Predmet: "Vještačka inteligencija"
Laboratorijska vježba 9: Genetički algoritmi
Odgovorna nastavnica: Vanr. prof. dr Amila Akagić

Sadržaj vježbe:

1 Cilj vježbe

Na ovoj laboratorijskoj vježbi studenti se upoznaju sa osnovama genetičkih algoritama, kao jednim od najpoznatijih i najčešće korištenih metaheurističkih algoritama u praksi. Kroz zadatke, studenti rješavaju primjere optimizacije multimodalnih kontinualnih funkcija, ali i neke složenije primjere iz oblasti diskretne optimizacije koristeći Python biblioteku **pyeasyga**[1].

2 Teoretski dio

2.1 Uvod u genetičke algoritme

Genetički algoritmi (eng. *Gentic Algorithms*, *GA*) su skupina algoritama koji spadaju u metaheurističke evolucione algoritme i koji se mogu koristiti za kontinulanu i diskretnu optimizaciju, kako sa ograničenjima, tako bez ograničenja. GA predstavljaju algoritme pretraživanja koji se zasnivaju na mehanizmu *prirodne selekcije*. Preciznije, oslanjaju se na jednu od najvažnijih Darvinovih teza: *opstanak najjačih* (eng. *surrvival of the fittest*). Opća ideja ove skupine algoritama je da se početna populacija kroz niz genetičkih transformacija približi globalnom optimumu. Nakon nekoliko iteracija (koje se u kontekstu GA nazivaju *generacije*), kada populacija više ne prolazi kroz promjene ili kada se dostigne maksimalan broj iteracija, najbolja jedinka iz populacije se uzima kao optimalno rješenje. Važno je napomenuti da GA, kao skupina metaheurističkih algoritama, *pokušavaju* pronaći *globalni optimum*, tj. pokušavaju *izbjeći zapadanje u lokalne optimume*, ali da nemaju *nikakvu garanciju* da je pronađeni optimum ujedno i globalni iz jednostavnog razloga što je problem globalne optimizacije jedan od najtežih problema trenutno prisutnih u nauci i inženjeringu.

