Sveučilište u Zagrebu
Fakultet prometnih znanosti
Zavod za inteligentne transportne sustave
Katedra za primijenjeno računarstvo

Vježba:	#8
Kolegij:	Umjetna inteligencija
Tema:	Genetski algoritmi
Vježbu pripremili:	doc. dr. sc. Edouard Ivanjko Martin Gregurić, mag. ing. traff. Mario Buntić, mag. ing. traff.

Upute za izradu vježbi/zadataka

Prije dolaska na vježbu potrebno je proučiti pripremu za vježbu. Vježbe je potrebno izraditi pomoću alata koji se koriste. Vrijeme za izradu zadanih zadataka iznosi 90 minuta.

Cilj vježbe

Pregled osnovnih funkcija i njihovih svojstva za rad s genetskim algoritmom unutar skupa alata „*Genetc algorithm and Direct Search*“. Demonstracija uspješnosti pronalaska globalnog minimuma na jednostavnoj funkciji cilja u ovisnosti o promjeni svojstva genetskog algoritma. Upoznavanje s radom u GUI okruženju genetskim algoritmom uz postavljanje ograničenja i njihovog kodiranja.

Opis vježbe

Temeljna svrha vježbi je upoznavanje sa osnovnim funkcijama genetskih algoritama unutar skupa alata *„Genetic algorithm and Direct Search Toolbox*“ u MATLAB okruženju. Posebice će se staviti naglasak na svojstva strukture - „options“ koja imaju temelju ulogu u umjeravanju funkcije rada genetskog algoritma. Zatim će se prikazati funkcije rada „Optimization Tool“-a s postavljenim solverom genetičkog algoritma. Studenti će rješavati problem maksimiziranja funkcije cilja uz ograničenja pomoću „Optimization Tool“-a. Kako bi aktualizirali problematiku genetskih algoritama u transportu, studenti će primijeniti genetski algoritam na problemu trgovačkog putnika.

Genetski algoritam u programskom sustavu MATLAB

Genetski algoritmi nemaju vlastiti skup alata (engl. „toolbox“) unutar MATLAB programskog sustava iz razloga što se genetski algoritam klasificira kao jedan od optimizacijskih alata (engl. „Optimization Tool“). Optimizacijski alati, zajedno s alatom genetskog algoritma mogu se naći pod skupom alata „Genetic Algorithm and Direct Search Toolbox“.

Osnovne funkcije i naredbe genetskog algoritma u skupu alata *„Genetic Algorithm and Direct Search“*

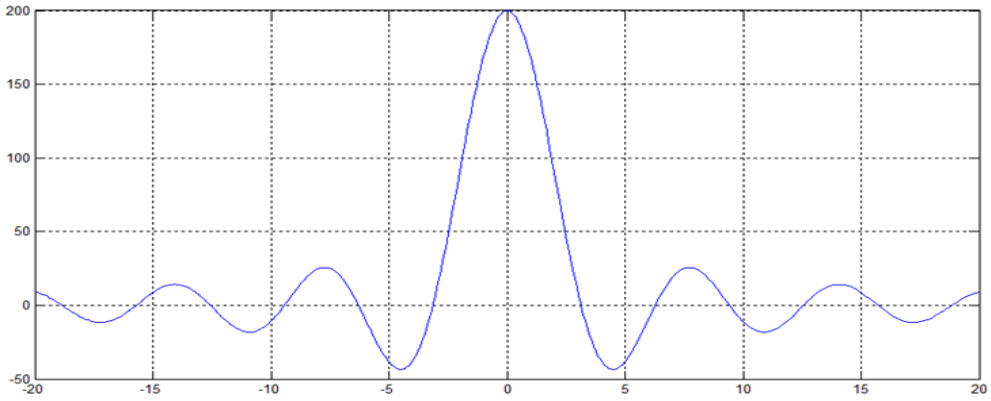
Funkcija za pokretanje genetskog algoritma koji traži minimum matematičko definirane funkcije u MATLAB programskom okruženju iz komandnog prostora je *„ga*“. Najosnovnija sintaksa ulaznih argumenata funkcije *„ga*“ je sljedeća:

- *[x fval] = ga(@fitnessfun, nvars, options, A, b, Aeq, beq)* - gdje je „@fitnessfun“ – referenca za funkciju cilja, „nvars“ – broj nezavisnih ulaznih varijabli u funkciju cilja, a argument „options„ – struktura koja sadrži postavke genetskog algoritma. Ako se ovaj argument izostavi, koriste se pretpostavljene postavke. Argumenti „A“ i „b“ opisuju ograničenja linearnih nejednadžba oblika „A*x ≤ b“, dok „Aeq“ i „beq“ argumenti označavaju linearna ograničenja jednadžba oblika „Aeq*x = beq“. Funkcija vraća vrijednost rezultata rada genetskog algoritma „fval“ u trenutak kada je dostignuta konačna vrijednost funkcije cilja „x“.

