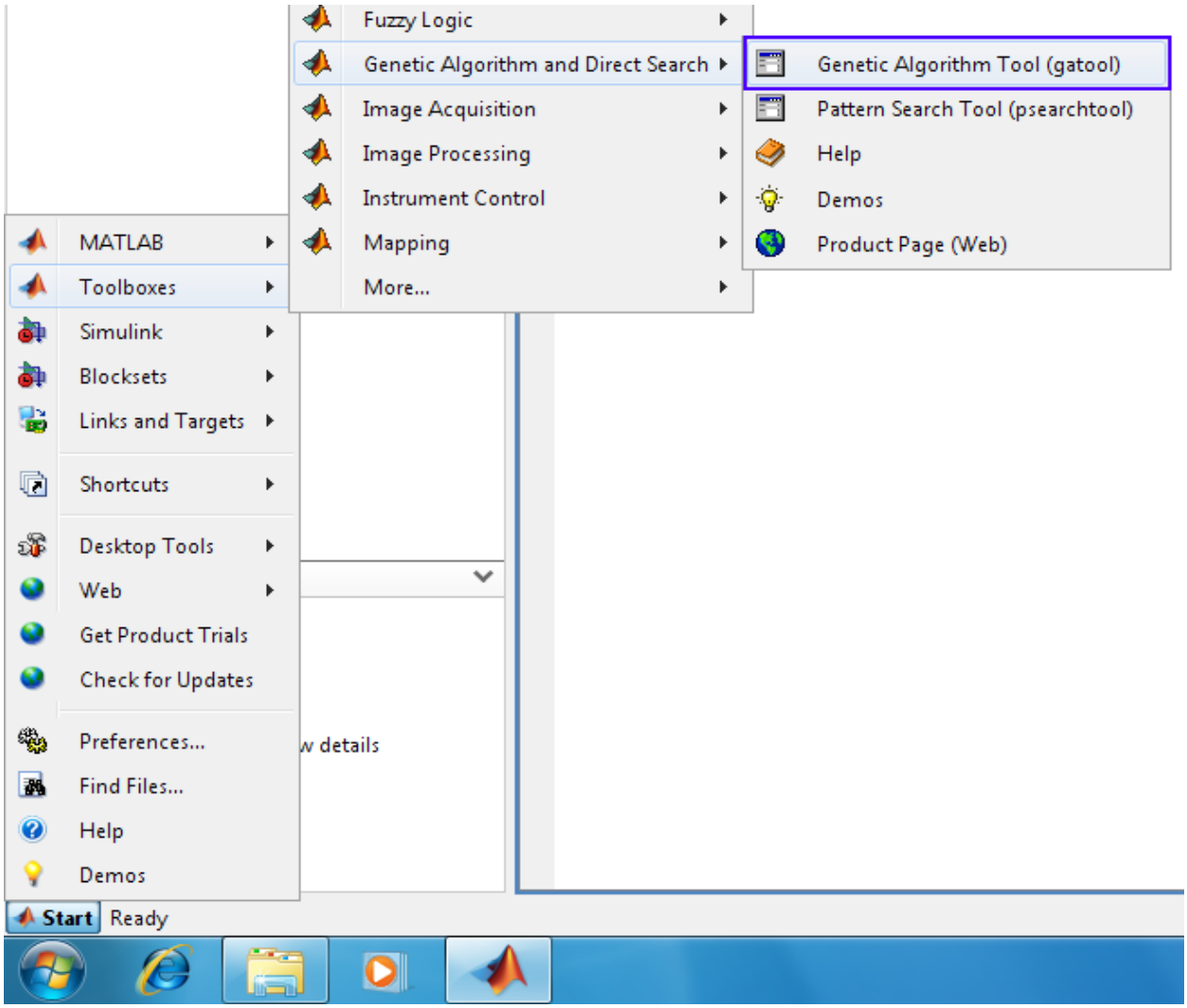
- Krupnozrnata podjela. Podjela velike populcije na manje dijelove pod-populacije. U ovom slučaju izvodi se više genetskih algoritama nad manjim pod-populacijama.
- Sitnozrnata podjela. Ekstremni oblik podjele velike populacije na manje dijelove, a to su jedinke. Svaki procesor (ili umreženo računalo) dobiva zadatak procesuiranja jedne jedinke. Na toj jedinki vrše se sve operacije koje ima genetski algoritam.
- Paralelna evaluacija. Vrijednosti funkcije cilja za svsku odabranu jedinku izračunavaju se paralelno dok se genetski operatori izvode sekvencijski. Ovakva podjela se naziva „podjela na gospodara i sluge“. U ovako postavljenoj izvedbi algoritma „sluge“ obavljaju evaluaciju nakon što „gospodar“ obavi svoj dio posla, a to je odabir jedinki, te operacije križanja i mutacije.



Slika 3.2. Podjela paralelnih genetskih algoritama

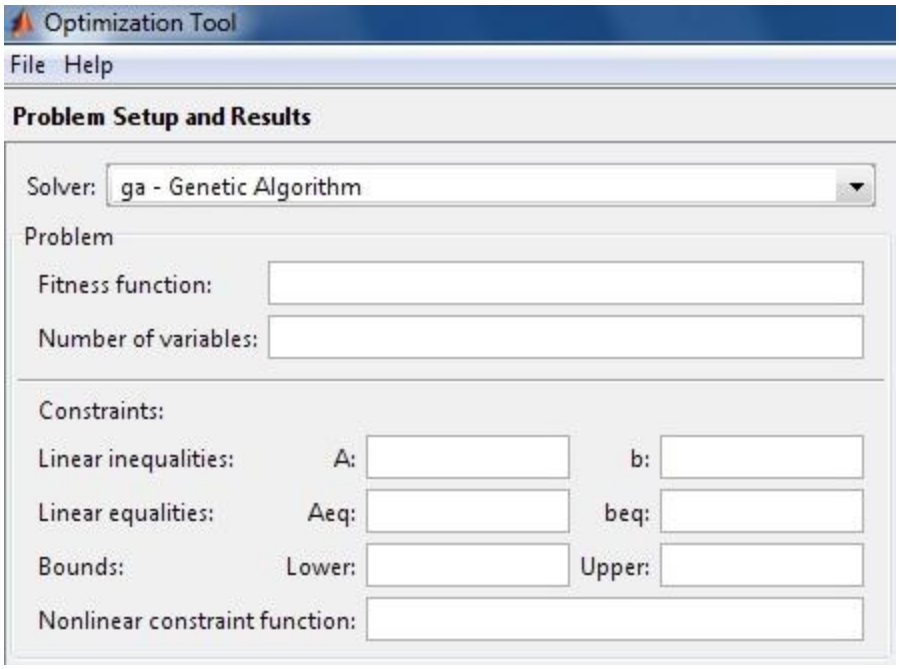
3.4. Izvođenje genetskog algoritma u MATLAB-u [5]

U MATLAB-u, matematičkom softveru za računanje s matricama, genetski algoritam se može izvoditi na dva načina. Prvi je izravnim upisom naredbi u prozor za naredbe (*Command window*), a drugi pomoću alata za optimizaciju (*Optimization Tool*) unutar aplikacije *Toolboxes*. Prvi način zahtijeva poznavanje programske sintakse kojom se pozivaju funkcije za izvođenje algoritma. Drugi način (*Toolboxes*) je prihvatljiviji za osobu koja se prvi put susreće sa MATLAB-om i ne poznaje dovoljno sintaksu. U ovom DIPLOMSKOM RADU prikazati će se izvođenje genetskog algoritma u alatu za optimizaciju. Pokretanje optimizacijskog alata moguće je upisom sintakse „optimtool“ u *Command window*, ili preko izbornika „Start“ u MATLABU, prikazano na slici 3.3.



Slika 3.3. Otvaranje alata za optimizaciju (*Optimization Tool*) u MATLAB-u

Optimization Tool prozor ima tri odjeljenja, a to su **Problem Setup and Results**, prikazan na slici 3.3, koji služi za postavljanje optimizacijskog problema i prikaz rezultata izvođenja algoritma. Zatim **Options**, prikazan na slici 3.4, koji služi za odabir postavki izvođenja algoritma. Dio **Quick Reference** služi kao podsjetnik, odnosno objašnjenje pojmova u *Optmization Tool* prozoru.



Slika 3.4. Dio *Optimization Tool* prozora: postavljanje optimizacijskog problema

3.4.1. Obrazac „Problem Setup and Results“ u *Optimization Tool* prozoru

Obrazac „**Problem Setup and Results**“ sadrži nekoliko dijelova koje je potrebno ispuniti. Prvi je *solver*, u kojem se odabire metoda kojom se želi riješiti optimizacijski problem. Odabire se: ga – Genetic Algorithm. U dijelu *Problem* potrebno je upisati koja funkcija cilja (*fitness function*) se promatra. Funkcija cilja se izrađuje (definira) u M-datoteci (M-file). Također je potrebno upisati broj varijabli funkcije cilja (*Number of variables*). U dijelu *Constraints* definiraju se ograničenja optimizacijskog problema. Ograničenja mogu biti linearna u obliku nejednadžbi (*Linear inequalities*) i jednadžbi (*Linear equalities*), te u obliku granica (*Bounds*). Linearna ograničenja definiraju se matrično, a opći oblik je prikazan u izrazima (3.1). Nelinearna ograničenja definiraju se slično kao i linearna, samo se u njihovom slučaju za rješavanje optimizacijskog problema koristi prošireni Lagrangeov genetski algoritam. Za upis nelinearnih ograničenja potrebno je izraditi M-datoteku. Opći oblik nelinearnih ograničenja prikazan je u izrazima (3.8).

$$\boldsymbol{A} \cdot \mathbf{x} \leq \boldsymbol{b} \qquad (3.8\ a)$$

$$\boldsymbol{A}_{eq} \cdot \mathbf{x} = \boldsymbol{b}_{eq} \qquad (3.8\ b)$$

$$\boldsymbol{dg} \leq \mathbf{x} \leq \boldsymbol{gg} \qquad (3.8\ c)$$

Gdje su **A**, **x**, **b**, **dg** (donja granica), **gg** (gornja granica) vektori i kad se jednadžba prikaže sa svim elementima vektora izgleda kako slijedi u izrazima (3.9), također su neposredno ispod svakog izraza prikazane i sintakse za unos u MATLAB. Paziti na razmak (ili zarez) koji označava upis slijedećeg elementa i točku-zarez koja označava prijelaz u novi red matrice.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \qquad (3.9\ a)$$

$$\begin{bmatrix} a_{(m+1)1} & a_{(m+1)2} & \dots & a_{(m+1)n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{mt\,1} & a_{mt\,2} & \dots & a_{mt\,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_{m+1} \\ b_2 \\ \vdots \\ b_{mt} \end{bmatrix} \qquad (3.9\ b)$$

$$\begin{bmatrix} dg_1 \\ dg_2 \\ \vdots \\ dg_n \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} gg_1 \\ gg_2 \\ \vdots \\ gg_n \end{bmatrix} \qquad (3.9\ c)$$

MATLAB sintaksa za matrice **A**, **b**, **dg** i **gg** je:

A: $[a_{11}\ \ a_{12}\ \ \dots\ \ a_{1n};\ a_{21}\ \ a_{22}\ \ \dots\ \ a_{2n};\ a_{m1}\ \ a_{m2}\ \ \dots\ \ a_{mn}]$,
b: $[b_1;\ b_2;\ \dots\ ;b_m]$,
dg (*Lower bounds*) : $[dg_1;\ \ dg_2;\ \ \dots\ ;dg_n]$,
gg (*Upper bounds*): $[gg_1;\ gg_2;\ \ \dots\ ;gg_n]$.

Nelinearna ograničenja u obliku nejednadžbi i jednadžbi izgledaju kako slijedi (MATLAB sintaksa upisuje se u M-datoteci, jer su nelinearna ograničenja funkcije):

$c_i(\mathbf{x}) \leq 0, \qquad i = 1, \dots, m$ (3.10 a)
$c_{eq\,i}(\mathbf{x}) = 0, \qquad i = m+1, \dots, mt$ (3.10 b)

gdje je *m* broj ograničenja u obliku nejednadžbi, a *mt* je ukupan broj ograničenja.

Potrebno je upisati ovakva ograničenja u obrazac na ispravan način, kako bi se izbjeglo krivo pretraživanje mogućih rješenja. To će biti prikazano na primjeru u točki 3.4., te 4. i 5. DIPLOMSKOG RADA.

Drugi dio obrasca, koji se zove **Results** prikazuje rezultate tijekom izvođenja i na kraju algoritma. Dio *Final point* prikazuje vrijednost varijabli (točke) funkcije cilja za koju se postiže optimum. Ovaj dio sa podacima biti će prikazan u točkama 3.5., te 4. i 5. DIPLOMSKOG RADA.

3.4.2. Obrazac Options u *Optimization Tool* prozoru

Obrazac **Options** sadrži 14 glavnih postavki koje je moguće mijenjati. Ovisno o tim postavkama promijeniti će se i izvedba algoritma i njegovi krajnji rezultati. Rezultati se mogu pogoršati ili poboljšati. Da bi postavke bile najprikladnije za problem koji se trenutno rješava

potrebni su višestruki pokušaji i promašaji. Višestruko ispitivanje istog problema uz promijenjene postavke i praćenje dobivenih rezultata izvući će najbolje iz genetskog algoritma.

Slika 3.5. Obrazac Options u *Optimization Tool* prozoru

Opis svih glavnih četrnaest postavki slijedi u nastavku. Prve na redu u obrascu **Options** su postavke populacije (*Population*).

3.4.2.1. Populacija

Postavke populacije su slijedeće:

- **Tip populacije (*Population type*)**. Predstavlja način zapisivanja vrijednosti varijabli prikladan za genetski algoritam. Može biti dvostruki vektor (*Double* vector), binarni broj određene duljine (Bit *string*), ili korisnički definiran (*Custom*). Obično se uzima dvostruki vektor jer tada algoritam radi i kad su postavljena ograničenja optimizacijskog problema. Kad se definira korisnički tada je potrebno napisati funkcije stvaranja početne populacije i funkcije križanja i mutacije, te ih unijeti u predviđena polja (čelije) u obrascu.
- **Veličina populacije (*Population size*)**. Ova postavka određuje koliko se različitih vrijednosti dvostrukog vektora (ili binarnog broja), odnosno točaka domene razmatra u svakom koraku algoritma. Točka domene funkcije cilja naziva se i jedinka populacije. Pretpostavljena vrijednost je 20. Ova postavka se može odrediti i kao vektor veličine veće od jedan, tada se izrađuje više pod-populacija. Vektorom npr. [20 50 100] izrađuju se tri pod-populacije veličine 20, 50 i 100 jedinki.
- **Funkcija izrade početne populacije (*Creation function*)**. Može se odabrati postavka *Use constraint dependent default,* koja odabire jedinke ovisno o tome postoje li ili ne postoje ograničenja. Ako nema ograničenja tada se jednoliko (sa jednakom vjerojatnošću, dakle slučajno) odabire jedinke iz cijele domene funkcije cilja (*Uniform*). Ako ograničenja postoje tada se odabiru moguće jedinke (*Feasible*), a to su jedinke čije

se vrijednosti varijabli nalaze unutar ograničenja. Moguće je i korisnički definirati postavku.

- **Početna populacija (*Initial population*)**. Moguće je posebno odrediti početnu populaciju, inače se ona izrađuje pomoću postavke *Creation function*.
- **Početni rezultati (*Initial scores*)**. Moguće je rezultate za početnu populaciju definirati drukčije nego što je funkcija cilja.
- **Početni raspon (*Initial range*)**. Ovom postavkom određuje se raspon između jedinki unutar početne populacije. Definira se donja i gornja granica za varijable funkcije cilja (jedinke) koje će biti odabrane u početnu populaciju.

Za genetski algoritam izrazito je važna raznolikost populacije. O raznolikosti populacije, ili raspršenosti vrijednosti varijabli funkcije cilja, ovisi izvedba algoritma tijekom evolucije. Bitno je početnu populaciju dovoljno rasprostraniti po području definicije funkcije cilja (domeni) kako bi se algoritmom i u slijedećim generacijama ispitivalo dovoljno široko područje. Potrebno je također voditi brigu o ograničenjima i određenim donjim i gornjim granicama. Ako se populacija suzi na određeno područje postoji opasnost zaglavljivanja algoritma u lokalnom optimumu. Ovdje je potrebno napomenuti da je operacija mutacije instrument kojim se osigurava od zaglavljivanja u lokalnom optimumu, ali ta operacija neznatno mijenja svojstva populacije. Također je potrebno osigurati optimalnu raznolikost populacije, ne preveliku i ne premalu. To je dakle stvar pokušaja i promašaja i ovisi o funkciji cilja čije rješenje se traži. Na osnovu najboljih rezultata iz više pokušaja određuje se i raznolikost početne populacije. Na korisniku je dakle da odredi raznolikost samo početne populacije, dok se algoritmom određuju ostale populacije u sljedećim koracima.