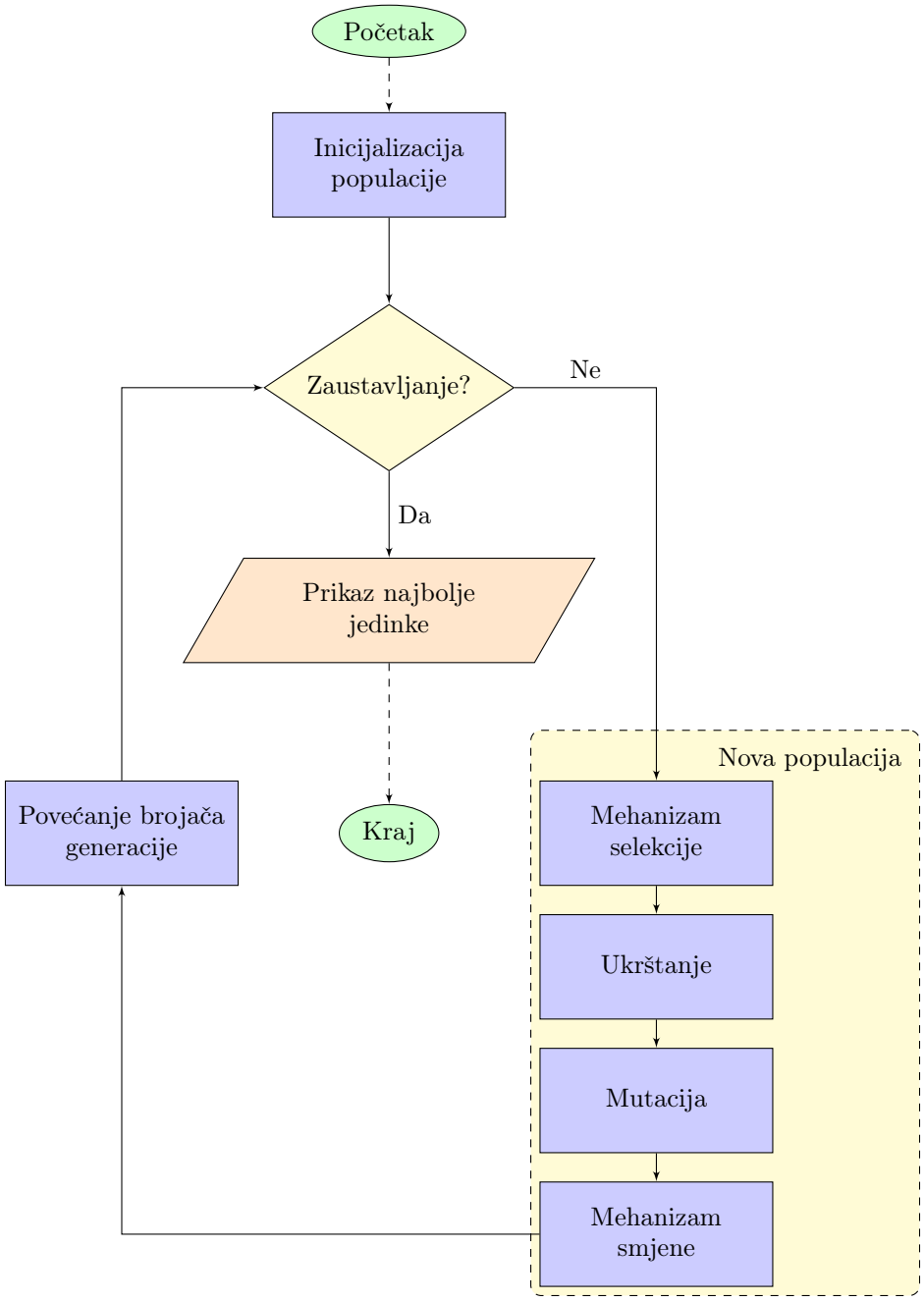
Genetički algoritmi su razvijani od strane John Hollanda zajedno sa svojim kolegama i studentima na Univerzitetu Michigen tokom 60-ih i 70-ih godina 20. vijeka. Prema definiciji jednog od svojih tvoraca, Davida Goldberga, genetički algoritmi predstavljaju "*algoritme pretraživanja koji su bazirani na mehanizmu prirodne selekcije i prirodne*

[1] https://pypi.org/project/pyeasyga/

genetike". Iz prethodne definicije možemo zaključiti da je opća ideja GA inspirisana genetičkim pojavama i procesima uočenim u prirodi. U nastavku su kratko opisani koraci GA.

U prvoj generaciji GA se vrši odabir *mehanizma kodiranja jedinki* i nasumična *inicilalizacija populacije* i kao rezultat se dobiva početna populacija. Nakon toga se, u ovisnosti od odabranog *mehanizma selekcije*, računa vrijednost *fitness funkcije* (eng. *fitness function*) za svaku jedinku unutar populacije, te se na osnovu tih vrijednosti iz trenutne populacije biraju jedinke koje će imati ulogu roditelja. Putem procesa *ukrštanja i mutacije* se od prethodno odabranih roditelja generiše nova populacija koja *mehanizmom smjene* mijenja trenutnu populaciju. Prethodno opisani postupak se ponavlja dok ne bude ispunjen kriterij zasutavljanja. Kao kriterij zaustavljanja se obično uzima ograničenje na maksimalan broj generacija ili uslov da se najbolja vrijednost populacije u odnosu na određen broj susjednih iteracija razlikuje za manje od neke male vrijednosti, tj. kada populacija duže vremena ne uspijeva pronaći bolje rješenje. Kada je zadovoljen kriterij zaustavljanja, najbolja jedinka iz populacije se uzima kao rješenje.