Za testiranje funkcije „ga“ za primjer se može riješiti zadatak u kojem se traži globalni maksimum funkcije cilja :

$$f(x) = 200\frac{\sin(x)}{x}$$

Grafički prikaz te funkcije cilja možemo vidjeti na slici 1.



Slika 1. Graf funkcije cilja

Primjećujemo kako naša funkcija cilja ima svoj globalni maksimum na vrijednosti „200“. Genetski algoritmi su predviđeni za pronalaženje globalnog minimuma u funkcijama (kao i sve funkcije iz skupa alata „*Genetic Algorithm and Direct Search Toolbox*“), te se ne mogu direktno koristiti za pronalaženje globalnog maksimuma funkcije cilja. Ukoliko želimo pronaći maksimum funkcije cilja, potrebno je izvršiti promjenu predznaka funkcije cilja, odnosno u ovom slučaju se traži minimum funkcije *„– f(x)“.*

Funkciju cilja ćemo smjestiti u MATLAB *m-funkciju*. M-funkciju ćemo nazvati „funkcija_cilja“:

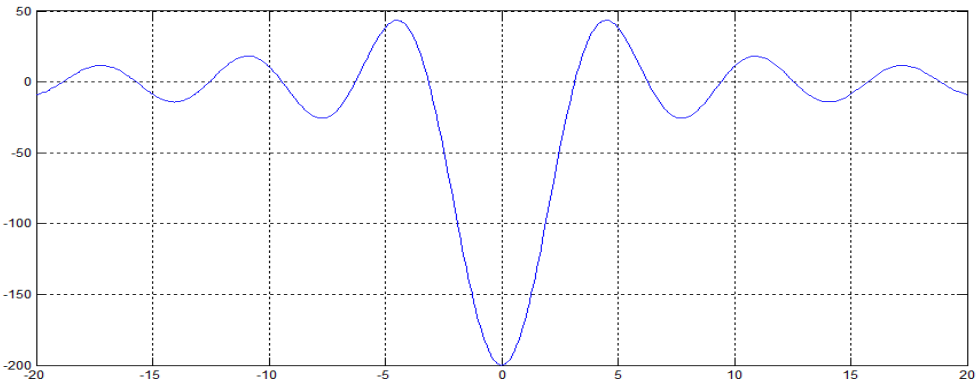
```
function y = funkcija_cilja(x)
y(:,1) = -200*sin(x(:))./(x(:));
```

Ukoliko se želi izvršiti ispis grafa funkcije koristi se funkcija „fplot“:

- *fplot(fun,limits)* – funkcija vrši ispis grafa funkcije „fun“, koji može biti i referenca funkcije koji se piše s predznakom „@“. Dok atribut „limits“ je vektor koji specificira ograničenje na x- osi grafa ([xmin xmax]) ili x i y osi grafa ([xmin xmax ymin ymax]).

Zatim u MATLAB radni prostor unesimo naredbu koja će iscrtavati graf „funkcije_cilja“ (slika 2.) (referenca - *„@funkcija_cilja“)* sa minimumom i maksimumom x–osi [-20 20]:

```
fplot(@funkcija_cilja,[-20 20])
grid on
```



Slika 2. Graf funkcije cilja s negativnim predznakom

Primjećujemo kako je sada maksimum funkcije cilja iz slike 2. postao minimum zahvaljujući promjeni predznaka funkcije cilja.

Prije nego što započnem s traženjem globalnog minimuma funkcije cilja, moramo stvoriti strukturu „options“. Ukoliko pri generiranju postavki genetskih algoritama želimo većinu postavki postaviti na standardne vrijednosti, a neke promijeniti, to možete učiniti pomoću funkcije „gaoptimset“:

- *options = gaoptimset('var1', value1, 'var2', value2,…)* – argument 'var1', označava ime svojstva genetskog algoritma, dok „value1“ označava pripadajuću vrijednost svojstva 'var1'.

U ovom slučaju neće se unositi nove vrijednosti u postavke genetskog algoritma već će se zadržati sve standardne. To se postiže funkcijom „gaoptimset“ kojoj se prosljeđuje referenca „@ga“.

```
options = gaoptimset(@ga)
```

Funkcija „gaoptimset“ je strukturu „options“ inicirala standardnim postavkama/svojstvima. Neke od važnijih standardnih svojstva genetskog algoritma koje ćemo koristit tijekom vježbe su:

- *PopulationSize: 20* - broj jedinki u populacije, inicijalna vrijednost 20 jedinki.

- *PopulationType: {'doubleVector'},* - opis tip podataka populacije (tipa „string“), inicijalna vrijednost je {'doubleVector'} što označava da je tip populacije vektor tipa „double“. Moguće opcije su još 'bitString' i 'custom', ali u tom slučaju ne mogu biti zadovoljena linearna i nelinearna ograničenja.