3.4.2.2. Stupnjevanje jačine jedinke

Jačina (*fitness*) pojedine jedinke se dobiva izračunavanjem omjera vrijednosti funkcije cilja za tu jedinku i prosječne vrijednosti funkcije cilja cijele populacije u promatranom koraku. Ova postavka određuje stupanj jačine (*Fitness scale*) jedinke za potrebe funkcije odabira jedinki (*Selection*) za populaciju u sljedećem koraku. Stupanj jačine, kojeg izračunava funkcija stupnjevanja (*Scaling function*) je dakle izveden iz vrijednosti funkcije cilja. Svaka jedinka (točka domene funkcije cilja) daje određenu vrijednost funkcije cilja. Te vrijednosti moraju se stupnjevati (skalirati) kako bi se ocijenilo koje su najjače i najslabije. Pojam jačine u skladu je sa postavkama teorije evolucije gdje se kaže da najjače jedinke preživljavaju. U slučaju genetskog algoritma pojam jačine ovisi o traženom ekstremu funkcije cilja, minimumu ili maksimumu. Ako se traži minimum, najjače jedinke u populaciji su one koje daju minimalne vrijednosti funkcije cilja, kod maksimuma to su one sa maksimalnim vrijednostima. O jačinama ovisi izvedba algoritma. Ako se one mnogo razlikuju algoritam brzo konvergira (popuni se genetski kapacitet ili bazen) i ne pretražuje se ostalo područje. Ako je ta razlika slabo izražena, sve jedinke imaju približno jednaku vjerojatnost odabira i algoritam će biti sporiji u konvergiranju prema konačnom rješenju.

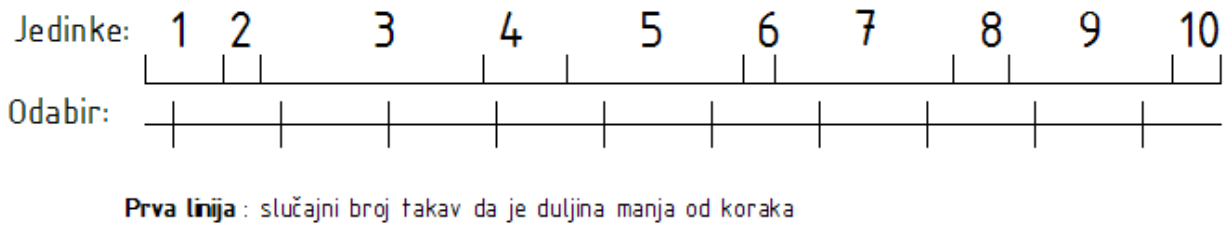
Moguće je odabrati slijedeće vrste funkcije stupnjevanja (*Scaling function*):

- **Poredak (*Rank*)**. Smješta jedinke u poredak (1., 2., 3.,…) na temelju vrijednosti funkcije cilja za svaku jedinku u populaciji.
- **Proporcionalnost (*Proportional*)**. Izrađuje očekivanja koja su razmjerna rezultatima funkcije cilja.
- **Vrh (*Top*)**. Ova mogućnost stupnjevanja jedinki smješta najjače jedinke (one sa najmanjim ili najvećim vrijednostima funkcije cilja) na vrh bez poretka. Ako se odabere ova mogućnost moguće je i odrediti broj jedinki [1,..,*Population size*] koje će proizvesti izdanke. Te jedinke imaju jednaku vjerojatnost odabira.
- **Linearni pomak (*Shift linear*)**. Stupnjevanje vrijednosti funkcije cilja takvo da očekivanje najjače jedinke bude jednako konstanti. Ta konstanta se može odrediti kao maksimalni stupanj preživljavanja pomnožen sa prosječnim rezultatom.
- **Korisnička (*Custom*)**. Mogućnost definiranja funkcije stupnjevanja prema želji.

3.4.2.3. Selekcija

Funkcijom selekcije odabire se jedinke za populaciju u slijedećem koraku, na temelju stupnjevanih jačina. Mogu se odabrati sljedeće karakteristike ove funkcije:

- **Stohastičko jednoliko (*Stochastic uniform*)**. Odabir jedinki tako da se prva odabere na temelju slučajnog broja, a ostale u pravilnim jednakim razmacima. Jedinke imaju vjerojatnost odabira ovisno o jačini. To se može prikazati kao dužina sa različitim dijelovima za svaku jedinku posebno. Algoritam na paralelnoj liniji ocrtava crtice, koje označavaju razmake između dva odabira. Prva duljina je određena slučajno, a ostale su u pravilnim razmacima. Na slici 3.4. prikazan je slučaj sa 10 jedinki. Odabir je izvršen na način: 1. i 4. jedinka odabrane su jednom, 2., 6., 8. i 10. niti jednom, 3., 5., 7. i 9. dva puta.



Slika 3.6. Stohastičko jednoliko odabiranje jedinki

- **Ostatak (*Remainder*)**. Dodjeljuje roditelje (*parents*) novoj populaciji deterministički preko funkcije jačine, a ostatak jedinki se bira na principu ruleta.
- **Jednoliko (*Uniform*)**. Slučajni odabir jedinki na temelju jednolike razdiobe vjerojatnosti. Koristi se samo za probu genetskog algoritma.
- **Linearni pomak (*Shift linear*)**. Definirano kao i u postavkama funkcije stupnjevanja jačine.
- **Rulet (*Roulette*)**. Simulacija kola ruleta i odabir jedinki (roditelja) ovisno o jačini.

- **Turnirski odabir (*Tournament*)**. Jedinke se biraju na temelju slučajnih brojeva, te unutar te populacije (određene veličine: *Tournament size*) izvvršiti ocjenjivanje i odabir najboljih za slijedeći korak.
- **Korisničke postavke (*Custom*)**. Programiranjem samostalno izraditi funkciju odabiranja jedinki.

3.4.2.4. Reprodukcija

Postavkama reprodukcije (*Reproduction*) se određuje kako genetski algoritam izrađuje novu generaciju. Mogućnosti su slijedeće:

- **Broj elitnih jedinki (*Elite count*)**. To su jedinke koje se izdvajaju iz populacije trenutne generacije i na njima se ne vrši nikakva daljnja operacija, kao što su križanje ili mutacija. Dobro je postaviti što manje elitnih jedinki zbog osiguranja raznolikosti populacije. Elitne jedinke služe za usmjeravanje populacije, a ne smiju u njoj dominirati jer je tada moguća prerana konvergencija.
- **Broj jedinki koje će se dobiti križanjem (*Crossover fraction*)**. Te jedinke su dakle frakcija populacije u slijedećoj generaciji dobivena operacijom križanja jedinki iz prethodne generacije. Broj ovih jedinki određuje izvedbu algoritma, stoga je potrebno izabrati optimalan broj. To ovisi o funkciji cilja.

3.4.2.5. Mutacija

Funkcija mutacije (*Mutation function*) radi vrlo male promjene nad cijelom populacijom. Mijenja se samo mali broj gena pojedine jedinke (reda veličine jedna ili dvije), a mnoge ostaju nepromijenjene. Ipak ova funkcija igra značajnu ulogu u osiguravanju algoritma od konvergencije prema lokalnom optimumu. Dovoljno je da samo jedna jedinka, posredstvom mutacije, izađe van prostora lokalnog optimuma i algoritmom će se izraditi nova populacija oko nje. Tada se algoritam usmjerava u drugom pravcu. Operacija mutacije jednostavnog genetičkog algoritma objašnjena je u točki 3.1.5. DIPLOMSKOG RADA. Moguće je odabrati sljedeće postavke funkcije mutacije:

- **Automatska postavka ovisna o ograničenjima (*Use constraint dependent default*)**. Za ovu postavku automatski se izabire način mutiranja gena ovisno o ograničenjima, i to:
  1. normalnom razdiobom (*Gaussian*), ako ograničenja nisu definirana,
  2. moguće i prilagodljivo (*Adaptive feasible*), ako ograničenja jesu definirana.

Termin „moguće“ (prihvatljivo, ono koje funkcionira) znači da jedinka nakon mutacije mora biti u području mogućih rješenja koje je određeno ograničenjima.

- **Normalna razdioba (*Gaussian*).** Pridružuje se po jedan slučajni broj svakoj znamenki jedinki u cijeloj populaciji. Slučajni broj je generiran prema zakonu normalne razdiobe koja ima aritmetičku sredinu vrijednosti 1 u nuli (gausova funkcija - zvono). Količina mutacije je proporcionalna dakle sa standardnom devijacijom razdiobe i smanjuje se sa

svakom slijedećom generacijom. Količina mutacije (mutiranih gena) može se kontrolirati parametrima *Scale* i *Shrink*:

1. ***Scale*** određuje standardnu devijaciju mutacije u prvoj generaciji tako da se taj parametar pomnoži sa rasponom početne populacije (*Initial Range*) definiranom u postavkama populacije.
2. ***Shrink*** kontrolira stupanj prema kojem se smanjuje prosječna količina mutacije. Standardna devijacija se smanjuje linearno, stoga će u konačnom koraku biti smanjena za *Shrink* puta svoje početne vrijednosti u prvoj generaciji.

- **Jednolika razdioba (*Uniform*)**. Proces se odvija na osnovu jednolike razdiobe u dva koraka. U prvom koraku algoritmom se odabere dio (frakcija) znamenaka (ili ulaza vektora) jedinke za mutaciju svaka znamenka (ili ulaz vektora) jedinke, odnosno svaki gen, ima jednaku vjerojatnost mutiranja prema zadanom parametru p_m. U drugom koraku odabrani geni mutiraju ovisno o pridruženom slučajnom broju u druge gene koji određuju jedinke.
- **Moguće i prilagodljivo (*Adaptive feasible*)**. Slučajno se generiraju smjerovi koji su prilagodljivi ovisno o prethodnoj uspješnoj ili neuspješnoj generaciji. Duljina koraka se odabire između svih mogućih smjerova kako bi se zadovoljila linearna ograničenja i granice. Uočljiva je analogija sa numeričkom *Hooke – Jevesovom* metodom.
- **Korisničke postavke (*Custom*)**.

3.4.2.6. Križanje

Moguće je odrediti postavke funkcije križanja (*Crossover function*) i to na slijedeći način:

- **Nepovezano (*Scattered*)**. Izrađuje se slučajni binarni vektor (ili broj), odnosno niz nula i jedinica slučajno odabranih. Taj vektor služi za odabir znamenki (gena) odabranih jedinki (roditelja) za oblikovanje znamenki (gena) slijedeće jedinke (djeteta). Pozicija znamenke slučajnog binarnog vektora i njena vrijednost (0 ili 1) određuje koji geni iz dvaju roditelja se biraju za dijete. Ako na *i*-toj poziciji slučajnog binarnog vektora stoji 1 bira se *i*-ti gen iz prvog roditelja, ako pak stoji 0 bira se *i*-ti gen drugog roditelja. Prikaz na slici 3.5.

1. roditelj:	[1 2 3 4 5 6 7]
2. roditelj:	[a b c d e f g]
Slučajni binarni vektor:	[0 1 0 0 1 1 0]
Dijete:	[a 2 c d 5 6 g]

Slika 3.7. Križanje prema *Scattered* postavci funkcije križanja

- **Jedna točka križanja (*Single point*)**. Odabire se slučajni broj *n* iz raspona [1,…, *m* - 1], gdje je *n* znamenka djeteta koja predstavlja točku križanja, a *m* je ukupan broj znamenki djeteta. Nakon operacije križanja, prvih *n* znamenki djeteta (do uključivo *n*-te znamenke)

su prvih *n* znamenki iz prvog roditelja, dok su ostale znamenke djeteta (od *n*+1 do *m*) preostale znamenke iz drugog roditelja (prikazano na slici 3.6).

1. roditelj: [1 2 3 4 5 6 7]
2. roditelj: [a b c d e f g]
Slučajni broj iz raspona [1, 7]: 3
Dijete: [1 2 3 d e f g]

Slika 3.8. Križanje prema *Single point* postavci funkcije križanja

- **Dvije točke križanja (*Two points*)**. Biraju (generiraju) se dva slučajna broja *m* i *n* iz raspona [1,…, *M*], gdje je *m* prva točka križanja, *n* druga točka križanja, a *M* ukupan broj znamenki (gena) djeteta i jednak je broju znamenki (gena) oba roditelja. Nakon operacije križanja, prvih *m* znamenki djeteta (od 1 do uključivo *m*) su znamenke odabrane iz prvog roditelja, drugi dio znamenki djeteta (od *m*+1 do *n* uključivo) je odabrano iz drugog roditelja (iste pozicije znamenki), dok je preostali dio znamenki djeteta (od *n*+1 do *M*) odabran iz prvog roditelja (također iste pozicije znamenki). Prikaz na slici 3.7.

1. roditelj: [1 2 3 4 5 6 7 8 9]
2. roditelj: [a b c d e f g h i]
Slučajni brojevi iz raspona [1, 9]: *m* = 3, *n* = 7
Dijete: [1 2 3 d e f g 8 9]

Slika 3.9. Križanje prema *Two point* postavci funkcije križanja

- **Srednje (*Intermediate*)**. Djeca se izrađuju pomoću srednje vrijednosti roditelja pomnožene sa slučajnim brojem. Križanje je kontrolirano omjerom (*Ratio*) u obliku slijedećeg izraza: **dijete1** = **roditelj1 + rand** * **Ratio *** (**roditelj2** – **roditelj1**). Omjer (*Ratio*) može biti unutar granica [0,1], ili izvan tih granica. Omjer određuje smještaj djece, a to može biti unutar hiperkocke ili izvan nje.
- **Heuristično (*Heuristic*)**. Određuje se dijete koje leži na spojnoj crti dvaju roditelja. Dijete je bliže roditelju s boljim svojstvima. Parametrom omjer (*Ratio*) može se odrediti koliko je dijete udaljeno od boljeg roditelja. Ako roditelj1 ima bolja svojstva, dijete se određuje slijedećom formulom: **dijete1** = **roditelj2 + Ratio *** (**roditelj1** – **roditelj2**).
- **Aritmetičko (*Arithmetic*)**. Izrađuju se djeca koja su težinska aritmetička sredina roditelja.
- **Korisnički (*Custom*)**.

3.4.2.7. Migracija

Migracija je kretanje jedinki između pod-populacija koje algoritam izrađuje ako se veličina populacije (*Population size*) unutar postavki populacije postavi kao vektor dimenzije veće od jedan. Vektor definiran pomoću sintakse npr.: [50 30 100] određuje tri pod-populacije veličine 50 u prvoj, 30 u drugoj i 100 jedinki u trećoj pod-populaciji. Postavkama migracije određuje se

kako jedinke putuju među pod-populacijama. Kada dođe do migracije, najbolje jedinke iz jedne pod-populacije zamjenjuju najlošije jedinke u drugoj pod-populaciji. Jedinke koje migriraju iz jedne pod-populacije u drugu samo su kopirane u drugu pod-populaciju i nisu uklonjene iz vlaste pod-populacije. Moguće je odrediti sljedeće postavke:

- **Smjer (*Direction*)**. Ovom postavkom određuje se smjer u kojem se migracija odvija. Može se odabrati dva načina migriranja:

  - **Naprijed (*Forward*)**. Migriranje se odvija prema posljednjoj pod-populaciji i to tako da jedinke iz *n*-te pod-populacije (*n*-ti element *Population size* vektora) migriraju prema (*n*+1)-toj pod-populaciji ((*n*+1)-ti element *Population size* vektora).
  - **Dvostruko (*Both*)**. Jedinke iz *n*-te pod-populacije migriraju u dva smjera i to prema (*n*-1)-toj i (*n*+1)-toj pod-populaciji.

Migracija se može zavaljati između krajnjih pod-populacija, odnosno odvija se migracija između prve i posljednje pod-populacije i obrnuto. Ta pojava se može izbjeći postavljanjem jednog elementa vektora *Population size* na nulu, npr: [50 0 30 100].

- **Odjeljak (*Fraction*)**. Ovom postavkom određuje se koliko jedinki će se kretati između pod-populacija. Vrijednost odjeljka se postavlja kao decimalni broj unutar raspona ‹0, 1]. Veličina najmanje pod-populacije unutar definiranog vektora *Population size* uzima se kao mjerodavna za određivanje broja jedinki koje migriraju. Npr. za gore spomenuti vektor populacije [50 0 30 100] i uz postavku *Fraction* kao 0.1 broj jedinki koje migriraju je 30x0.1 = 3, dakle tri jedinke.
- **Raspon (*Interval*)**. Ovom postavkom određuje se broj generacija između odvijanja migracija. Ako se Raspon postavi kao broj 20, tada se u algoritmu migracija odvija svakih 20 generacija.