Prethodni opis genetičkih algoritama se može prikazati dijagramom toka prikazanim na slici 1



Slika 1: Dijagram toka genetičkih algoritama.

2.2 Mehanizam kodiranja

Prilikom korištenja GA, javlja se potreba kodiranja tačke n-dimenzionalnog problemskog prostora u neki drugi oblik koji je pogodan za realizaciju operatora genetičkih algoritama. Odnosno, potrebno je vršiti mapiranje iz prostora *fenotipa* u prostor *genotipa*. Dobiveni kodirani oblik tačke problemskog prostora se naziva *hromosom*, dok se funkcija koja vrši prethodno mapiranje naziva *mehanizam kodiranja*. Postoje mnogi načini mehanizma kodiranja od kojih se izdvajaju GA sa *binarnim kodiranjem* i GA sa *realnim kodiranjem*. S obzirom da je binarno kodiranje najjednostavniji mehanizam kodiranja i da je često zastupljen u primjenama GA, nastavak vježbe se fokusira na taj način kodiranja. U slučaju binarnog kodiranja jedinki, jedan od mogućih načina kodiranja je da se svaka koordinata jedinke (koja odgovara jednoj dimenziji problemskog prostora) zasebno kodira kao binarni string, te da se hromosom koji predstavlja jedinku dobije jednostavnim spajanjem dobivenih binarnih stringova. Naravno, što je veća dužina binarnog stringa, to je diskretizacija problemskog prostora bolja jer se omogućava pretraživanje većeg broja tačaka.

2.3 Inicijalizacija populacije

Inicijalizacija populacije predstavlja način određivanja početne populacije, tj. prve generacije. Obično se ovaj korak izvodi tako što se nasumično inicijaliziraju jedinke unutar dopustive oblasti, te se za tu svrhu mogu koristiti različite funkcije gustoće vjerovatnoće: uniformna raspodjela, normalna/Gaussova raspodjela, Cauchyjeva raspodjela i sl. Prethodno se obično realizira tako što se za svaku dimenziju problemskog prostora generiše onoliko slučajnih brojeva u intervalu [0,1] kolika je dužina binarnog stringa koji odgovara toj dimenziji. Nakon toga se zaokruživanjem generisanih brojeva dobivaju elementi binarnog stringa.

2.4 Mehanizam selekcije

S obzirom da se mehanizam selekcije zasniva na poređenju vrijednosti fitnessa jedinki, potrebno je prije opisa mehanizma selekcija, opisati šta predstavlja fitness funkcija. Funkcija kriterija/kriterijalna funkcija predstavlja funkciju cilja, tj. onu funkciju koju optimiziramo. Fitness funkcija predstavlja pogodno transformisanu kriterijalnu funkciju koja predstavlja mjeru kvalitete jedinke, te se koristi za poređenje jedinki u mehanizmu selekcije. Što je vrijednost fitness funkcije za neku jedinku veća, to je ta jedinka bliža globalnom optimumu. Obično se kao vid transformacije kriterijalne funkcije u fitness funkciju koristi neka linearna transformacija.

Mehanizam selekcije predstavlja jedan od najvažnijih koraka u GA s obzirom da se u ovom koraku na osnovu fitness-a jedinki vrši odabir koje jedinke će činiti osnov za izgradnju naredne populacije, tj. koje jedinke će preći u narednu fazu ukrštanja i mutacije. U slučaju da mehanizam selekcije previše favorizuje bolje jedinke, moguća je pojava preuranjene konvergencije ka lokalnom optimumu. S druge strane, ako mehanizam selekcije skoro nikako ne uzima u obzir bolje jedinke, moguća je pojava nasumične pretrage. Stoga se pokušavalo dizajnirati mehanizam selekcije koji pronalazi balans između prethodne dvije krajnosti. U nastavku ćemo opisati neke od najčešće korištenih mehanizama selekcije.