- *PopInitRange: [2x1 double]*- matrica ili vektor koji određuje raspon jedinki u inicijalnoj populaciji, odnosno matrica dimenzija 2x1, dok je tipa podataka „double“

- *EliteCount: 2* - Pozitivan cijeli broj Jedinki koje iz trenutne generacije prelaze u iduću generaciju, bez vršenja ikakvih promjena na njima, tzv. *elitizam*. Inicijalna vrijednost je „2“.

- *Generations: 100* - Pozitivan cijeli broj koji određuje najveći mogući broj ponavljanja (iteracija) algoritma prije njegova prekidanja, inicijalna vrijednost je „100“.

- *MutationFcn: {[1x1 function_handle] [1] [1]} , tj. {[function_handle] [scale] [shrink]} odnosno* referenca funkcije koja vrši operator mutacije na djeci jedinki. Postoje tri vrste postojećih referenci funkcija mutacije:
  - *Gaussova*- („@mutationgaussian“ ili se piše još [1x1 function_handle]) koja je ujedno standardna funkcija mutacije. Mjesto mutacije je slučajan broj u rasponu kojom raspolaže jedinka, koje je generirano prema Gaussovoj distribuciji. Standardna devijacija distribucije je određena parametrima „Scale“ i „Shrink“.

- *„Scale“* parametar određuje standardnu devijaciju prve generacije, dok *„Shrink“* kontrolira kako se standardna devijacija smanjuje kako generacija prolazi. „Scale“ i „Shrink“ imaju inicijalnu vrijednost „1“ što ima daje linearni karakter. Postoje još dva gotova referenca za funkciju mutacije „@mutationuniform“,i „@mutationadaptfeasible“.

- *CrossoverFcn: @crossoverscattered* - Referenca funkcije koja se koristi za križanje dviju jedinki iz populacije (roditelja) i stvaranje novih jedinki nove populacije (djece). Standardna funkcija je: *Rasuta („@crossoverscattered“)* – funkcija stvara slučajno generiran binarni vektor, te selektira gene gdje vektor kod prvog roditelja ima vrijednost „1“, te kod drugog roditelja vrijednost „0“. Zatim kombinira gene kako bi stvorilo dijete. Na primjer, ukoliko imamo dva roditelja: „r1“ i „r2“:

  r1 = [a b c d e f g h]
  r2 = [1 2 3 4 5 6 7 8]

  te slučajni binarni vektor:

  bv = [1 1 0 0 1 0 0 0]

  dijete „d1“ će biti sljedeći novi vektor:

  d1 = [a b 3 4 e 6 7 8]

  Nadalje treba napomenuti kako postoji još gotovih funkcija za križanje unutar skupa alata „*Genetic Algorithm and Direct Search*“, npr. Funkcija heurističkog križanja „@crossoverheuristic“, funkcija uravnoteženog križanja„@crossoverintermediate“, funkcija sa jednom točkom križanja „@crossoversinglepoint“, funkcija sa dvije točke križanja „@crossovertwopoint“ i funkcija križanja uravnotežene aritmetičke sredine roditelja „@crossoverarithmetic“.

- CreationFcn: @gacreationuniform - Referenca funkcije koja generira početnu populaciju. Inicijalna funkcija je *uniformna* („@gacreationuniform“) ukoliko nemamo nikakvih ograničenja, ona generira slučajnu inicijlanu populaciju sa uniformnom distribucijom.Postoji još i *ostvarljiva* („@gacreationlinearfeasible“) funkcija, koja pri generiranju inicijalne populacije uzima u obzir ograničenja. Također je moguće stvoriti vlastitu funkciju, ali u tom slučaju moramo koristit njenu referencu, te postaviti svojstvo „PopulationType“ na 'custom'.

Kada smo definirali strukturu „options“, možemo pokrenuti funkciju „ga“ sa argumentima funkcije cilja i brojem nezavisnih ulaznih varijabli koji je u našem slučaju „1“.

```
[x fval] = ga(@funkcija_cilja, 1)
```

Rezultantne vrijednost funkcije „ga“ nakon prvog pokretanja iznosi:

```
x =

   0.0022

fval =

 -199.9998
```

Uviđamo kako vrijednost varijable „fval“ pri prvom pokretanju iznosi „-199.9998“ što je dosta blizu pravom rješenju minimuma, odnosno maksimuma funkcije cilja koja ima vrijednost „200“, tj. „-200“. Također, uviđamo kako je genetski algoritam uspio izbjeći sve lokalne minimume, odnosno maksimume. Nakon drugog pokretanja funkcije „ga“ dobivamo sljedeće vrijednosti:

```
x =

   0.0067

fval =

 -199.9985
```

Očekivano dobivamo rezultate različite od prvog pokretanja funkcije „ga“. No i dalje je vrijednost varijable „fval“ blizu lokalnog minimuma „-200“.