3.4.2.8. Postavke algoritma

Postavkama algoritma (*Algorithm settings*) određuju se specifični parametri algoritma, a to su:

- ***Initial penalty***: Broj veći ili jednak od jedan (x≥1), kojim se određuje početna vrijednost kažnjavanja algoritma u slučaju pogreške ili nezadovoljavajućih rezultata.
- ***Penalty factor***: faktor (također broj veći od jedan, f≥1) kojim se povećava početni parametar kazne algoritma kada se problem ne rješava na zahtijevanu preciznost ili uz zadana ograničenja.

3.4.2.9. Hibridna funkcija

Postavkama hibridne funkcije (*Hybrid function*) omogućeno je određivanje druge funkcije minimizacije (drugog algoritma) koja se pokreće nakon zaustavljanja genetskog algoritma. Ova

postavka se upotrebljava kada genetski algoritam ne daje zadovoljavajuće konačne rezultate, odnosno daje rezultate oko optimuma. Moguće je odabrati slijedeće funkcije traženja optimuma:

- **Nijedna (*None*)**. Nema određenih tehnika traženja optimuma
- **Funkcija traženja minimuma (*fminsearch*)**. Samo za probleme bez ograničenja.
- **Funkcija uobičajenog traženja minimuma (*patternsearch*)**. Za probleme bez i sa ograničenjima.
- **Problemi bez ograničenja (*fminunc*)**.
- **Problemi sa ograničenjima (*fmincon*)**.

3.4.2.10. Kriteriji zaustavljanja algoritma

Ovim postavkama određuje se kakvi će biti uzroci zaustavljanja algoritma. Mogu se odrediti slijedeće postavke:

- **Broj generacija (*Generations*)**. Ovom postavkom određuje se najveći broj generacija (iteracija) u izvođenju genetskog algoritma.
- **Vremensko ograničenje (*Time limit*)**. Ovom postavkom određuje se ukupno vrijeme izvođenja algoritma u sekundama, neovisno o tome da li se pronašao optimum ili ne.
- **Granica jačine (*Fitness limit*)**. Ako je poznata jačina jedinke, odnosno vrijednost funkcije cilja, koja predstavlja optimum onda se ta vrijednost može zadati kao granica na kojoj se algoritam zaustavlja.
- **Broj generacija u zaustavljanju algoritma (*Stall generations*)**. Prilikom izračunavanja jačine najbolje jedinke u generaciji genetski algoritam vrši usporedbu sa najboljim jedinkama iz prethodnih generacija. Ako je promjena vrijednosti jačine najboljih jedinki u uzastopnim generacijama unutar zadane tolerancije tada algoritam započinje proces zaustavljanja. Proces zaustavljanja traje onoliko generacija koliko je zadano ovom postavkom.
- **Vrijeme trajanja zaustavljanja (*Stall time limit*)**. Ako prilikom izvođenja algoritma nema naznaka poboljšanja vrijednosti jačine najboljih jedinki u generacijama tada se algoritam zaustavlja za određeno vrijeme. Vrijeme zaustavljanja algoritma u sekundama određuje se ovom postavkom.
- **Tolerancija funkcije (*Function tolerance*)**. Ako se kumulativna promjena vrijednosti funkcije cilja ne mijenja unutar vrijednosti zadane postavkom tolerancije funkcije (*Function tolerance*), te ako se ta promjena ne dogodi u generacijama unutar procesa zaustavljanja algoritma, algoritam prekida s izvođenjem.
- **Tolerancija nelinearnih ograničenja (*Nonlinear constraint tolerance*)**. Ovom postavkom određuje se tolerancija prekidanja algoritma za koju je moguće izići van područja nelinearnih ograničenja.

3.4.2.11. Funkcije / postavke ispisa

Postavkama, odnosno funkcijama ispisa (*Plot functions*) omogućava se grafički ispis

rezultata izvođenja algoritma s različitih gledišta. Također je moguće zaustaviti proces izvođenja algoritma u bilo kojem trenutku (koraku). Moguće je odabrati slijedeće postavke / funkcije ispisa:

- **Interval ispisa (*Plot interval*)**. Ovom postavkom određuje se broj generacija između dva uzastopna ispisa rezultata izvođenja algoritma. Uobičajena vrijednost je jedan, dakle nakon svakog izvedenog koraka algoritma ispiše se i željeni rezultat.
- **Najjača jedinka (*Best fitness*)**. Odabirom ove funkcije grafički se ispisuju vrijednosti jačine najboljih jedinki po generacijama, najjačih jedinki. Prevedeno u jezik matematike ovom funkcijom ispisa prikazuju se najmanje / najveće vrijednosti funkcije cilja po koracima izvođenja algoritma u problemu traženja minimalne / maksimalne vrijednosti funkcije cilja. Graf ima dvije osi. Na apscisnoj (horizontalnoj) osi prikazan je broj generacija (koraka algoritma), a na ordinatnoj (vertikalnoj) osi najveće vrijednosti jačine jedinki i prosječne vrijednosti jačine jedinki u promatranoj generaciji. Iz ovog grafa ne vidi se koja jedinka ima najveću vrijednost jačine u promatranoj generaciji. Graf s takvim rezultatima se može izraditi samostalno programiranjem.
- **Najbolja jedinka (*Best individual*)**. Odabirom ove funkcije ispisuju se vektori jedinki koje imaju najbolje vrijednosti funkcije cilja po generacijama. U grafu je moguće uočiti samo konačan rezultat, jer se proces odvija prebrzo. Vektor jedinke je skup elemenata koji predstavljaju vrijednosti varijabli funkcije cilja. Zadnji ispisani rezultat, ako izvođenje algoritma nije namjerno prekinuto, predstavlja optimum dobiven genetskim algoritmom.
- **Udaljenost (*Distance*)**. Odabirom ove funkcije grafički se ispisuju prosječne udaljenosti među jedinkama po generacijama. Udaljenost (*Distance*) predočava koliko je raspršena populacija jedinki u trenutnoj generaciji (koraku algoritma). Dakle što je veća prosječna udaljenost između jedinki to se pretražuje šire područje domene funkcije cilja. Prosječna udaljenost jedinki sve se više smanjuje kako algoritam konvergira prema optimumu.
- **Očekivanje (*Expectation*)**. Odabirom ove funkcije grafički se ispsuje očekivani broj djece, dakle jedinki nastalih genetskim operacijama križanja i mutacije, u odnosu na neobrađene jedinke, odnosno nepoznat im je uzrok nastanka.
- **Obiteljsko stablo (*Genealogy*)**. Odabirom ove funkcije grafički se ispisuje porijeklo jedinki. Porijeklo se ocrtava linijama različitih boja i to na sljedeći način:
  - crvene linije: porijeklo djece dobivene mutiranjem,
  - plave linije: porijeklo djece dobivene križanjem,
  - crne linije: označavaju porijeklo elitnih jedinki (one se kopiraju iz koraka u korak).

  Ovaj graf nije prikladan za pregled osim ako se prikaže u velikim dimenzijama jer zbog mnoštva linija nije moguće očitati porijeklo jedinki.
- **Raspon (*Range*)**. Odabirom ove funkcije grafički se ispisuju minimalne, maksimalne, te prosječne vrijednosti funkcije cilja po generacijama. Graf prikazuje trenutne rezultate, stoga je konačni ispisani rezultat prikaz stanja posljednje generacije u algoritmu.
- **Raznovrsnost rezultata (*Score diversity*)**. Ispis histograma rezultata po generacijama. Ispis prikazuje trenutno stanje (rezultati trenutno izvedenog koraka).
- **Odabir (*Selection*)**. Ispis histograma roditelja.
- **Zaustavljanje (*Stopping*)**. Ispis kriterija zbog kojih se algoritam zaustavio i koliko je trajao proces zaustavljanja.

- **Maksimalno prekoračenje ograničenja (*Max. constraint*)**. Ispis maksimalnog prekoračenja izvan nelinearnih ograničenja tijekom izvedbe algoritma.
- **Prema želji (*Custom*)**. Unos samostalno u M-datoteci izrađene funkcije ispisa rezultata.

3.4.2.12. Izlazna funkcija

Ovom postavkom moguće je prikazati izlazne podatke tijekom izvedbe algoritma i to:

- **Povijesni podaci u odvojenom prozoru (*History to new window*)**. Na odvojenom prozoru ispisuju se izlazni podaci tijekom izvedbe algoritma.
- **Interval**. Određuje se broj generacija (iteracija) između dva uzastopna ispisa podataka.
- **Prema želji (*Custom*)**. Moguće je definirati vlastitu izlaznu funkciju.

3.4.2.13. Prikaz podataka u prozoru za naredbe

Ovom postavkom (*Display to Command window)* moguće je detaljno prikazati u prozoru za naredbe tijek izvedbe genetskog algoritma. Podaci se mogu prikazati iterativno, te dijagnostički, tako da se prikažu odabir jedinki, funkcije odabira, genetskih operatora, funkcije izračunavanja jačine jedinki, razlog zaustavljanja algoritma itd.

3.4.2.14. Ocjena korisničkih funkcija

Ovom postavkom određuje se način ocjenjivanja funkcije cilja i zadanih ograničenja za svaku jedinku u populaciji. Ocjenjivanje korisničkih funkcija predstavlja funkcije u algoritmu kojima se određuju vrijednosti funkcije cilja svake pojedine jedinke, te da li jedinke zadovoljavaju ograničenja, itd. Moguće je odabrati slijedeće načine ocjenjivanja:

- **Serijsko (*serial*)**. Funkcija cilja i zadana ograničenja za svaki član populacije (jedinku) ocjenjuju se odvojeno.
- **Vektorski (*vectorized*)**. Funkcija cilja i ograničenja za cijelu populaciju ocjenjuju se pozivom jedne funkcije.
- **Paralelno (*in parallel*)**. Moguće je odabrati ovu postavku samo ako se omogući računalno višeprocesorsko radno okruženje.

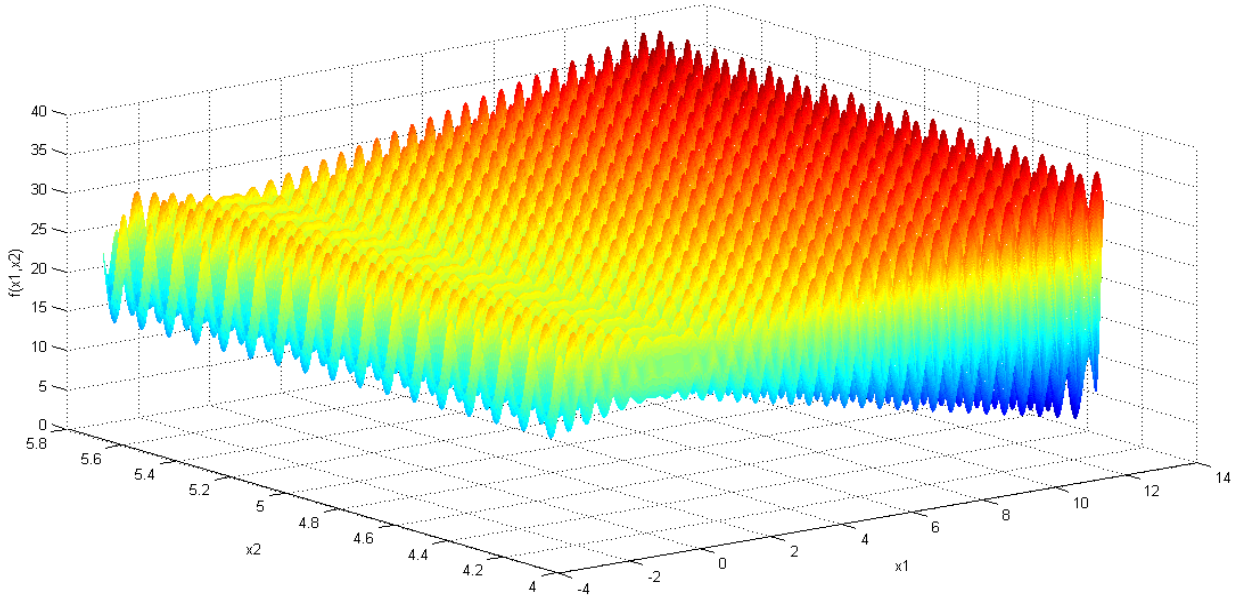
3.5. Prikaz izvedbe jednostavnog genetskog algoritma na primjeru

Kako bi se stekao uvid u način rada bilo koje vrste genetskog algoritma korisno je objasniti izvođenje jednostavnog genetskog algoritma na primjeru. Složeniji genetski algoritmi nadograđuju se na temeljima jednostavnih, a koje je postavio J. Holand 1975. godine zajedno sa skupinom studenata. Temeljito su prikazane osnove rada samo prvog koraka (iteracije), dok se cijela evolucija izvodi u programskom paketu MATLAB u točki 3.5. DIPLOMSKOG RADA. Također se može uočiti mala razlika u terminima, npr. jedinka se naziva kromosom, itd.

3.5.1. Problem traženja maksimalne vrijednosti na području definicije za periodički oblik funkcije cilja [3]

Funkcija za koju se traži maksimalna vrijednost (optimum) na području definicije je zadana tako da se prikaže prednost genetskog algoritma u odnosu na klasične metode traženja optimuma. Periodički oblik funkcije ima beskonačan broj lokalnih maksimuma i minimuma na cijelom području definicije. Stoga će se područje definicije ograničiti, tako da se pretražuje konačan broj mogućih rješenja. Funkcija ima dvije varijable x_1 i x_2, stoga je i prostornog oblika. Prikazano na slici 3.5. Klasične metode traženja maksimalne (minimalne) vrijednosti bi se zaustavile pri pronalasku lokalnog optimuma, što nije dovoljno za rješavanje problema. Genetski algoritam će vršiti usporedbu najboljih rješenja iz svih populacija (generacija) i odabrati ono najbolje na području definicije. Zbog ograničene domene algoritam konvergira prema zadanom cilju, maksimumu funkcije na području definicije. Funkcija cilja je oblika:

$$f(x_1, x_2) = 21{,}5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2) \rightarrow \max \;,$$



Slika 3.10. Periodička funkcija cilja, sinusoida u prostoru

uz ograničenje $-3{,}0 \le x_1 \le 12{,}1$, što predstavlja domenu D_1 i $4{,}1 \le x_2 \le 5{,}8$, što predstavlja domenu D_2. Potrebno je odrediti preciznost realnih brojeva na području definicije, odnosno diskretizirati to područje. Neka je tražena preciznost realnih brojeva na četvrtu decimalu. Jedinica će biti podijeljena na 10 000 dijelova. To znači da se skupom prirodnih brojeva {1…10 000} opisuje interval realnih brojeva npr. unutar granica [-3,0001;…;-2,0000]. Varijabla x_1 promatra se na domeni D_1 koja ima 12,1 – (-3,0) = 15,1 jedinica. Varijabla x_2 promatra se na domeni D_2 koja ima 5,8 – 4,1 = 1,7 jedinica. Stoga je potrebno domenu D_1 podijeliti (diskretizirati) na 15,1x10 000, a domenu D_2 na 1,7x10 000 ekvidistantnih dijelova. Odnosno za

opis domene D_1 potrebno je 151 000 prirodnih brojeva, a za opis domene D_2 potrebno je 17 000 prirodnih brojeva. Naravno zbog oblika binarnog zapisa prirodnih brojeva neće biti moguće za svaku domenu dobiti točno toliko, minimalno potrebno brojeva, već onoliko koliko to omogućava cijeli eksponent baze 2. Uz traženu preciznost od četiri decimale slijedi da je za opis domene D_1 potrebno osigurati:

$$2^{17} = 131\ 072 < 151\ 000 < 2^{18} = 262\ 144$$

minimalno 262 144 prirodna broja, odnosno 18 mjesta (za upis znamenki) u binarnom zapisu. Za domenu D_2 potrebno je osigurati:

$$2^{14} = 16\ 384 < 17\ 000 < 2^{15} = 32\ 768$$

minimalno 32 768 prirodnih brojeva, odnosno 15 mjesta u binarnom zapisu. Dakle za cijeli kromosom v_i, koji će predstavljati točku (x_1, x_2) na području definicije funkcije $f(x_1, x_2)$, biti će potrebno 18 + 15 = 33 mjesta u binarnom zapisu.