2.4.1 Rulet-točak selekcija

Izvorni mehanizam selekcije koji je korišten u radu John Hollanda je *rulet-točak selekcija*[2] Ovaj tip selekcije za svaku jedinku S_i računa vjerovatnoću p_i da će se odabrati kao roditelj koristeći izraz 1. Uočava se da se vjerovatnoća odabira dobiva prosto kao omjer fitnessa te jedinke $F(S_i)$ i sume fitnessa svih jedinki u populaciji.

$$p_i = \frac{F(S_i)}{\sum_{i=1}^{m} F(S_i)} \tag{1}$$

Nakon toga se za svaku jedinku definiše polje koje ima širinu jednako p_i koja se poredaju jedno do drugog na osi u rasponu $[0, 1]$, te se nasumično generiše broj unutar tog opsega, i bira ona jedinka čije polje odgovara nasumično generisanoj vrijednosti. Iz prethodnog opisa je jasno odakle potiče naziv "rulet-točak" selekcija jer se jedinke mogu posmatrati kao polja rulet-točka.

Kao mane rulet točak selekcije se može izdvojiti to što je eksperimentalno pokazano da često rezultira prebrzoj konvergenciji ka lokalnom optimumu, tj. da često previše naglašava bolje jedinke u odnosu na lošije, mogućnost da se ista jedinka odabere veliki broj puta, te da direktno ne funkcioniše sa negativnim vrijednostima fitnessa.

[2]Ovaj tip selekcije se može pronaći i pod nazivom *selekcija proporcionalna fitnessu*.