Ukoliko želimo poboljšati konačno rješenje moramo mijenjati svojstva strukture „options“. Promijenimo primjerice vrijednosti veličine populacije na vrijednost „1000“ i broj iteracija genetskog algoritma na „4000“ idućom linijom koda u MATLAB komandni prostor:

```
options = gaoptimset('PopulationSize', 1000, 'Generations', 4000)
```

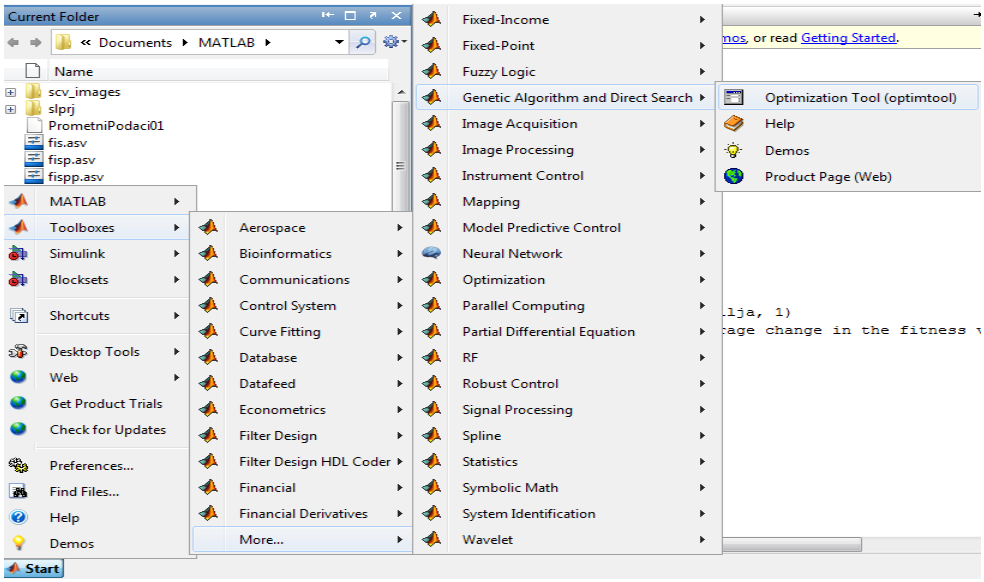
te ponovno pokrenimo funkciju „ga“, vrijednosti izlaznih varijabli će biti sljedeće:

```
x =

  6.9075e-004

fval =

 -200.0000
```

Vidimo da smo već pri prvom pokretanju funkcije „ga“ dobili pravu vrijednost globalnog maksimuma funkcije cilja. No to ne znači kako ćemo tu egzaktnu vrijednost dobivati prvi svakom pokretanju funkcije „ga“, već će se u rješenju češće pojavljivati egzaktna vrijednost nego u slučaju kada nismo mijenjali svojstva veličine populacije i broja iteracija. Time je poboljšan rad genetskog algoritma.

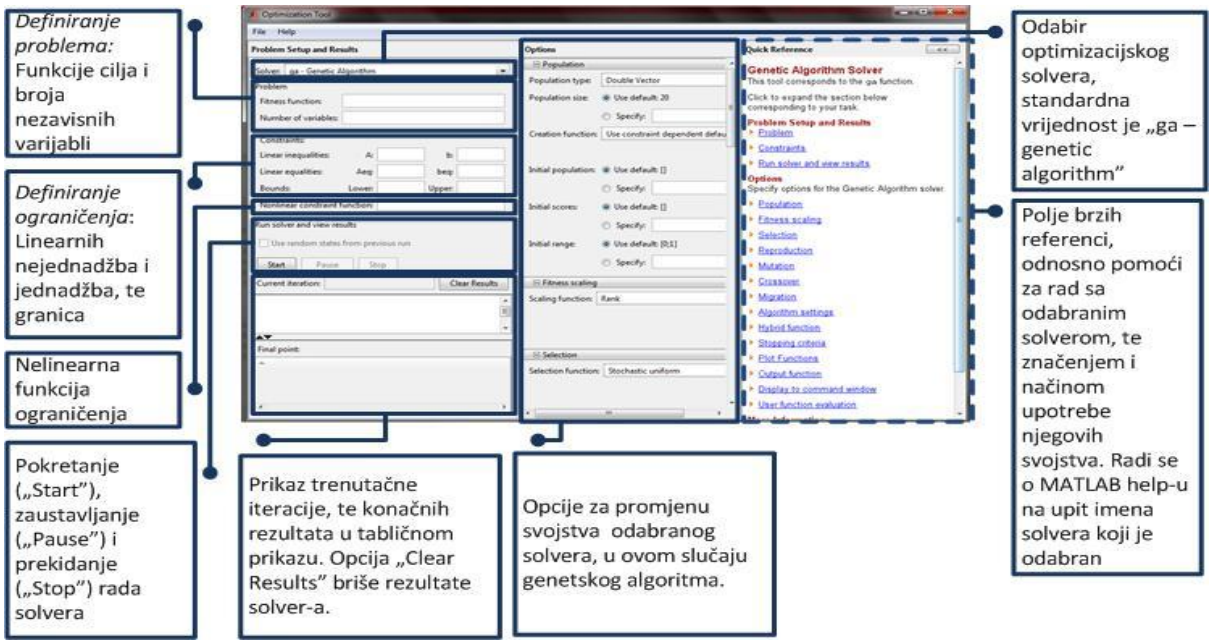
Grafičko korisničko sučelje (GUI) za rad sa genetskim algoritmom