Sada kad je definiran oblik kromosoma moguće je odrediti populaciju istih koja će se pretraživati i ocjenjivati genetskim algoritmom. Početna populacija određuje se nasumičnim odabirom, a odabire se veličina populacije *pop_size* =20 jedinki (kromosoma). Početna populacija izgleda kako slijedi:

v_1 = (1001101000 00001111 1110100110 11111)
v_2 = (1110001001 00110111 0010101000 11010)
v_3 = (0000100000 11001000 0010101110 11101)
v_4 = (1000110001 01101001 1110000011 10010)
v_5 = (0001110110 01010011 0101111110 00101)
v_6 = (0001010000 10010101 0010101111 11011)
v_7 = (0010001000 00110101 1110110111 11011)
v_8 = (1000011000 01110100 0101101011 00111)
v_9 = (0100000001 01100010 1100000011 11100)
v_{10} = (0000011110 00110000 0110100001 11011)
v_{11} = (0110011111 10110101 1000011011 11000)
v_{12} = (1101000101 11101101 0001010100 00000)
v_{13} = (1110111110 10001000 1100000010 00110)
v_{14} = (0100100110 00001010 1001111001 01001)
v_{15} = (1110111011 01110000 1000111110 11110)
v_{16} = (1100111100 00011111 1000011010 01011)
v_{17} = (0110101111 11001111 0100011011 11101)
v_{18} = (0111010000 00001110 1001111101 01101)
v_{19} = (0001010100 11111111 1100001100 01100)
v_{20} = (1011100101 10011110 0110001011 11110)

Zbog preglednosti kromosoma vide se dva odjeljenja u svakom pojedinom kromosomu. Ona predstavljaju binarni zapise za varijablu x_1 (18 binarnih znamenki) i za varijablu x_2 (15 binarnih znamenki). Te binarne zapise u svakom kromosomu pojedinačno potrebno je dekodirati. Tako se određuju vrijednosti varijabli preko kojih se dobivaju vrijednosti funkcije cilja u točkama, odnosno vrši se evaluacija. Tada je moguće uspoređivati vrijednosti funkcije cilja za svaki pojedini kromosom ocjenjivanjem (posredstvom funkcije *eval*) i za čitavu populaciju (generaciju) odabrati najbolja rješenja. Taj postupak naziva se izračunavanje jačine jedinke. Slijedi dakle dekodiranje kromosoma prema izrazu (3.4), prikazano je izračunavanje obje varijable i vrijednost funkcije cilja za te varijable:

kromosom v_1 : $x_1 = -3{,}0 + decimal\ (1001101000\ \ 00001111) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} = 6{,}0845$

$x_2 = 4{,}1 + decimal\ (1110100110\ \ 11111) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} = 5{,}6522$

$f(x_1, x_2) = 26{,}0196$

kromosom v_2 : $x_1 = -3{,}0 + decimal\ (1110001001\ \ 00110111) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} = 10{,}3484$

$x_2 = 4{,}1 + decimal\ (0010101000\ \ 11010) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} = 4{,}3803$

$f(x_1, x_2) = 7{,}5800$

kromosom v_3 : $x_1 = -3{,}0 + decimal\ (0000100000\ \ 11001000) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} = -2{,}5166$

$x_2 = 4{,}1 + decimal\ (0010101110\ \ 11101) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} = 4{,}3904$

$f(x_1, x_2) = 19{,}5263$

kromosom v_4 : $x_1 = -3{,}0 + decimal\ (1000110001\ \ 01101001) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} = 5{,}2786$

$x_2 = 4{,}1 + decimal\ (1110000011\ \ 10010) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} = 5{,}5935$

$f(x_1, x_2) = 17{,}4067$

kromosom v_5 : $x_1 = -3{,}0 + decimal\ (0001110110\ \ 01010011) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} = -1{,}2552$

$x_2 = 4{,}1 + decimal\ (0101111110\ \ 00101) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} = 4{,}7344$

$f(x_1, x_2) = 25{,}3412$

kromosom v_6 : $x_1 = -3{,}0 + decimal\ (0001010000\ \ 10010101) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} = -1{,}8117$

$x_2 = 4{,}1 + decimal\ (0010101111\ \ 11011) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} = 4{,}3919$

$f(x_1, x_2) = 18{,}1004$

kromosom v_7 : $x_1 = -3{,}0 + decimal\ (0010001000\ \ 00110101\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ -0,9915

$x_2 = 4{,}1 + decimal\ (1110110111\ \ 11011\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 5,6803

$f(x_1, x_2) =$ 16,0208

kromosom v_8 : $x_1 = -3{,}0 + decimal\ (1000011000\ \ 01110100\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 4,9106

$x_2 = 4{,}1 + decimal\ (0101101011\ \ 00111\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 4,7030

$f(x_1, x_2) =$ 17,9597

kromosom v_9 : $x_1 = -3{,}0 + decimal\ (0100000001\ \ 01100010\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 0,7954

$x_2 = 4{,}1 + decimal\ (1100000011\ \ 11100\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 5,3815

$f(x_1, x_2) =$ 16,1278

kromosom v_{10} : $x_1 = -3{,}0 + decimal\ (0000011110\ \ 00110000\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ -2,5548

$x_2 = 4{,}1 + decimal\ (0110100001\ \ 11011\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 4,7937

$f(x_1, x_2) =$ 21,2784

kromosom v_{11} : $x_1 = -3{,}0 + decimal\ (0110011111\ \ 10110101\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 3,1301

$x_2 = 4{,}1 + decimal\ (1000011011\ \ 11000\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 4,9961

$f(x_1, x_2) =$ 23,4107

kromosom v_{12} : $x_1 = -3{,}0 + decimal\ (1101000101\ \ 11101101\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 9,3562

$x_2 = 4{,}1 + decimal\ (0001010100\ \ 00000\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 4,2395

$f(x_1, x_2) =$ 15,0116

kromosom v_{13} : $x_1 = -3{,}0 + decimal\ (1110111110\ \ 10001000\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 11,1346

$x_2 = 4{,}1 + decimal\ (1100000010\ \ 00110\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 5,3787

$f(x_1, x_2) =$ 27,3167

kromosom v_{14} : $x_1 = -3{,}0 + decimal\ (0100100110\ \ 00001010\ ) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 1,3359

$x_2 = 4{,}1 + decimal\ (1001111001\ \ 01001\ ) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 5,1514

$f(x_1, x_2) =$ 19,8763

kromosom v_{15} : $x_1 = -3{,}0 + decimal\ (1110111011\ \ 01110000) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 11,0890

$x_2 = 4{,}1 + decimal\ (1000111110\ \ 11110) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 5,0545

$f(x_1, x_2) =$ 30,0602

kromosom v_{16} : $x_1 = -3{,}0 + decimal\ (1100111100\ \ 00011111) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 9,2116

$x_2 = 4{,}1 + decimal\ (1000011010\ \ 01011) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 4,9938

$f(x_1, x_2) =$ 23,8672

kromosom v_{17} : $x_1 = -3{,}0 + decimal\ (0110101111\ \ 11001111) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 3,3675

$x_2 = 4{,}1 + decimal\ (0100011011\ \ 11101) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 4,5713

$f(x_1, x_2) =$ 13,6962

kromosom v_{18} : $x_1 = -3{,}0 + decimal\ (0111010000\ \ 00001110) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 3,8430

$x_2 = 4{,}1 + decimal\ (1001111101\ \ 01101) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 5,1582

$f(x_1, x_2) =$ 15,4141

kromosom v_{19} : $x_1 = -3{,}0 + decimal\ (0001010100\ \ 11111111) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ -1,7466

$x_2 = 4{,}1 + decimal\ (1100001100\ \ 01100) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 5,3956

$f(x_1, x_2) =$ 20,0959

kromosom v_{20} : $x_1 = -3{,}0 + decimal\ (1011100101\ \ 10011110) \cdot \frac{12{,}1-(-3{,}0)}{2^{18}-1} =$ 7,9360

$x_2 = 4{,}1 + decimal\ (0110001011\ \ 11110) \cdot \frac{5{,}8-4{,}1}{2^{15}-1} =$ 4,7573

$f(x_1, x_2) =$ 13,6669

Najveću vrijednost u populaciji ima kromosom v_{15} («najjača» jedinka), dok najmanju ima kromosom v_2 («najslabija» jedinka). Slijedi postupak odabira kromosoma za slijedeću populaciju. To je postupak razvoja, odnosno pretrage drugih rješenja, koja su u svezi sa prethodno odabranim, inače bi pretraga bila posve slučajna. Potrebno je odrediti jačinu *F* populacije (dobrotu funkcije cilja):

$F = \sum_{i=1}^{20} eval\ (v_i) =$ 26,0196+7,5800+19,5263+17,4067+25,3412+18,1004+16,0208+

+ 17,9597 + 16,1278 + 21,2784 + 23,4107 + 15,0116 + 27,3167 + 19,8763 +30,0602 + 23,8672 + 13,6962 + 15,4141 + 20,0959 + 13,6669 = **387,7768**

Vjerojatnost odabira kromosoma za slijedeću populaciju ovisi o omjeru vrijednosti evaluacijske funkcije za taj kromosom i ukupne jačine populacije (dobrote funkcije cilja). Dakle vjerojatnosti odabira kromosoma biti će slijedeće, prikazano u TABLICI 3.1:

TABLICA 3.1. KUMULATIVNE VJEROJATNOSTI ODABIRA KROMOSOMA

Vjerojatnosti p_i:	Kumulativne vjerojatnosti q_i:
$p_1 = eval(v_1)/F = 26{,}0196/387{,}776 = 0{,}0671$	$q_1 = 0{,}0671$
$p_2 = eval(v_2)/F = 7{,}5800/387{,}776 = 0{,}0195$	$q_2 = 0{,}0866$
$p_3 = eval(v_3)/F = 19{,}5264/387{,}776 = 0{,}0503$	$q_3 = 0{,}1370$
$p_4 = eval(v_4)/F = 17{,}4067/387{,}776 = 0{,}0449$	$q_4 = 0{,}1819$
$p_5 = eval(v_5)/F = 25{,}3412/387{,}776 = 0{,}0653$	$q_5 = 0{,}2472$
$p_6 = eval(v_6)/F = 18{,}1004/387{,}776 = 0{,}0467$	$q_6 = 0{,}2939$
$p_7 = eval(v_7)/F = 16{,}0208/387{,}776 = 0{,}0413$	$q_7 = 0{,}3352$
$p_8 = eval(v_8)/F = 17{,}9597/387{,}776 = 0{,}0463$	$q_8 = 0{,}3815$
$p_9 = eval(v_9)/F = 16{,}1278/387{,}776 = 0{,}0416$	$q_9 = 0{,}4231$
$p_{10} = eval(v_{10})/F = 21{,}2784/387{,}776 = 0{,}0549$	$q_{10} = 0{,}4780$
$p_{11} = eval(v_{11})/F = 23{,}417/387{,}776 = 0{,}0604$	$q_{11} = 0{,}5384$
$p_{12} = eval(v_{12})/F = 15{,}0116/387{,}776 = 0{,}0387$	$q_{12} = 0{,}5771$
$p_{13} = eval(v_{13})/F = 27{,}3167/387{,}776 = 0{,}0704$	$q_{13} = 0{,}6475$
$p_{14} = eval(v_{14})/F = 19{,}8763/387{,}776 = 0{,}0512$	$q_{14} = 0{,}6988$
$p_{15} = eval(v_{15})/F = 30{,}0602/387{,}776 = 0{,}0775$	$q_{15} = 0{,}7763$
$p_{16} = eval(v_{16})/F = 23{,}8672/387{,}776 = 0{,}0615$	$q_{16} = 0{,}8379$
$p_{17} = eval(v_{17})/F = 13{,}6962/387{,}776 = 0{,}0353$	$q_{17} = 0{,}8732$
$p_{18} = eval(v_{18})/F = 15{,}4141/387{,}776 = 0{,}0397$	$q_{18} = 0{,}9129$
$p_{19} = eval(v_{19})/F = 20{,}0959/387{,}776 = 0{,}0518$	$q_{19} = 0{,}9647$
$p_{20} = eval(v_{20})/F = 13{,}6669/387{,}776 = 0{,}0352$	$q_{20} = 1{,}0000$

Potrebno je ispisati (generirati) slučajne brojeve unutar raspona [0,…,1], te ih pridružiti intervalima kumulativnih vjerojatnosti i tako odabrati kromosome za slijedeću populaciju.

Kromosomi se odabiru prema pravilu: ako je $r < q_1$ odabrati prvi kromosom v_1, ako je drukčije odabrati *i*-ti kromosom v_i $(2 \le i \le 20)$ takav da vrijedi $q_{i-1} < r \le q_i$. Za ispisane slučajne brojeve odabrani su slijedeći kromosomi, prikazani u TABLICI 3.2:

TABLICA 3.2.: SLUČAJNI ODABIR KROMOSOMA NA TEMELJU EVALUACIJE

Redni broj kromosoma u novoj populaciji:	Slučajni brojevi *r* :	Redni broj odabranog kromosoma iz prethodne populacije:
1	0,51387	11
2	0,17574	4
3	0,30865	7
4	0,53453	11
5	0,94763	19
6	0,17174	4
7	0,70223	15
8	0,22643	5
9	0,49477	11
10	0,42472	3
11	0,70390	15
12	0,38965	9
13	0,27723	6
14	0,36807	8
15	0,98344	20
16	0,00540	1
17	0,76568	10
18	0,64647	13
19	0,76714	15
20	0,78024	16

Sada se može ispisati nova populacija koja se dalje podvrgava osnovnim operacijama genetskog algoritma, križanju i mutaciji. Nova populacija dakle sadrži kromosome koji se pojavljuju više od jednog puta jer im je vjerojatnost odabira veća. Nova populacija u izvedbi ovdje promatranog prvog koraka algoritma još uvijek ne predstavlja novu generaciju. Nova generacija će nastupiti kad se izvrše svi potrebni operatori genetskog algoritma i kad se izvrši konačna evaluacija unutar koraka algoritma. Ovaj način izrade nove populacije za slijedeću generaciju (korak algoritma) ima nedostatak jer se neki kromosomi ponavljaju i potrebna je dvostruka evaluacija. Ponavljanje kromosoma nije dobro zbog neraznovrsnosti populacije, odnosno dva put se ocjenjuju slične populacije, što bitno usporava algoritam. Nova populacija prvog koraka algoritma prikazana je u nastavku.

v'_1 = (0110011111 10110101 1000011011 11000)

v'_2 = (1000110001 01101001 1110000011 10010)

v'_3 = (0010001000 00110101 1110110111 11011)

v'_4 = (0110011111 10110101 1000011011 11000)

v'_5 = (0001010100 11111111 1100001100 01100)

v'_6 = (1000110001 01101001 1110000011 10010)

v'_7 = (1110111011 01110000 1000111110 11110)

v'_8 = (0001110110 01010011 0101111110 00101)

v'_9 = (0110011111 10110101 1000011011 11000)

v'_{10} = (0000100000 11001000 0010101110 11101)

v'_{11} = (1110111011 01110000 1000111110 11110)

v'_{12} = (0100000001 01100010 1100000011 11100)

v'_{13} = (0001010000 10010101 0010101111 11011)

v'_{14} = (1000011000 01110100 0101101011 00111)

v'_{15} = (1011100101 10011110 0110001011 11110)

v'_{16} = (1001101000 00001111 1110100110 11111)

v'_{17} = (0000011110 00110000 0110100001 11011)

v'_{18} = (1110111110 10001000 1100000010 00110)

v'_{19} = (1110111011 01110000 1000111110 11110)

v'_{20} = (1100111100 00011111 1000011010 01011)

Sada je potrebno primijeniti operatore rekombinacije, križanje i mutaciju. Kada se ne bi primijenili ovi operatori genetski algoritam bi pretraživao samo one brojeve (kromosome) koji su odabrani u početnoj populaciji. To je jako malen izbor ako se uzme u obzir da neki kromosomi otpadaju nakon procesa biranja (selekcije). Križanje i mutacija u populaciju uvode potpuno nove brojeve (kromosome). Obično se prvo izvodi operacija križanja. Za njeno izvođenje također je potrebno ispisati slučajne brojeve iz raspona [0,…,1] za sve kromosome. Ako je slučajni broj *r* za *i*-ti kromosom manji od vjerojatnosti križanja p_c ($r_i < p_c = 0{,}25$) taj kromosom se podvrgava operaciji križanja. Za ispisane slučajne brojeve odabani su slijedeći kromosomi prikazani u tablici 3.3.