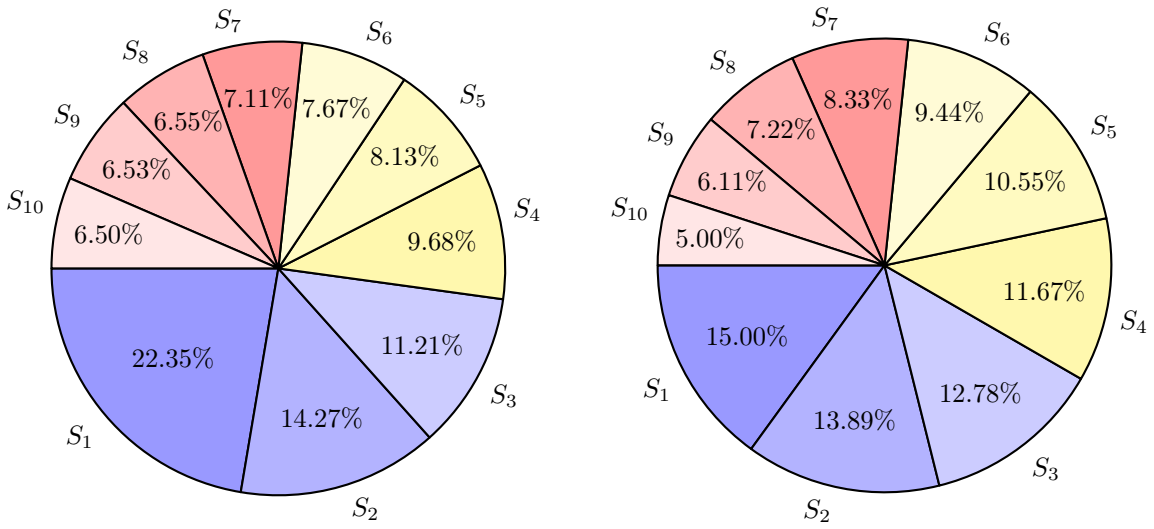
2.4.2 Selekcija na bazi ranga

Da bi se izbjegla mana rulet-točak selekcije da previše naglašava vjerovatnoću odabira hromosoma sa boljim fitness-om za reprodukciju, čime se umanjuje raznolikost populacije, koriste se drugi mehanizmi selekcije kao što je *selekcija bazirana na rangu hromozoma* (eng.*ranking selection*). Selekcija na bazi ranga se zasniva na modificiranju originalnog fitness-a u njegovu modificiranu vrijednost, na način da se ujednači razlika u vrijednosti fitness-a između boljih i lošijih jedinki populacije. Postoji više načina na koji se može provesti selekcija na bazi ranga i oni se uglavnom razlikuju po tome kako se izračunava modificirana vrijednost fitness-a. Ukoliko se modificirani fitness određuje na bazi *linearnog izraza*, tada će i vjerovatnoće selekcije opadati linearno od jedinke sa najboljim fitness-om do jedinke sa najlošijim fitness-om. Modificirani fitness jedinke S_i pri linearnoj selekciji na bazi ranga se obično određuje na osnovu izraza 2:

$$F_{mod}(S_i) = (2 - SP) + 2 \cdot (SP - 1) \cdot \frac{rank(S_i) - 1}{m - 1} \tag{2}$$

gdje je $rank(S_i)$ redni broj jedinke u populaciji poredane po neopadajućim vrijednostima fitnessa, m predstavlja broj jedinki unutar populacije, SP je parametar koji se naziva *selekcijski pritisak* (eng. *selection pressure*) i uzima se iz opsega $(1, 2]$. Mehanizam selekcije se dalje provodi na isti način kao i u rulet-točak selekciji, uz razliku što se umjesto izvornog fitnessa koristi modificirana vrijednost fitnessa. Primijetimo da u slučaju da parametar SP ima vrijednost blisku 1, tada bi i F_{mod} imala vrijednost blisku 1 za sve jedinke, drugim riječima tada bi GA vršio *potpuno slučajno pretraživanje* dopustive oblasti. U slučaju da je $SP = 2$ tada je razlika između vjerovatnoće odabira boljih i lošijih jedinki najveća, tj. tada se najviše favoriziraju bolje jedinke. Iz prethodnog je jasno zašto se parametar SP naziva selekcijski pritisak - predstavlja *značaj fitnessa* prilikom odabira jednike.

Na slici 2 može se vidjeti razlika vjerovatnoće odabira jedinki koristeći rulet-točak selekciju i selekciju na bazi ranga.



(a) Rulet-točak selekcija.

(b) Selekcija na bazi ranga, $SP = 1.5$.

Slika 2: Pie-chart vjerovatnoće odabira jedinke za slučaj populacije od 10 jedinki.

2.4.3 Turnirska selekcija

Turnirska selekcija za svaku jedinku nasumično bira određen broj jedinki iz trenutne populacije koje će sudjelovati u "turniru", tj. nadmetanju. Najbolja jedinka se proglašava kao "pobjednik" i prelazi u narednu fazu. Prednost ovog mehanizma selekcije je što, za razliku od rulet-točak selekcije, u općem slučaju *ne zahtijeva modifikaciju kriterijalne funkcije u fitness funkciju.* Naime, ako je u pitanju minimizacija, onda se uzima ona jedinka sa najmanjom vrijednosti kriterijalne funkcije, dok ako je u pitanju maksimizacija uzima se ona jedinka sa najvećom vrijednosti kriterijalne funkcije. S obzirom da se u svakoj iteraciji ovog mehanizma selekcije formira "turnir" sa k jedinki, ovaj mehanizam selekcije se još zove i k-turnirska selekcija. Obično se za k uzima vrijednost $k = 2$.

2.5 Ukrštanje i mutacija

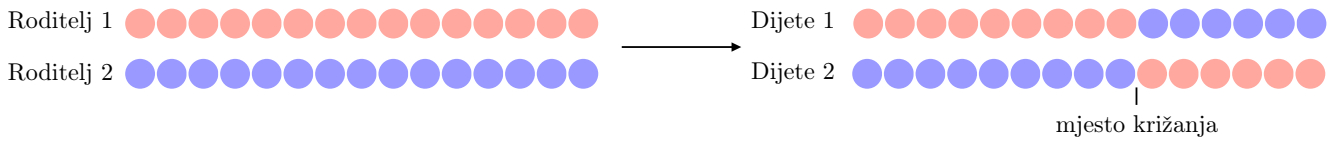
Prilikom mehanizma selekcije, izvršen je samo *odabir jedinki koje će učestvovati u ukrštanju i mutaciji.* Te jedinke se ni na koji način nisu mijenjale niti su generisale novo potomstvo. Taj korak je predviđen u ukrštanju/križanju (eng. *crossover*) i mutaciji (eng. *mutation*).