Grafičko korisničko sučelje za rad sa genetskim algoritmima je ostvareno pomoću „Optimization Tool“-a. U tu svrhu potrebno je nakon pokretanja razvojne okoline MATLAB-a, odabrati „Start“ (u donjem lijevom kutu početnog MATLAB prozora) -> „Toolboxes“ -> „More...“ -> „Genetic Algorithm and Direct Search“ -> „Optimization Tool (optimtool)“. Postupak je grafički prikazan na slici 3.



Slika 3. Pokretanje „Optimization Tool“-a u MATLAB okruženju

Na slici 4. prikazan je prozor, te su objašnjenje temeljne funkcije za rad s genetskim algoritmom. Primijetimo kako GUI „Optimization Tool“ nudi u padajućem izborniku više solvera od kojih je standardno postavljen solver „ga - Genetic Algorithm“.



Slika 4. Prikaz i objašnjenje funkcija „Optimization Tool“-a

Pomoću „Optimization Tool“-a koristeći „Genetic Algorithm“ kao solver, minimizirajmo funkciju cilja:

$$f(x) = 0.4x_1 + 0.5x_2$$

uz ograničenja:

$$0.3x_1 + 0.1x_2 \leq 2.7$$
$$0.5x_1 + 0.5x_2 = 6$$
$$0.6x_1 + 0.4x_2 \geq 6$$

Ukoliko ovaj program riješimo pomoću linearnog programiranja *simplex* metodom dobivaju se rezultati: $x_1 = 7.5$ i $x_2 = 4.5$.

Ranije smo napomenuli kako genetski algoritmi, pa tako i sve funkcije iz skupa alata „*Genetic Algorithm and Direct Search*“ mogu jedino tražiti minimum funkcije. Pa tako i u slučaju ograničenja potrebno je prilagoditi nejednadžbe traženju minimuma funkcije. U tom slučaju zadnje ograničenje maksimuma prikazujemo na idući način:

$$(-0.6x_1) + (-0.4x_2) \leq -6$$

Potrebno je najprije napisati *m-funkciju* funkcije cilja imena „funkcija_cilja2“:

```
function y = funkcija_cilja2(x)
y = 0.4*x(1)+ 0.5*x(2);
```

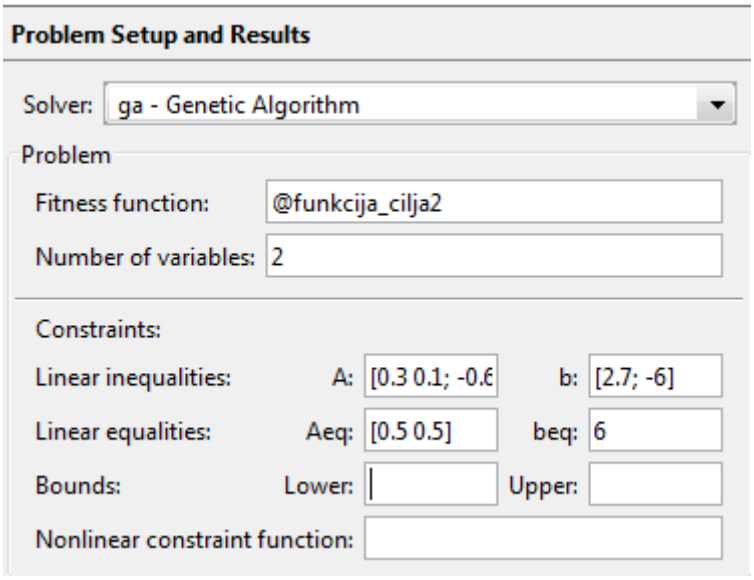
Zatim odrediti kako imamo dva nezavisna ulaza x_1 i x_2, te kodirati ograničenja na način:

A: [0.3 0.1; -0.6 -0.4]
b: [2.7; -6]
Aeq: [0.5 0.5]
Beq: 6

Napomenimo ovdje kako granice varijable primjerice:

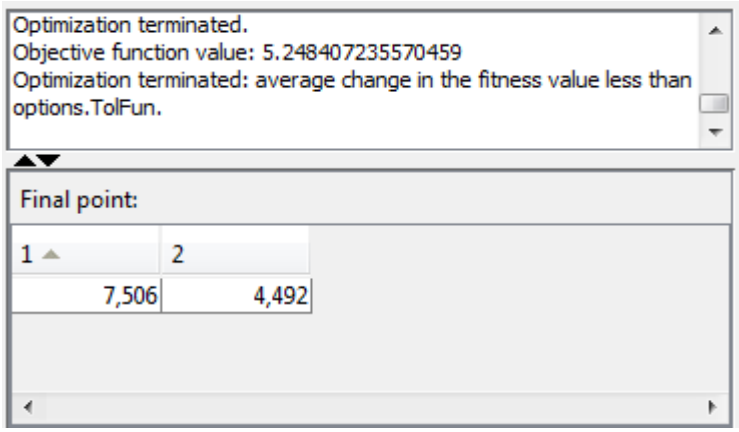
$$0.5 \geq x_1 \geq 0$$
$$1.5 \geq x_2 \geq 1$$

Unosimo pod polje „Bounds“ tako da razložimo granice na donje („Lower“) i gornje(„Upper“). A unosimo ih u obliku matrice n x 1 na sličan način kao i ograničenja, gdje je „n“ broj granica, u gornjem slučaju to je „2“ budući imamo dvije varijable. Kada smo sve to napravili unosimo podatke u „Optimization Tool“ obrasce kako je prikazano na slici 5.



Slika 5. Prikaz unosa podataka u „Optimization Tool“

Pritiskom na opciju „Start“ pokrećemo solver, te se dobivaju vrijednosti $x_1 = 7.506$ i $x_2 = 4.492$, dok je vrijednost funkcije cilja 5,248 kako je prikazano na slici 6.



Slika 6.6. Prikaz rezultata solvera „Genetic Algorithm“ u „Optimization Tool“-u

Zaključujemo kako su rezultati blizu stvarnih, ali ne točno baš oni, budući kako već znamo genetski algoritmi kao metaheuristička metoda daju optimalne rezultate prema definiranoj funkciji cilja. Kod ovakvih jednostavnih problema na kojima smo demonstrirali rad genetskog algoritma njihova je primjena upitna. Ukoliko rješavamo kompleksniji problem tradicionalnim metodama potrebno nam je puno više vremena, a i prijeti nam i mogućnost „zaglavljivanja“ u lokalnom minimumu funkcije cilja. Tada se koriste metaheurističke metode, među kojima je i genetski algoritam.

Rad na vježbi

Rad na vježbi sastoji se od rješavanja pronalaska maksimuma jedne funkcije cilja sa ograničenjima i donjim granicama koristeći solver genetskog algoritma unutar „Optimization Tool“-a. Idući zadatak se odnosi na rješavanje problema trgovačkog putnika. Prije vježbi potrebno je preuzeti potrebne gotove funkcije za crtanja grafa, funkcije cilja i funkcije generiranja inicijalne populacije. Zatim je potrebno izraditi matricu udaljenosti gradova, te pomoću postojećih funkcija genetičkih operatora i spomenutih preuzetih gotovih funkcija uz pomoć umjeravanja strukture „options“ i funkcije „ga“ riješiti zadani problem trgovačkog putnika.

Rješavanje problema pronalaženja maksimuma funkcije genetskim algoritmom

Koristeći „Optimization Tool“ posredstvom solvera „Genetic Algorithm“ pronađite maksimum funkcije:

$$\max f(x) = 2x_1 + 4x_2 + 3x_3$$

Uz ograničenja:

$$3x_1 + 4x_2 + 2x_3 \leq 60$$
$$2x_1 + x_2 + 2x_3 \leq 40$$
$$x_1 + 3x_2 + 2x_3 \leq 80$$

te donje granice („Bounds“ – „Lower“):

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_3 \geq 0$$

Pokrenite solver genetskog algoritma „12“ puta, te popunite tablicu 1.

Tab 1. Vrijednosti x1, x2 i x3

Varijabla x1	Varijabla x2	Varijabla x3

Analizirajte razlog promjena u izlaznim rezultatima u tablici 1. (povećavajte ili smanjite broj iteracija algoritma i populacije, te usporedite rezultate s kolegama). Koristeći MATLAB help pokušajte riješiti problem pronalaženja maksimuma - predstavljanjem nelinearnih ograničenja i granica koristeći funkciju „ga“ (bez „Optimization Tool“-a).