TABLICA 3.3. ODABIR KROMOSOMA ZA OPERACIJU KRIŽANJA

Redni broj kromosoma u novoj populaciji:	Slučajni broj *r* :	Da li je kromosom odabran (da: +, ne: -)
1	0,82295	-
2	0,15193	+
3	0,61548	-
4	0,31468	-
5	0,34690	-
6	0,91720	-
7	0,51976	-
8	0,40115	-
9	0,60676	-
10	0,78540	-
11	0,03152	+
12	0,86992	-
13	0,16652	+
14	0,67452	-
15	0,75840	-
16	0,58189	-
17	0,38925	-
18	0,20023	+
19	0,35563	-
20	0,82693	-

Za križanje su odabrani kromosomi iz nove populacije pod rednim brojem 2, 11, 13 i 18. Iako vjerojatnost križanja predviđa pet kromosoma, odabrano je četiri radi takvog ispisa slučajnih brojeva. Za operaciju križanja potrebno je odrediti točku križanja, *pos.* Ona se također određuje ispisom slučajnih brojeva unutar raspona [1,…, 32] za ovaj slučaj. Korisno je napomenuti da se ne odabire zadnja točka kao točka križanja jer tada operacija križanja ne bi imala smisla. Odabrana je točka križanja *pos* = 9. U nastavku je prikazano križanje kromosoma. Parovi se biraju nasumično.

Par kromosoma sa točkom križanja *pos* = 9:

$$v'_2 = (100011000 \quad / 101101001 \quad 1110000011 \quad 10010\ )$$
$$v'_{11} = (111011101 \quad / 101110000 \quad 1000111110 \quad 11110\ )$$

daje novi par kromosoma (izdanaka (*offsprings*) ili djece (*children*)):

$$v''_2 = (100011000 \quad / 101110000 \quad 1000111110 \quad 11110\ )$$
$$v''_{11} = (111011101 \quad / 101101001 \quad 1110000011 \quad 10010\ )$$

Također i slijedeći par kromosoma za koji je odabrana točka križanja *pos* = 20:

$$v'_{13} = (0001010000 \quad 10010101 \quad 00\ / 1010111111 \quad 011\ )$$
$$v'_{18} = (1110111110 \quad 10001000 \quad 11\ / 0000001000 \quad 110\ )$$

daje novi par kromosoma:

$$v''_{13} = (0001010000 \quad 10010101 \quad 00\ / 0000001000 \quad 110\ )$$
$$v''_{18} = (1110111110 \quad 10001000 \quad 11\ / 1010111111 \quad 011\ )$$

Potrebno je izdanke uvrstiti u populaciju, čime se dobije populacija sa četiri nova kromosoma. Na toj se populaciji još treba izvršiti operacija mutacije, što će neznatno izmijeniti samu populaciju. Vjerojatnost mutacije je $p_m = 0{,}01$, što znači da će oko 1 % znamenki od ukupno *m* x *pop_size* = 660, dakle oko šest znamenki, u cijeloj populaciji biti promijenjeno u suprotnu znamenku. Operacija mutacije izvodi se tako da se za svaku znamenku ispišu slučajni brojevi (dakle 660 slučajnih brojeva) unutar raspona [0,…,1] i ako je $r < p_m = 0{,}01$ znamenka se mijenja (mutira u suprotnu znamenku). Ovakva operacija mutiranja naziva se jednolika (uniformna) jer svaka znamenka od *m* x *pop_size* ima jednaku vjerojatnost mutiranja. Mutacija se dakle odvija prema zakonu jednolike razdiobe. Također je moguće mutaciju izvršiti prema zakonu normalne razdiobe, što znači da će neke znamenke imati veću vjerojatnost mutiranja. Takav način mutacije objašnjen je u točki 3.2.2.5. DIPLOMSKOG RADA. U nastavku je prikazana nova populacija u prvom koraku, dobivena nakon križanja odabranih kromosoma, i operacija mutacije znamenki te populacije u TABLICI 3.4.

Rezultati ispisa slučajnih brojeva i usporedba sa vjerojatnosti mutacije u TABLICI 3.4 prikazuju koje znamenke mutiraju (redni broj znamenke u kromosomu). Da bi se znalo u koje točno znamenke mutiraju one koje su odabrane, mora se pronaći točno mjesto mutacije. TABLICA 3.4 prikazuje redni broj kromosoma i redni broj znamenke koja mutira u tom kromosomu.

v'_1 = (0110011111 10110101 1000011011 11000)

v''_2 = (1000110001 01110000 1000111110 11110)

v'_3 = (0010001000 00110101 1110110111 11011)

v'_4 = (0110011111 10110101 1000011011 11000)

v'_5 = (0001010100 11111111 1100001100 01100)

v'_6 = (1000110001 01101001 1110000011 10010)

v'_7 = (1110111011 01110000 1000111110 11110)

v'_8 = (0001110110 01010011 0101111110 00101)

v'_9 = (0110011111 10110101 1000011011 11000)

v'_{10} = (0000100000 11001000 0010101110 11101)

v''_{11} = (1110111011 01101001 1110000011 10010)

v'_{12} = (0100000001 01100010 1100000011 11100)

v''_{13} = (0001010000 10010101 0000000010 00110)

v'_{14} = (1000011000 01110100 0101101011 00111)

v'_{15} = (1011100101 10011110 0110001011 11110)

v'_{16} = (1001101000 00001111 1110100110 11111)

v'_{17} = (0000011110 00110000 0110100001 11011)

v''_{18} = (1110111110 10001000 1110101111 11011)

v'_{19} = (1110111011 01110000 1000111110 11110)

v'_{20} = (1100111100 00011111 1000011010 01011)

TABLICA 3.4. OPERACIJA MUTACIJE

Pozicija Znamenke:	Slučajni broj:	Redni broj kromosoma:	Redni broj znamenke u kromosomu:
112	0,000213	4	13
349	0,009945	11	19
418	0,008809	13	22
429	0,005425	13	33
602	0,002836	19	8

Nakon operacije mutacije završen je proces evolucije na prvoj generaciji (prvom iterativnom koraku). Konačan izgled populacije u prvom koraku prikazan je u nastavku. Mutirane znamenke su masno otisnute. Novonastala populacija polazna je točka za slijedeći korak genetskog algoritma, odnosno tako izgleda slijedeća generacija. Za konačno rješenje optimizacijskog problema potrebno je oko 1000 koraka, odnosno 1000 generacija. Broj koraka ovisi o tipu populacije i postavkama genetskog algoritma. Odabirom pravih parametara za određeni optimizacijski problem bitno se poboljšava izvedba algoritma.

v_1 = (0110011111 10110101 1000011011 11000)

v_2 = (1000110001 01110000 1000111110 11110)

v_3 = (0010001000 00110101 1110110111 11011)

v_4 = (0110011111 10**0**10101 1000011011 11000)

v_5 = (0001010100 11111111 1100001100 01100)

v_6 = (1000110001 01101001 1110000011 10010)

v_7 = (1110111011 01110000 1000111110 11110)

v_8 = (0001110110 01010011 0101111110 00101)

v_9 = (0110011111 10110101 1000011011 11000)

v_{10} = (0000100000 11001000 0010101110 11101)

v_{11} = (1110111011 01101001 **0**1100000111 0010)

v_{12} = (0100000001 01100010 1100000011 11100)

v_{13} = (0001010000 10010101 000 **1**0000100011 **1**)

v_{14} = (1000011000 01110100 0101101011 00111)

v_{15} = (1011100101 10011110 0110001011 11110)

v_{16} = (1001101000 00001111 1110100110 11111)

v_{17} = (0000011110 00110000 0110100001 11011)

v_{18} = (1110111110 10001000 1110101111 11011)

v_{19} = (1110111 **1**1101110000 1000111110 11110)

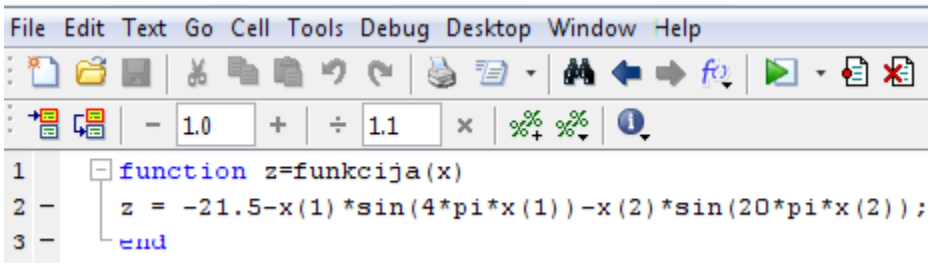
v_{20} = (1100111100 00011111 1000011010 01011)

Kroz evolucijski proces od 1000 koraka (razmatrano je dakle 1000 generacija), svaka populacija sastojala se od 20 kromosoma koji su davali sve veće vrijednosti funkcije cilja. Ipak ne znači da je najbolje rješenje (kromosom) u posljednjoj, tisućitoj, populaciji. Prilikom pregleda svih 1000 koraka za ovaj primjer ustanovljeno je da je u 396. koraku, odnosno populaciji, odabran kromosom koji je dao najveću vrijednost funkcije cilja. Ta vrijednost bila je $f(x_1, x_2) = 38{,}8275$, što je veće od maksimalne vrijednosti funkcije cilja u tisućitoj populaciji koja iznosi $f(x_1, x_2) = 35{,}4779$. Općenito se u genetskom algoritmu najbolje vrijednosti funkcije cilja u trenutnoj populaciji izdvajaju sa strane. Ta operacija predstavlja elitizam. Na elitnim jedinkama ne izvode se bilo kakve operacije genetskog algoritma. Tako je moguće sačuvati te jedinke umjesto da se izgube ili izmijene u procesu evolucije.

3.6. Problem traženja maksimalne vrijednosti na području definicije za periodički oblik funkcije cilja u MATLAB-u

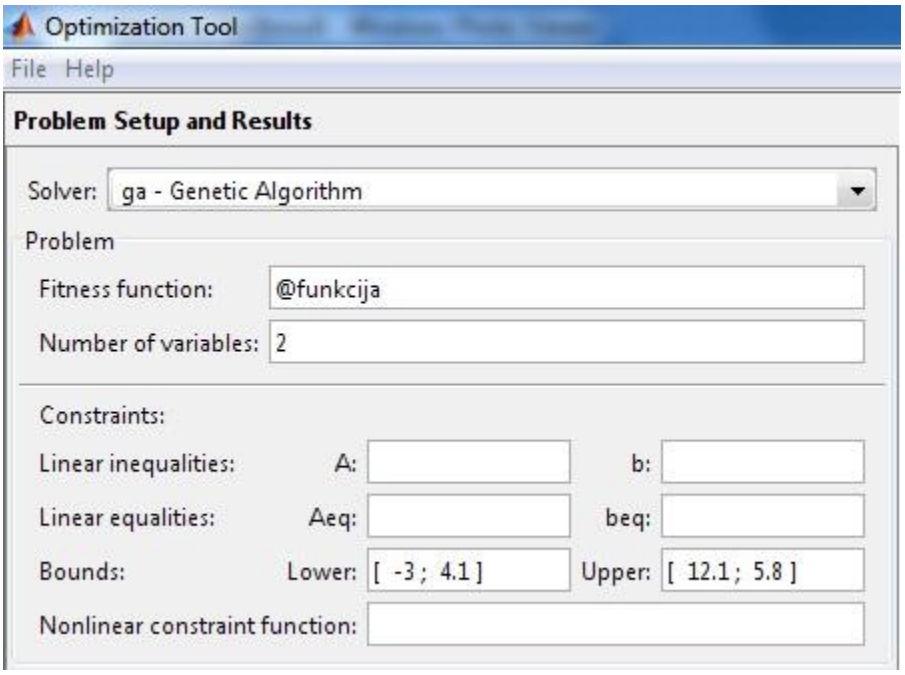
Na istom primjeru periodičke funkcije cilja prikazati će se izvedba genetskog algoritma u MATLAB-u. Sada će biti potrebno mnogo manje koraka u algoritmu, zbog prilagodbe postavki algoritma i zbog prikladnijeg tipa populacije. Domena je prikazana sa dvostrukim vektorom.

Za pokretanje genetskog algoritma potrebno je u M-datoteci napisati funkciju cilja (*fitness function*). Funkcija cilja treba imati svoje ime, npr. *funkcija*, kako bi se mogla pozvati. Kako je programskim paketom MATLAB moguće pretraživati samo minimum funkcije cilja, potrebno je uvesti negativan predznak. Tako će se dobiti obrnuti oblik funkcije cilja i optimum koji se pronađe genetskim algoritmom, a to je minimalna vrijednost funkcije cilja na području definicije, biti će zapravo maksimalna vrijednost funkcije cilja nakon poništavanja negativnog predznaka. Sintaksa u M-datoteci prikazana je na slici 3.11.



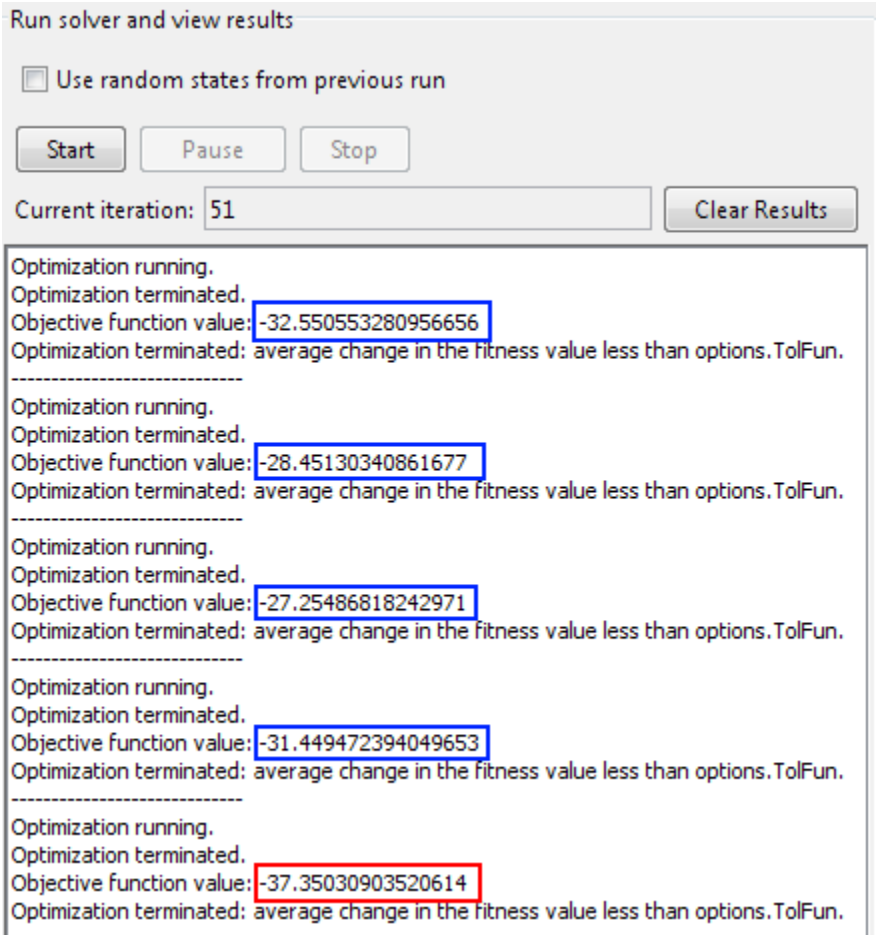
Slika 3.11. Upis periodičke funkcije cilja - sinusoide u M-datoteci

Nakon definiranja funkcije cilja potrebno pokrenuti optimizacijski alat u MATLAB-u, te unijeti i odgovarajuću sintaksu u obrazac „**Problem Setup and Results**“ unutar prozora optimizacijskog alata. Odabire se *Solver* genetski algoritmi i unosi se funkcija cilja pod nazivom @funkcija, te ograničenja optimizacijskog problema. U ovom slučaju ograničenja su samo granice (donje i gornje) dvaju područja definicije (dviju varijabli) funkcije cilja. Unošenje sintakse funkcije cilja i granica na predviđeno mjesto prikazano je na slici 3.12.