2.5.1 Ukrštanje

Ukrštanje predstavlja operator koji kao ulaz ima određen broj hromosoma koji predstavljaju roditelje, a kao rezultat daje isti broj hromosoma koji predstavljaju potomke (pri čemu se obično uzimaju 2 hromosoma roditelja). Prilikom ukrštanja, roditelji se nasumično biraju iz tzv. *bazena za reprodukciju* (eng. *mating pool*) koji predstavlja trenutnu vrijednost populacije nakon mehanizma selekcije. Nakon toga se vrši razmjena bita između roditelja i time se generišu novi potomci. Ideja ukrštanja je da će "dobri" roditelji generisati "bolju" djecu koja će se nalaziti bliže globalnom optimumu u odnosu na jedinke iz prethodne populacije. Prethodna ideja ukrštanja "dobrih" roditelja predstavlja osnovu hipoteze gradivnih blokova (eng. *building block hypothesis*) koja se smatra osnovnim principom funkcionisanja GA. Interesantno je da od svih evolucionih algoritama, operator ukrštanja se izvorno javlja samo u GA i neki autori smatraju da je operator ukrštanja primarni operator genetičkih algoritama. Kao parametar u GA se pojavljuje i vjerovatnoća ukrštanja p_c, što znači da je moguće da neki roditelji ne generišu potomke i time neizmjenjeni pređu u narednu generaciju. Obično se uzima da p_c ima vrijednost blisku 1.

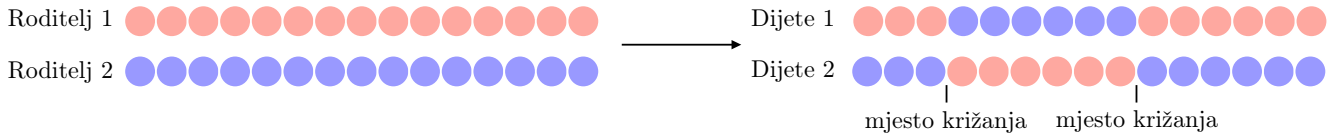
U nastavku su opisana tri najpoznatija tipa ukrštanja u GA: ukrštanje u jednoj tački (eng. *single point crossover*), ukrštanje u dvije tačke (eng. *two point crossover*) i uniformno ukrštanje (eng. *uniform crossover*).

Ukrštanje u jednoj tački je najjednostavnije ukrštanje u kojem se za svaki par roditelja nasumično bira jedna tačka ukrštanja. Ta tačka predstavlja granicu do koje dijete zadržava bite jednog roditelja nakon kojeg slijede biti drugog roditelja. Kao negativna strana ovog tipa ukrštanja se izdvaja činjenica da se krajevi hromosoma roditelja uvijek odvajaju, što može kao posljedicu imati da potomci mnogo odstupaju od roditelja. Na slici 3 je prikazano ukrštanje u jednoj tački.



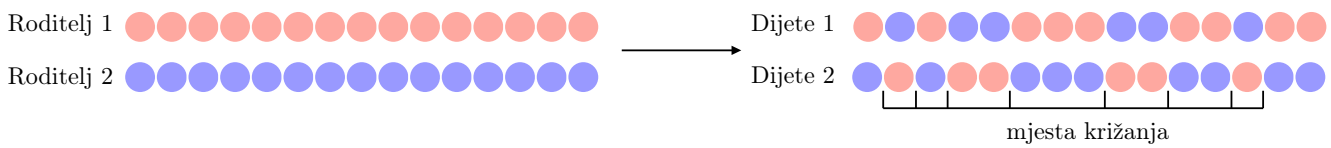
Slika 3: Ukrštanje u jednoj tački.