Rješavanje problema trgovačkog putnika genetskim algoritmom

Problem trgovačkog putnika (engl. „travelling salesman problem“) spada u grupu teških NP (engl. „non-polynomial“) problema, te je pogodan za primjenu genetskog algoritma. Problem koji je potrebno riješiti je da trgovac obiđe *n*-gradova, tako da ni u jedan grad ne dođe dva puta, te da se vrati u grad iz kojega je krenuo. Naravno, pri rješavanju tog problema trebamo pronaći optimalnu rutu kojom će trgovac proći minimalan put.

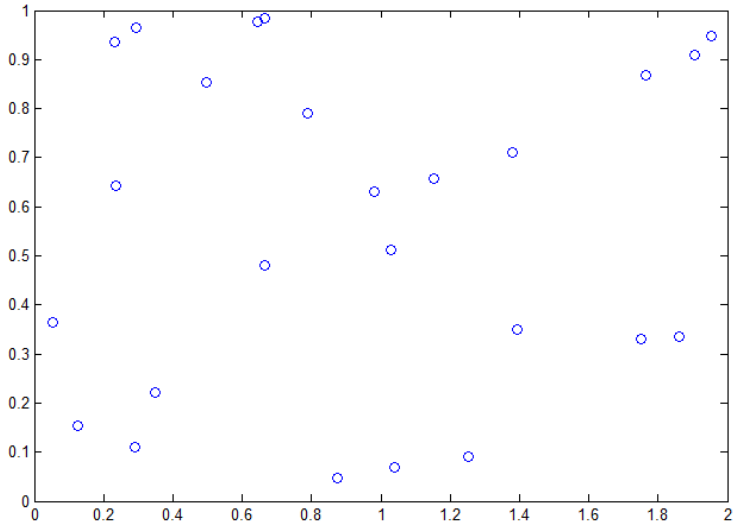
Problema trgovačkog putnika zbog svojih specifičnosti koristi posebne funkcije za genetičke operatore križanja i mutacije. Funkcije genetičkih operatora moraju reproducirati jedinke kod kojih ne dolazi do ponavljanja gradova budući trgovački putnik može doći u pojedini grad samo jednom. U MATLAB-u postoji gotova funkcija križanja za problem trgovačkog putnika, a njena referenca je: „@crossover_permutation“.

Genetički operator mutacije vrši zamjenu između dva slučajna grada. Unutar MATLAB-a koristimo funkciju mutacije za problem trgovačkog putnika sa referencom: „@mutate_permutation“.

Najprije kreirajte *m*-datoteku „vjezbe_genetski.m“ u kojoj ćete raditi sljedeće radnje.

Potrebno je napraviti varijablu „gradovi“ koja će sadržavat broj gradova, postavite tu varijablu na vrijednost „25“. Zatim stvorite matricu „lokacije“ koja će biti dimenzija [broj gradova x 2] koja će sadržavati koordinate gradova. Napunite matricu „lokacije“ slučajnim brojevima (koristit funkciju „rand“), tj. koordinatama gradova. S time da se lokacije „x“ elementa koordinata umnoži s vrijednošću „2“ kako bi koordinate dovoljno daleko bile udaljene jedne od druge.

Napravite ispis varijable „lokacije“ (označujući lokacije pomoću točaka - 'bo') kako je prikazano na slici 7.



Slika 7. Graf varijable „lokacije“

Zatim je potrebno izračunat udaljenosti između lokacija. Inicijalizirajte varijablu „udaljenost“ kao matricu sa vrijednostima „0“ dimenzija *broj gradova* x *broj gradova* (pri tome koristite funkciju „zeros“). U njoj će biti pohranjene vrijednosti udaljenosti između gradova. Zatim napravimo dvije ugniježđene „for“ petlje. Prva „for“ petlja se kreće kroz broj gradova, tj. puni varijablu „brojac1“ sa brojem grada kroz svaku iteraciju. Dok se druga „for“ petlja kreće do trenutačne vrijednosti „brojac1“ u toj iteraciji, tj. puni vrijednost varijable „brojac2“. Ovim ugniježđenim „for“ petljama iteriramo kroz indekse matrice „lokacije“ na način da se stvaraju parovi koordinata za izračun udaljenosti među njima. Princip je da se s trenutačnom lokacijom grada najprije izračunaju udaljenosti sa svim lokacijama prije nje u varijabli „lokacije“, a onda krene na drugu. Tako imamo uvijek dvije lokacije „x1“ i „y1“, te „x2“ i „y2“. Izračunamo udaljenost između te dvije lokacije po kodu jednadžbe za izračun udaljenosti dvije točke „sqrt((x1-x2)^2+(y1-y2)^2)“. Dobivene vrijednosti se spremaju u varijablu „udaljenost“ na mjesto „(brojac1,brojac2)“, a onda to učinimo i na mjesto „(brojac2,brojac1)“ kako bi matrica udaljenosti u varijabli „udaljenost“ bila simetrična. Na slici 6.8. vidimo primjer simetrične matrice udaljenosti.