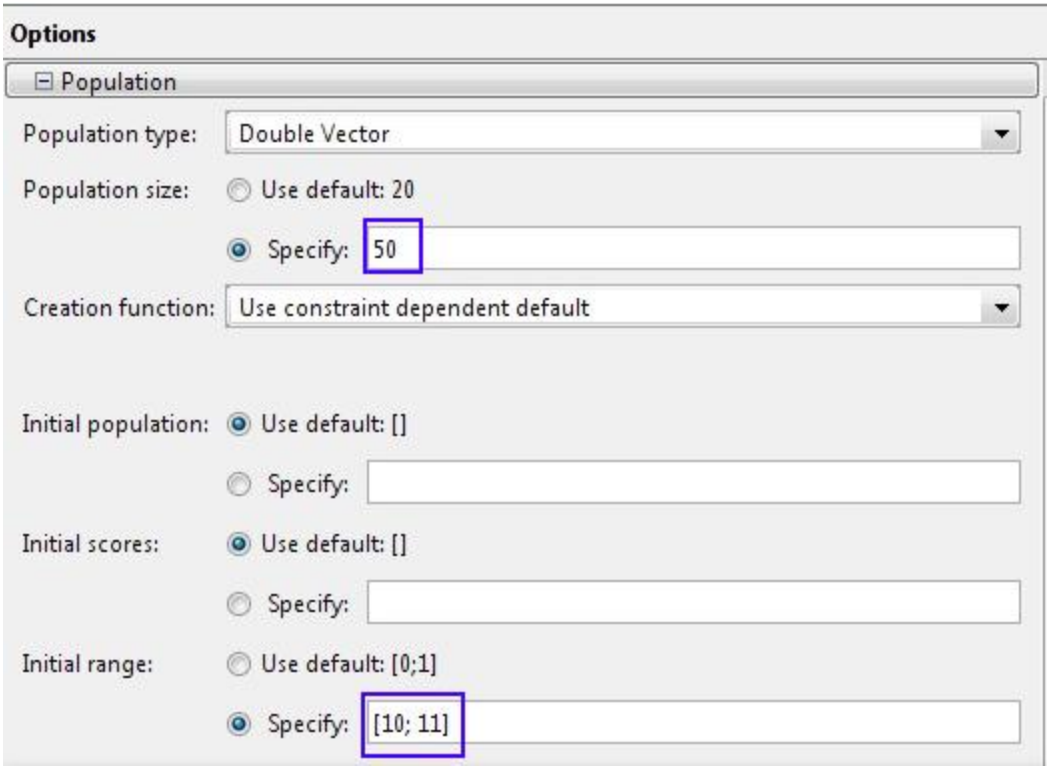
Slika 3.12. Upis sintakse funkcije cilja i ograničenja u optimizacijskom problemu

Nakon definiranja optimizacijskog problema pokrenut je genetski algoritam sa uobičajenim postavkama. Algoritam je pokrenut pet puta i svaki put je dobiveno različito rješenje za optimum. Počevši od vrijednosti optimuma: $\max f(x_1 = 5{,}226\,,\ x_2 = 5{,}255) = 32{,}55055$ do vrijednosti $\max f(x_1 = 10{,}126\,, x_2 = 5{,}725) = 37{,}350309$. Ovo je relativno dobro rješenje, ali kako je jednostavnim genetskim algoritmom predstavljenim u točki 3.4. DR-a u jednom od koraka algoritma odabran kromosom (ili jedinka) koja ima vrijednost funkcije cilja $f(x_1, x_2) = 38{,}8275$, što je veće od svih pet optimuma ovdje, logično je pokušati promijeniti postavke algoritma i pokušati dobiti bolje rješenje.



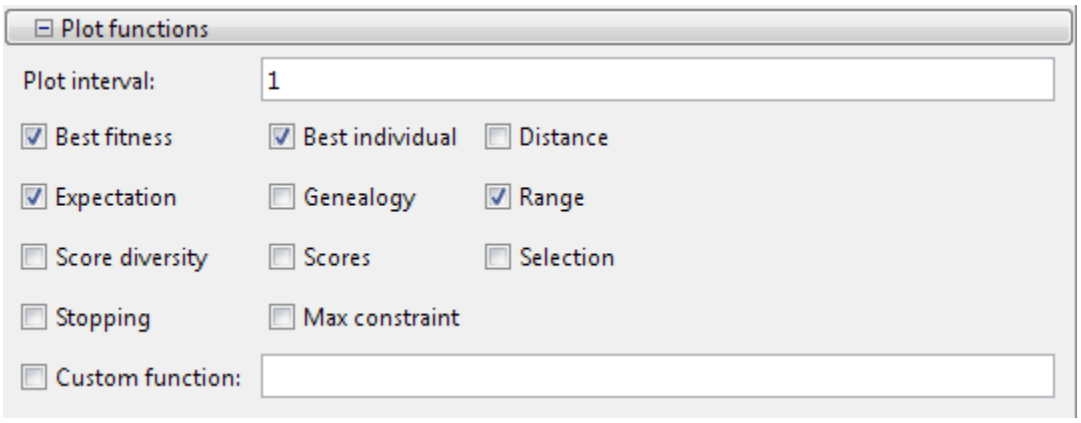
Slika 3.13. Optimumi dobiveni genetskim algoritmom iz više pokušaja

Zbog poboljšanja izvedbe algoritma promijenjene su postavke populacije. Promjena postavki algoritma obavlja se u obrascu O*ptions*. Povećan je broj jedinki populacije (*Population size*) sa 20 na 50, zatim je promijenjen raspon za jedinke početne populacije (*Initial range*) sa uobičajenog [0; 1] na novi [10; 11]. Promjenom raspona početne populacije pretraživanje jedinki je usmjereno na područje definicije između vrijednosti 10 i 11, za obje varijable. Iako varijabla x_2 nije definirana izvan područja [4.1; 5.8] i definiranjem ovakvog raspona početne populacije čini se prekršaj za varijablu x_2 , ali to se ispravlja funkcijom izrade početne populacije (*Creation function*). Promjena postavki populacije prikazana je na slici 3.14.



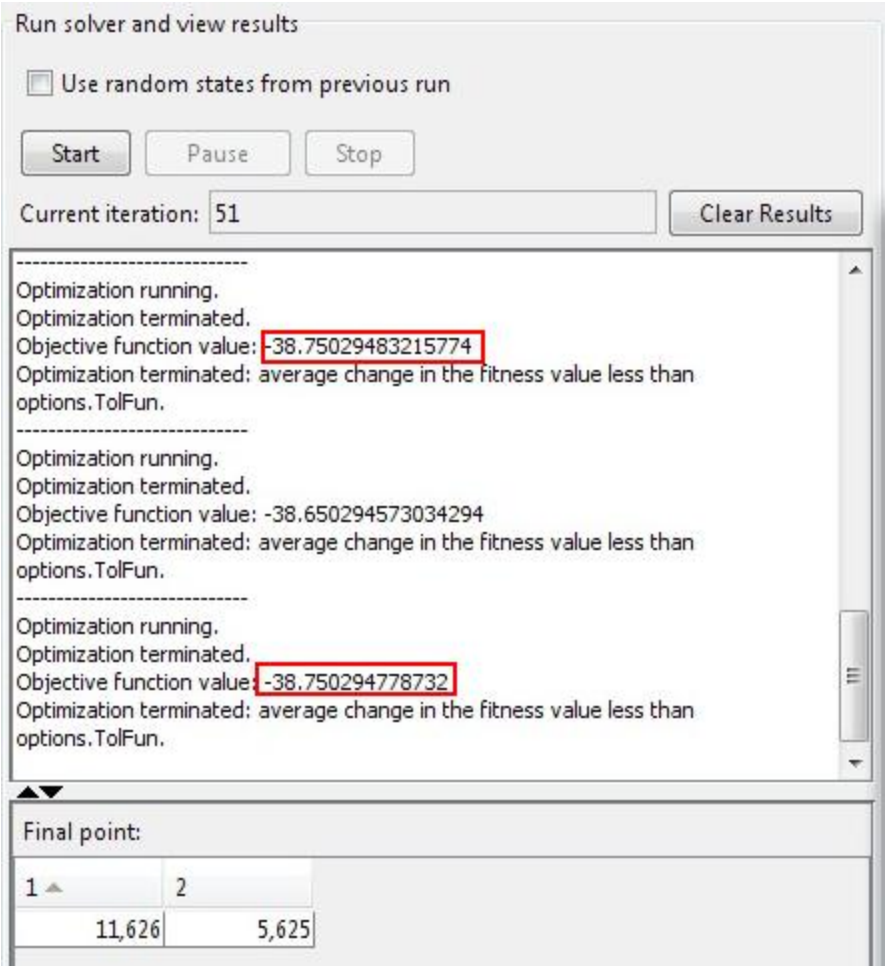
Slika 3.14. Promjena postavki populacije

Također se odabire više funkcija ispisa rezultata radi boljeg uvida u tijek genetskog algoritma. Odabrane su funkcije ispisa vrijednosti najjačih jedinki, ispis jedinke koja daje optimum, očekivanje i raspon. Rezultati se grafički ispisuju nakon svakog izvedenog koraka.



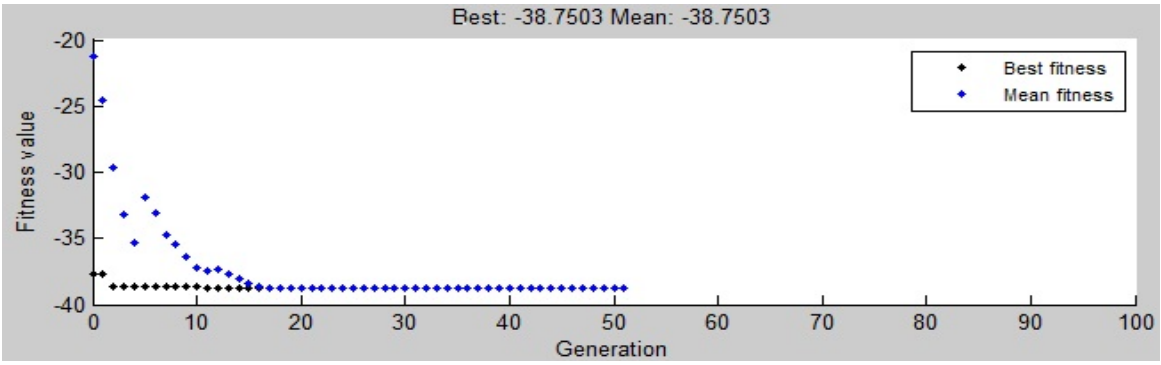
Slika 3.15. Odabir funkcija ispisa rezultata izvedbe algoritma

Ponovnim pokretanjem genetskog algoritma sa izmijenjenim postavkama dobiju se bolja rješenja. Sada je maksimum funkcije cilja veći nego u prethodnom slučaju. Optimum funkcije cilja na području definicije je $\max f(x_1 = 11{,}626\,;\, x_2 = 5{,}625) = 38{,}750294$, što je prikazano na slici 3.16. Optimum je dobiven iz više pokušaja, a konačna izvedba je trajala 51 iteraciju.



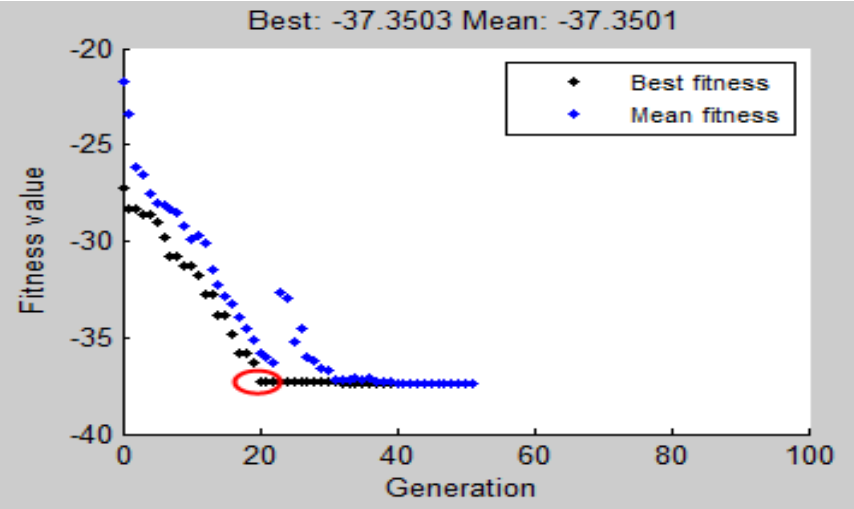
Slika 3.16. Optimum funkcije cilja na području definicije

Dobro je grafički prikazati izvedbu algoritma kroz korake, gdje se jednostavno uočava konvergencija, odnosno napredak algoritma prema optimumu. Na slici 3.17 je prikazan graf koji na ordinati ima vrijednost funkcije cilja najboljih jedinki (*Fitnes value*), a na apscisnoj osi broj generacija (*Generation*). Iz tog grafa se vidi da je već u trećoj generaciji pronađena vrijednost blizu optimuma, te da je algoritam morao biti zaustavljen zbog male razlike u rješenjima.



Slika 3.17. Grafički prikaz vrijednost najjačih jedinki po generacijama

Za usporedbu između dvije izvedbe genetskog algoritma, one sa uobičajenim postavkama i one sa izmijenjenim postavkama, na slici 3.18 grafički su prikazani rezultati izvedbe sa uobičajenim postavkama. Uočljivo je sporije napredovanje algoritma jer se oko 20. generacije pronašlo rješenje blizu optimuma. Vidjeti zaokruženu vrijednost (crveno) u grafu.

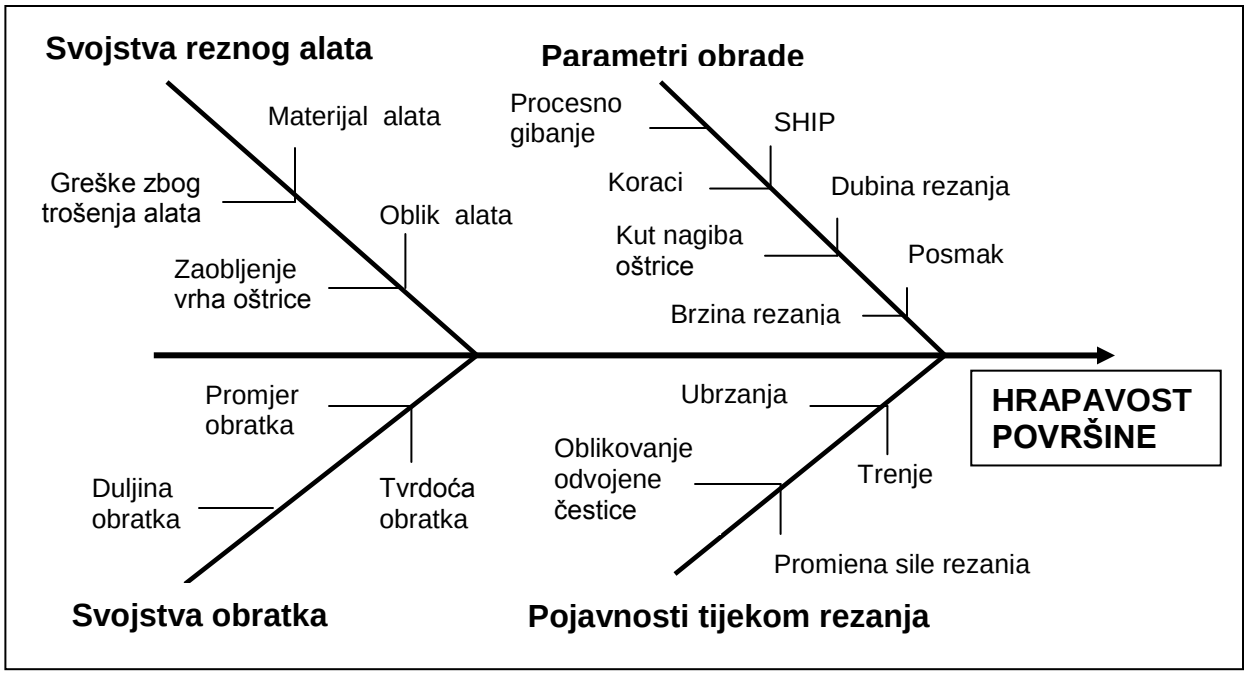


Slika 3.18. Grafički prikaz rezultata izvedbe algoritma sa uobičajenim postavkama

4.0. PRIMJENA GENETSKOG ALGORITMA U STROJARSKOJ PRAKSI

4.1. Opis problema određivanja optimalnih parametara obrade u cilju ostvarivanja minimalne hrapavosti površine [6]

U ovoj točki DIPLOMSKOG RADA promatra se problem optimizacije parametara obrade u cilju minimizacije hrapavosti površine. Odabran tehnološki proces je završno glodanje (end milling) titanovih slitina. Uz parametre obrade, brzinu rezanja v_c i posmak po zubu f, promatra se i kut oštrice alata γ. Izostavljena je dubina obrade. Osim ova tri spomenuta parametra postoje i drugi koji utječu na hrapavost površine, a prikazani su na slici 4.1.