	1	2	3	4	5	6	7	8
1	0	1.3755	0.8867	1.1309	0.2916	1.3158	0.5078	0.7645
2	1.3755	0	1.2989	0.4755	1.6618	0.0838	1.7146	1.5519
3	0.8867	1.2989	0	0.8383	1.0841	1.2899	0.7566	0.3388
4	1.1309	0.4755	0.8383	0	1.4209	0.4878	1.3587	1.1194
5	0.2916	1.6618	1.0841	1.4209	0	1.5998	0.4795	0.8810
6	1.3158	0.0838	1.2899	0.4878	1.5998	0	1.6711	1.5293
7	0.5078	1.7146	0.7566	1.3587	0.4795	1.6711	0	0.4652
8	0.7645	1.5519	0.3388	1.1194	0.8810	1.5293	0.4652	0

Slika 6.8. Primjer simetrične matrice udaljenosti

Zatim moramo definirat specifičan način izrade grafa. Budući ćemo kreirati dinamični graf za problem trgovačkog putnika koji će nam iscrtavati putanje između gradova ovisno o radu genetskog algoritma.

Koristimo gotovu funkciju „traveling_salesman_plot_vjezbe“ kojoj prosljeđujemo vrijednosti „options“, „state“,“flag“ i “lokacije“. Prvi tri vrijednosti su sistemske. Varijabla „state“ označava trenutačno stanje genetskog algoritma, dok se varijabla „flag“ i

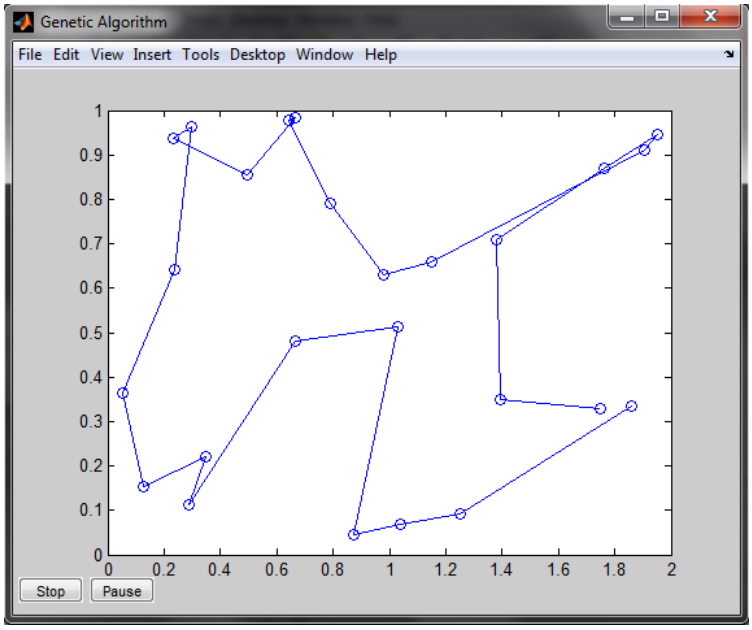
„options“ koriste kasnije kao povratna veza genetskom algoritmu. Idući korak je kreiranje „my_plot_vjezbe“ varijable koja će imati ulaznu referencu „@(options,state,flag)“ i izgledati ovako:

```
my_plot_vjezbe =
@(options,state,flag)traveling_salesman_plot_vjezbe(options,state,flag,
lokacije);
```

Zatim je potrebno stvoriti funkciju cilja za problem trgovačkog putnika. Koristimo gotovu funkciju „traveling_salesman_fitness_vjezbe“, te stvorimo varijablu „FitnessFcn“, sa ulaznom referencom „@(x)“, te proslijeđenim vrijednostima funkciji „traveling_salesman_fitness_vjezbe“ „x“ i „udaljenost“.

U ovom koraku potrebno je stvoriti strukturu „options“ za genetski algoritam. Populacijski tip je „custom“, raspon jedinki u inicijalnoj populaciji se kreće od vrijednosti „1“ do broja gradova (matrica raspona (2x1) se sastoji od vrijednosti „1“ i broj gradova upisanih jedna ispod druge). Referenca funkcije stvaranja inicijalne populacije je „@create_permutations“, dok su reference za gotovih funkcija mutacije i križanja za problem trgovačkog putnika spomenute ranije pri opisivanju problema. Broj iteracija postavite na vrijednost „100“, veličinu populacije na „30“, dok će broj nezavisnih ulaznih varijabli biti broj gradova.

Nakon završetka stvaranja strukture „options“ pokrenite genetski algoritam funkcijom „ga“. Rezultat bi trebao izgledati kako je prikazan na slici 9.



Slika 9. Problem trgovačkog putnika riješen genetskim algoritmom