Slika 4.1. Ishikava dijagram - utjecajni parametri na hrapavost površine

Promatrani optimizacijski problem je dakle traženje minimuma hrapavosti površine uz zadana ograničenja na promjenjive vrijednosti parametara. Funkcija cilja i ograničenja dobiveni su planom pokusa u navedenom znanstvenom članku. Odabran je najbolji regresijski model za formuliranje funkcije cilja. Matematički model optimizacijskog problema je sljedeći:
funkcija cilja:

$$\min R_a(v, f, \gamma) = 0{,}237 - 0{,}00175 \cdot v + 8{,}693 \cdot f - 0{,}00159 \cdot \gamma \qquad (4.1)$$

i ograničenja u obliku donjih i gornjih granica:

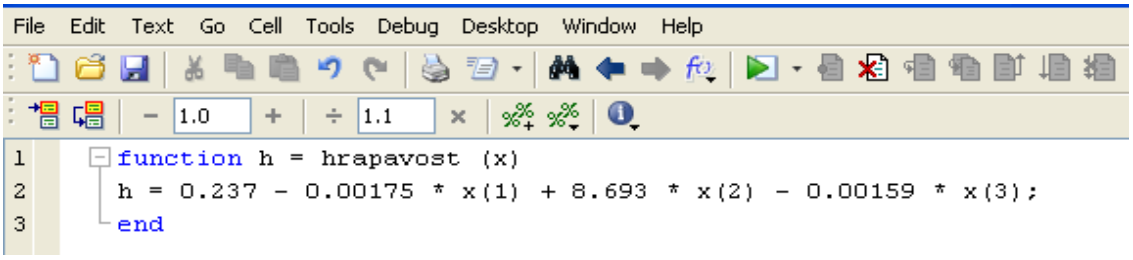
$$124{,}53 \leq v \leq 167{,}03$$
$$0{,}025 \leq f \leq 0{,}083$$
$$6{,}2 \leq \gamma \leq 14{,}8 \qquad (4.2)$$

gdje su v [m/min] brzina rezanja, f [mm/zub] posmak po zubu i γ [°] kut nagiba rezne oštrice.

Pomoću genetskog algoritma određeni su optimalni parametri, najveća brzina rezanja, najmanji posmak i najveći kut nagiba rezne oštrice, kojima se postiže minimalna hrapavost površine.

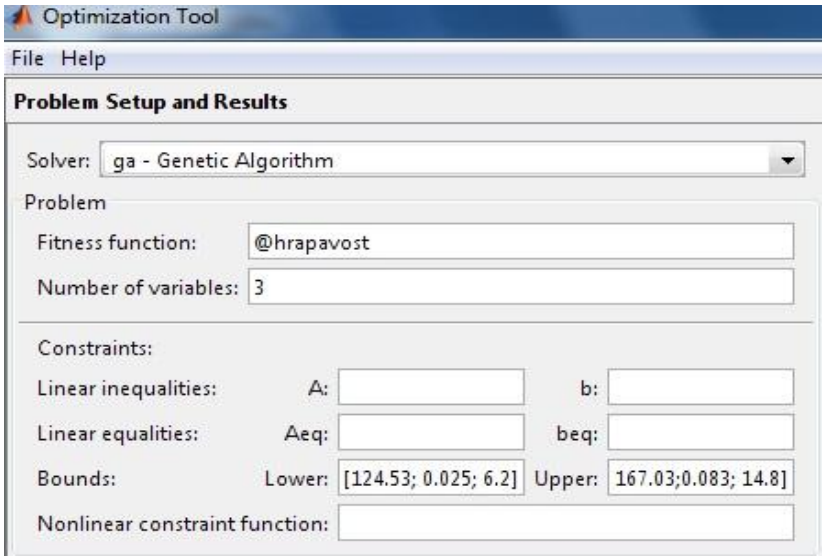
4.2. Optimizacija parametara obrade pomoću genetskog algoritma u problemu ostvarivanja minimalne hrapavosti površine

Za pokretanje genetskog algoritma potrebno je u M-datoteci napisati funkciju cilja (*fitness function*). Sintaksa u M-datoteci prikazana je na slici 4.2.



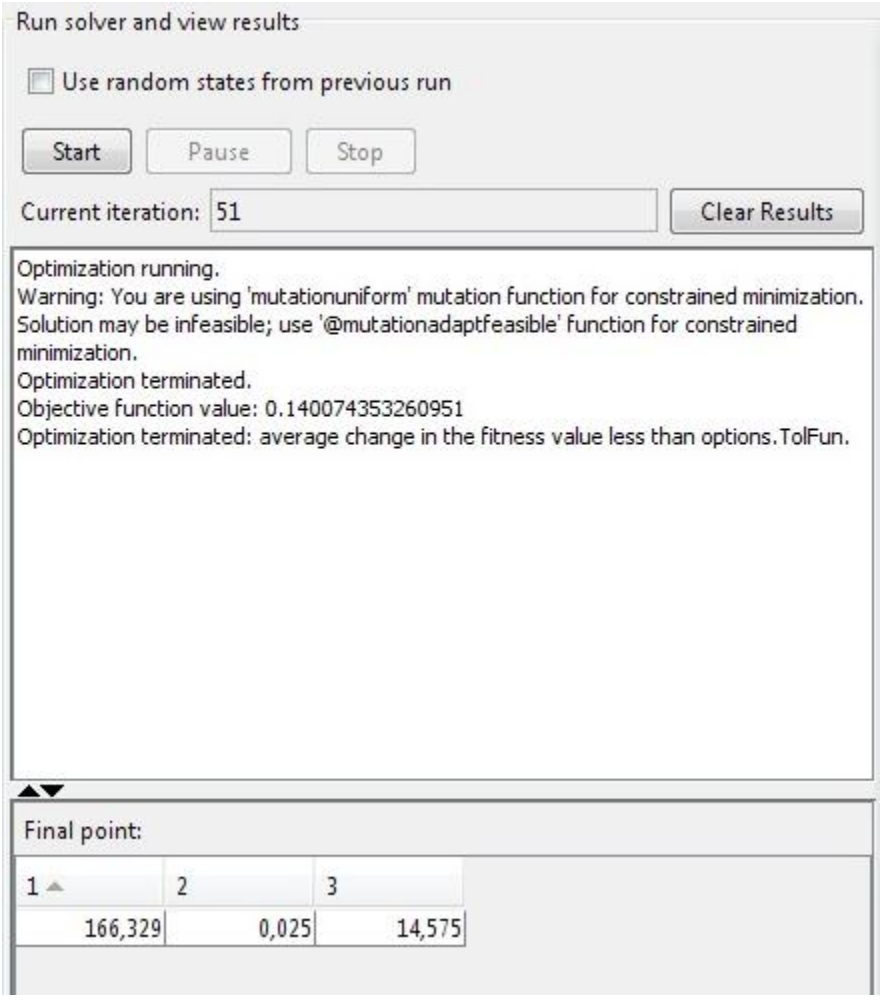
Slika 4.2. Prikaz funkcije cilja u M-datoteci

Nakon toga, već opisanim postupkom u točki 3. DIPLOMSKOG RADA, definira se optimizacijski problem u prozoru *optimization tool*. Odabire se način rješavanja problema (*Solver*): ga - Genetic Algorithm. Upisuje se funkcija cilja nazvana hrapavost pomoću sintakse: @hrapavost. Definira se broj varijabli funkcije cilja, što je u ovom slučaju tri. Potom se definiraju ograničenja (*Bounds*) kao donja (*Lower*) sintaksom: [124.53; 0.025; 6.2] i gornja (*Upper*) sintaksom: [167.03; 0.083; 14.8]. Vidjeti sliku 4.3. U ovom primjeru nema linearnih i nelinearnih ograničenja.



Slika 4.3. Formuliranje optimizacijskog problema

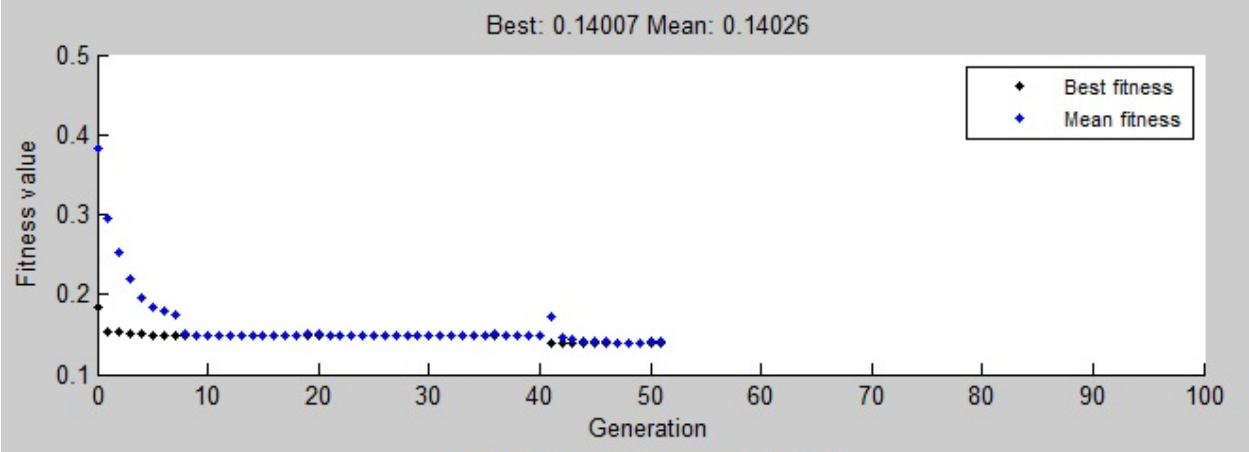
Nakon formulacije problema može se pokrenuti algoritam. Rezultati prvog pokretanja prikazani su na slici 4.4. Uočljive su na dnu prozora vrijednosti varijabli funkcije cilja za koju je postignuta optimalna vrijednost u prvoj izvedbi. Varijable su redom za brzinu rezanja $v = 166{,}329\ \mathrm{m/min}$, za posmak po zubu $f = 0{,}025\ \mathrm{mm/zubu}$, te za kut nagiba oštrice $\gamma = 14{,}575°$. Također se može vidjeti broj iteracija genetskog algoritma koje su bile potrebne za postizanje tog rješenja, a to je 51. Te konačno u središnjem dijelu prozora i minimalna vrijednost funkcije cilja, a to je $\min R_a = 0{,}140074353260951$. Ovoliko izražena preciznost za hrapavost nije potrebna, dovoljno ju je izraziti do treće decimale, stoga je $\min R_a = 0{,}140\ \mu\mathrm{m}$.



Slika 4.4. Prikaz rezultata nakon prvog pokretanja algoritma

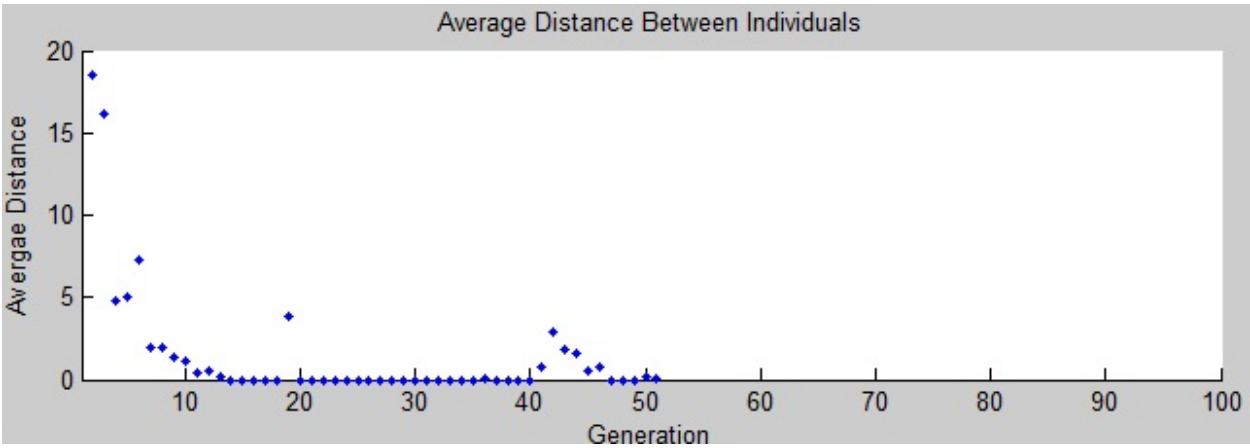
Također je korisno prikazati rezultate tijekom izvođenja algoritma, odnosno prikazati povijest izvedbe. Moguće je prikazati određene karakteristike izvedbe nakon svake iteracije. To se odabire u postavkama ispisa (*Plot functions*) u dijelu Options. Na slici 4.5. prikazane su vrijednosti najboljih jedinki i prosječne vrijednosti za svaku generaciju. Najbolja jedinka (*Best fitness*) kroz cijelu evoluciju (51 generaciju) ima vrijednost 0,14007, što je ujedno i minimum funkcije cilja, i dobivena je u 41. generaciji. Od tog trenutka pa do kraja izvođenja razlike u vrijednostima funkcije cilja najboljih jedinki u generacijama nisu se razlikovale više od vrijednosti

koja je definirana u postavkama (*Tolerance*). Zato je algoritam prekinuo s radom nakon određenog broja koraka.



Slika 4.5. Prikaz vrijednosti funkcije cilja kroz cijelu izvedbu algoritma

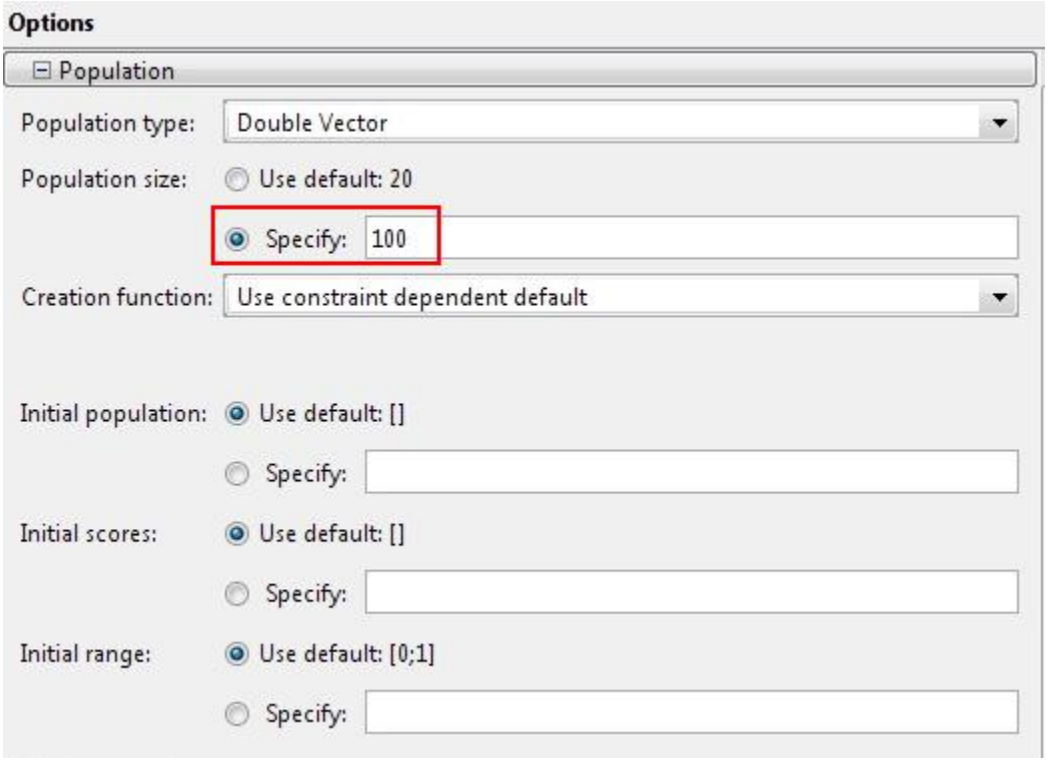
Također je korisno prikazati prosječnu udaljenost između jedinki u svakoj generaciji, slika 4.6. Ova vrijednost može poslužiti pri procjeni kakvo je područje domene pretraživano algoritmom. Ako su jedinke u populaciji bile više udaljene jedna od druge pretraživano je šire područje domene, ako je pak ta udaljenost bila manja odvijala se konvergencija algoritma. Iz grafa se može uočiti kako je prosječna udaljenost sve manja kako napreduje algoritam. To znači da algoritam konvergira prema konačnom rješenju.



Slika 4.6. Prikaz prosječne udaljenosti između jedinki kroz cijelu izvedbu algoritma

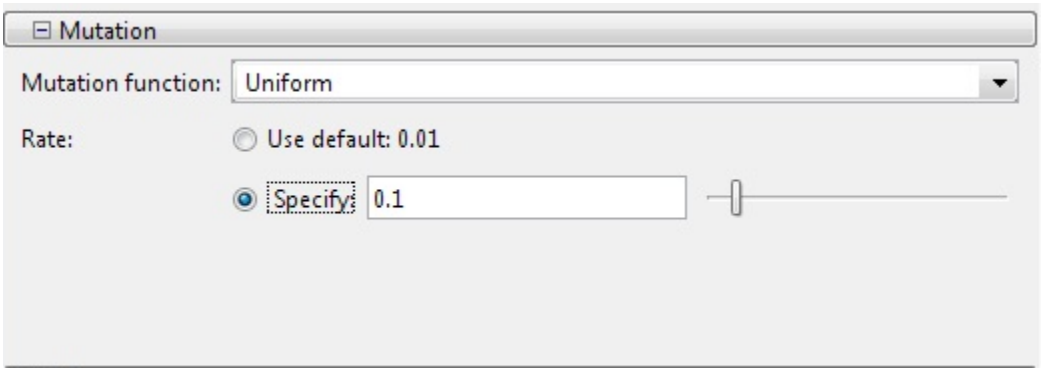
Kako bilo još uvijek se ne može reći da li je algoritam konvergirao prema globalnom optimumu. Iz uzastopnih pokretanja algoritma može se uočiti kako se pri svakom sljedećem pokretanju dobivaju drukčija rješenja. Ta rješenja, odnosno vrijednosti funkcije cilja, mogu se razlikovati neznatno (na npr. trećoj decimali) ili pak znatnije. Za poboljšanje rezultata promijeniti

će se postavke algoritma i to na temelju pokušaja i promašaja. Nakon nekoliko isprobavanja rada algoritma s različitim postavkama odabrane su sljedeće postavke:
Promijenjena je veličina populacije u postavkama populacije (*Population*) sa 20 (uobičajena vrijednost) na 100, slika 4.7.



Slika 4.7. Promjena veličine populacije

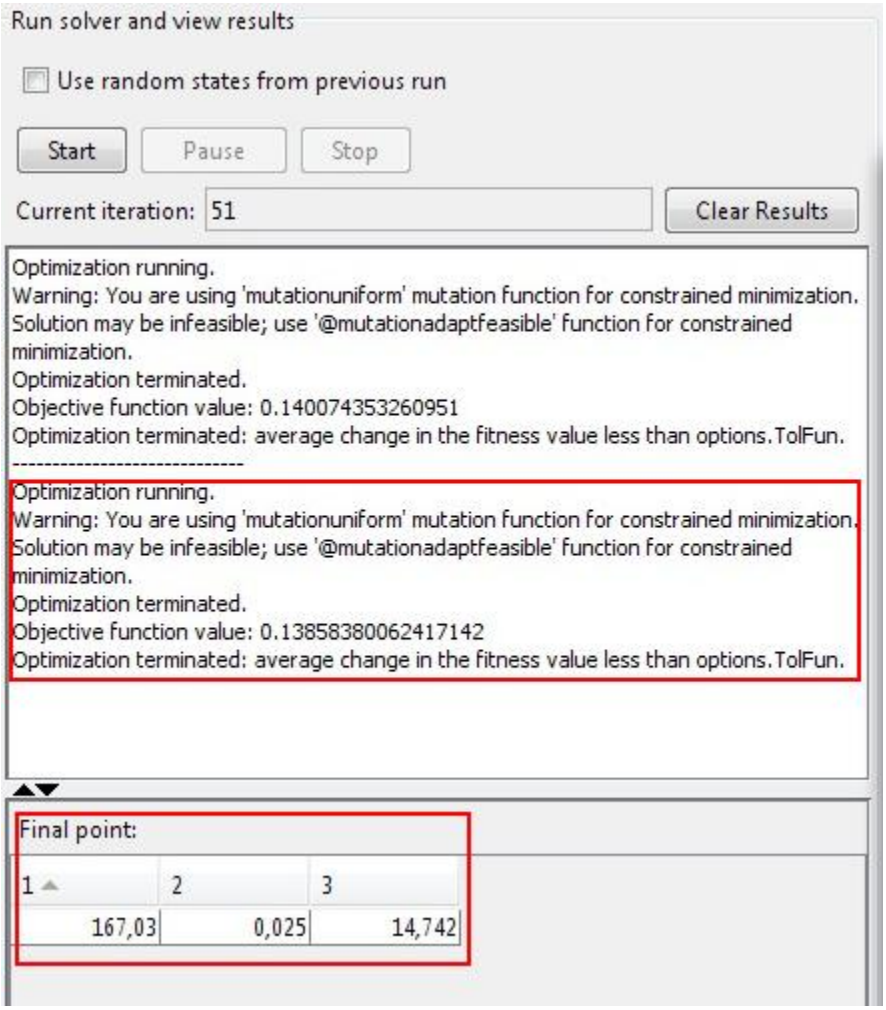
Zatim je odabrana prilagodljiva funkcija mutacije koja daje moguća rješenja (*Adapt feasible*) i stupanj (ili vjerojatnost) mutacije je postavljen na 0.1, slika 4.8.



Slika 4.8. Promjena funkcije mutacije i stupnja mutiranja

Nakon odabira novih postavki pokrenut je algoritam i dobiveni su malo bolji rezultati. Slika 4.9. Vrijednost funkcije cilja smanjena je na $\min R_a = 0{,}13858380062417142$, odnosno $\min R_a =$

$0{,}1385\ \mu\mathrm{m}$. Vrijednosti parametara su redom za brzinu rezanja $v = 167{,}03\ \mathrm{m/min}$, za posmak po zubu $f = 0{,}025\ \mathrm{mm/zubu}$, te za kut nagiba oštrice $\gamma = 14{,}742°$.



Slika 4.9. Pokretanje algoritma sa izmijenjenim postavkama, konačno rješenje

5.0. PROCJENA PREDNOSTI PRIMJENE GENETSKIH ALGORITAMA U ODNOSU NA KLASIČNE METODE OPTIMIRANJA NELINEARNIH FUNKCIJA CILJA

5.1. Sažetak načina rada genetskog algoritma [5]

Genetski algoritam je metoda rješavanja matematički definiranih optimizacijskih problema, sa ili bez ograničenja, na principu željenog odabira. Ograničenja se mogu javiti u obliku linearnih ili nelinearnih jednadžbi odnosno nejednadžbi i ona diktiraju područje (domenu) djelovanja algoritma. Algoritam je termin „genetički“ dobio zbog paralele ili sličnosti sa principima *prirodne selekcije*. Prema njima, evolucija vrsta prirodnom svijetu, odnosno prenošenje gena (tjelesnih i psiholoških karakteristika) sa roditelja na djecu se odvija ovisno o želji roditelja. Stoga se povlači paralela u genetskim algoritmima, te se karakteristike (geni) realnih brojeva varijabli funkcije cilja prenose u slijedeće iterativne korake algoritma (generacije), a evolucija rješenja se odvija prema želji korisnika (osobe koja oblikuje postavke algoritma s ciljem pronalaženja optimuma funkcije cilja). U svakom slijedećem koraku populacija mogućih rješenja (jedinki) se mijenja. Promjena populacije se odvija slučajnim odabirom izdvojenih jedinki od kojih zatim nastaju nove jedinke (izdanci) za slijedeću generaciju. Izdvajanje se izvodi na temelju ocjenjivanja cijele populacije u generaciji (iterativnom koraku), prema kriteriju najveće ili najmanje vrijednosti funkcije cilja. Slučajnost je potrebna kako bi se izbjegla prerana konvergencija prema lokalnom optimumu. Genetski algoritam je heuristička metoda, što znači da se njena konvergencija prema globalnom optimumu ne može potkrijepiti teoretskim postavkama, već samo na uspješno riješenim primjerima.

Genetski algoritmi imaju tri glavna tipa pravila za dobivanje slijedeće generacije:

- Selekcija: Pravila za odabir jedinki, *roditelja*, koje pridonose populaciji u slijedećoj generaciji (iterativnom koraku),
- Križanje: Pravila za kombiniranje *roditelja* za dobivanje novih jedinki, *djece*.
- Mutacija: Pravila za promjenu (mutiranje) karakteristika jedinke, od kojih nova jedinka (izdanak) nastaje iz jedne jedinke, a ne dvije kao u slučaju križanja.

Genetski algoritmi se primjenjuju na raznim optimizacijskim problemima koji nisu prikladni za rješavanje sa standardnim algoritmima (algoritmima klasičnih numeričkih metoda). Tu su uključeni problemi u kojima je funkcija cilja isprekidana (diskontinuirana), nederivabilna na nekom intervalu, stohastička, ili ima veliki stupanj nelinearnosti (što otežava deriviranje).

5.2. Sažetak načina rada algoritma klasičnih numeričkih metoda

Svi algoritmi klasičnih numeričkih metoda traženja optimuma počivaju na sličnim načelima, a to su slijedeći:

- Izabrati početnu točku $x^{(0)}$ u slučaju funkcije jedne varijable, ili $\mathbf{x}^{(0)}$ u slučaju funkcije više varijabli, iz područja definicije funkcije cilja uzevši u obzir ograničenja.
- Izračunati vrijednost funkcije u toj točki.
- Odrediti derivacije funkcije (parcijalne derivacije u slučaju funkcije više varijabli), odnosno odrediti gradijent funkcije u promatranoj točki radi usmjeravanja rješenja prema najstrmijem nagibu funkcije (osigurana konvergentnost prema optimumu).
- Odrediti veličinu slijedećeg koraka (stopu promjene vrijednosti varijable) Δx u slučaju funkcije jedne varijable, ili vektor **s** u slučaju funkcije više varijabli, u cilju pretrage drugih točaka domene funkcije cilja.
- Odrediti stupanj promjene vrijednosti funkcije cilja α, ako postoji, kako se ne bi ostalo na istoj nivo liniji (konturi).
- Odrediti vrijednost ε unutar koje se razlika dva rješenja (vrijednosti funkcije cilja) mora nalaziti da bi se algoritam zaustavio.
- Izabrati slijedeću točku $x^{(1)}$ (ili $\mathbf{x}^{(1)}$), odnosno ponoviti cijeli korak algoritma za slijedeću točku $x^{(i)}$ (ili $\mathbf{x}^{(i)}$).

Klasični algoritmi su uglavnom temeljeni na principima diferencijalnog računa ili se područje definicije funkcije pretražuje simultano ili deterministički. Ne spadaju u heurističke postupke jer su zasnovani na matematičkoj teoriji.

5.3. Usporedba metodike genetskog algoritma i klasičnih algoritama [5]

Prvo što se može uočiti kod uspoređivanja ovih dviju metoda jest da numeričke metode svoj algoritam uvijek započinju s jednom početnom točkom, dok genetski algoritam započinje s jednom populacijom koja ovisno o postavkama može imati i preko tisuću jedinki (točaka). Genetski algoritmi su tu u velikoj prednosti u osiguranju od zaglavljivanja u lokalnom optimumu, jer se u startu pretražuje mnogo šire područje. Odabir točaka u genetskim algoritmima za populaciju početne generacije je na principima slučajnog odabira, dok se u sljedećim generacijama slučajno odabiru i mijenjaju već izdvojene jedinke. Numeričke metode slijede princip determinističkog odabira u slijedećim koracima i stoga mogu konvergirati prema lokalnom optimumu. Ova usporedba prikazana je u TABLICI 5.1.

TABLICA 5.1. USPOREDBA GENETSKIH I KLASIČNIH ALGORITAMA

KLASIČNI ALGORITMI	GENETSKI ALGORITMI
Ispitivanje jedne točke u svakoj novoj iteraciji. Približavanje optimumu odvojeno točka po točka.	Ispitivanje cijele populacije točaka u svakoj novoj iteraciji. Najbolja točka unutar populacije uzima se kao mjerilo približavanja optimumu.
Odabiranje nove točke odvojeno po principima determinističkog računanja (ovisno o gradijentu).	Odabir nove populacije koristeći principe slučajnog odabira.

6.0. NEDOSTATCI I OGRANIČENJA PRIMJENE GENETSKOG ALGORITMA

Pod pretpostavkom da su odnosi među parametrima nelinearni, prostor traženja rješenja je velik onoliko koliko ima znamenki u cijeloj populaciji. Za jedinku duljine L binarnih znamenki moguće je dobiti 2^L različitih vrijednosti. Dakle složenost pretraživanja vrijednosti jedinki optimizacijskog problema ovisi o traženoj preciznosti i ukupnom broju utjecajnih parametara, odnosno varijabli funkcije cilja.

U točki DIPLOMSKOG RADA 3.4.1 prikazana je populacija od 20 jedinki sa 32 znamenke u svakoj jedinki. To znači da svaka jedinka ima 2^{32} mogućih načina zapisa, odnosno moguće ju je opisati s toliko prirodnih brojeva. Postoji toliko različitih vrijednosti pojedine jedinke koliki je rezultat zbroja svih mogućih vrijednosti za svaku od varijabli funkcije cilja unutar jedinke (kromosoma). Broj 2^{32} je potrebno pomnožiti 20 puta i dobiti će se jasna slika o prostoru mogućih rješenja.

Uzevši u obzir da su primjeri iz strojarske prakse složeni primjeri sa mnogo utjecajnih parametara (varijabli), nekad i sa više stotina, odmah je jasno da se genetskim algoritmom ne može u realnom vremenu riješiti takve probleme. Prostor traženja povećava se dakle eksponencijalno i svakim njegovim povećanjem bitno se povećava vrijeme izvedbe ili opterećenje procesora. Stoga se pribjegava drugim oblicima ili vrstama genetskih algoritama kao što su paralelni genetski algoritmi. Međutim za njihovo izvođenje potrebna su skupa višeprocesorska računala, ili mreža računala. To podiže cijenu izvedbe, zato se mora izračunati isplativost takvog pothvata.

Također se genetskim algoritmima ne daje izrazito precizno rješenje kao što se to može dobiti analitičkim metodama. Ipak imaju prednost što funkcije cilja ne moraju biti derivabilne ili neprekidne. Zato je dobro genetske algoritme kombinirati sa klasičnim, numeričkim i analitičkim, metodama. Genetske algoritme u tom slučaju koristiti za grubo pronalaženje optimuma iz više pokušaja. Tako steći iskustvo kako se ponaša funkcija cilja na području definicije. Nakon toga upotrebom hibridne funkcije pronaći preciznije rješenje.

7.0. ZAKLJUČAK

U ovom DIPLOMSKOM RADU obrađen je uglavnom jednostavni genetski algoritam. Proteklih godina (počevši od kraja 20. stoljeća do sada) bitno se povećao broj istraživača ove tematike, stoga je i razvijeno mnogo naprednijih oblika izvedbe algoritma. To su npr. paralelni genetski algoritmi, prilagodljivi genetski algoritmi, koji imaju različite funkcije odabira jedinki i uspoređivanja istih. Svrha ovog rada nije obraditi sve to već dati detaljan opis principa, temelja, algoritma. Jer na tim temeljima razrađene su sve ostale vrste algoritama. Ovim DIPLOMSKIM RADOM prikazana je još i izvedba u softveru, što je nezaobilazan dio ove teme, jer se iterativni koraci ne rješavaju ručno, zbog opsežnosti cijele izvedbe. U izvedbu algoritma uključene su pretraga, izračunavanje, evaluacija, i druge funkcije genetskog algoritma. Dio DIPLOMSKOG RADA u kojem je obrađena izvedba u softveru dotiče se ujedno i složenijih genetskih algoritama, kao što je algoritam s migracijama, odnosno algoritmi s pod-populacijama, algoritmi s različitim funkcijama križanja, mutacije, odnosno reprodukcije itd. Važno je napomenuti da su genetski algoritmi jednostavno primjenjivi upravo zbog toga što nemaju zahtjeva za derivabilnošču funkcija cilja, zatim neprekidnosti, mogu raditi sa ili bez ograničenja. Genetski algoritmi su heuristička metoda koja se potvrđuje na velikom broju uspješno riješenih primjera, što je djelomice prikazano i u ovom radu.

8.0. LITERATURA

[1] L. Neralić, Uvod u matematičko programiranje 1, Element, Zagreb, 2008.
[2] Inženjerski priručnik, Proizvodno strojarstvo, sv. 4, Tehnička knjiga, Zagreb, 2001.
[3] M. Zbigniew, Genetic algorithms + data structures = evolution programs, Springer, Berlin, 1999.
[4] M.Golub, Genetski algoritam: 1. i 2. dio, FER, Zagreb, 2004.
[5] MATLAB Help, verzija R 2008.
[6] http:// www.sciencedirect.com, 2010.