Zbog prethodno navedene mane ukrštanja u jednoj tački, uvedeno je ukrštanje u dvije tačke koje krajeve hromosoma ne odvaja. Kao što i sam naziv sugeriše, u ovom slučaju postoje dvije umjesto jedne tačke ukrštanja. Do prve tačke križanja se zadržavaju geni jednog roditelja, između dvije tačke križanja se zadržavaju geni drugog roditelja, dok ostatak čine geni onog roditelja čiji su geni zadržani do prve tačke križanja. Na slici 4 je prikazano ukrštanje u dvije tačke.



Slika 4: Ukrštanje u dvije tačke.

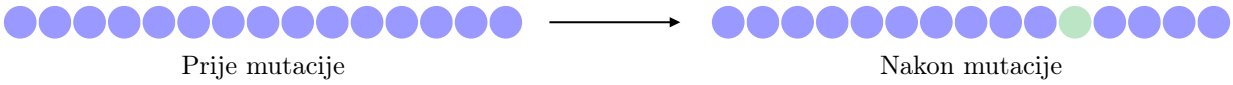
Dalje povećavanje broja mjesta ukrštanja dovodi do ukrštanja u s tačaka, odnosno kada je s jednako dužini hromosoma, tada se dolazi do uniformnog ukrštanja. U ovom tipu ukrštanja svaki bit hromosoma potomka ima jednaku vjerovatnoću da potiče od odgovarajućeg bita hromosoma bilo kojeg roditelja. Na slici 5 je prikazan primjer uniformnog ukrštanja.



Slika 5: Uniformno ukrštanje.

2.5.2 Mutacija

Mutacija je najjednostavniji korak u GA koji slučajno, sa malom vjerovatnoćom p_m, modificira hromosom na koji je primijenjen. Matematički gledano, mutacija predstavlja unarni operator jer se primjenjuje nad jednim hromosomom. Postoji više vrsta mutacije kao što su mutacija pomakom, inverzija dijela hromosoma i sl. U slučaju GA sa binarnim kodiranjem, obično se koristi mutacija koja negira vrijednost bita na nasumičnim lokacijama. Za potrebe mutacije se obično uzima mala vjerovatnoća u opsegu $0.001 \leq p_m \leq 0.05$, a njena vrijednost je obično uvjetovana veličinom populacije na način da "manje" populacije imaju veću vjerovatnoću mutacije u odnosu na "veće" populacije. Na početku razvoja GA smatralo se da operator mutacije, iako je neophodan, nema veliki značaj kao što je to slučaj sa ukrštanjem, mada mu se u posljednje vrijeme pridaje sve više značaja. Osnovna funkcija mutacije je da osigura raznolikost populacije GA i time omogući veću pretragu dopustivog prostora. Na slici 6 je prikazan primjer mutacije.



Slika 6: Primjer mutacije inverzijom dijela hromosoma.

2.6 Mehanizam smjene

Mehanizam smjene vrši zamjenu jedinki trenutne i nove populacije, što omogućava gradnju različitih verzija GA, od generacijskog GA (eng. *Generation GA*) do stabilnog GA (eng. *Steady-State GA*) kao krajnjih varijanti. Kod generacijskog GA se cjelokupna populacija roditelja mijenja populacijom potomaka. S druge strane, kod stabilnog GA se u populaciju uvodi samo mali broj novih jedinki dok se većina roditelja zadržava. GA koji koristi samo selekciju, ukrštanje i mutaciju neprestano otkriva i (zbog stohastičkih elemenata sadržanih u operatorima ukrštanja i mutacije) uništava hromosome sa visokom vrijednosti fitnessa. Zbog toga će populacije takvog GA neprekidno oscilirati u okolini tačke optimuma i vrlo je mala vjerovatnoća da će se u posljednjoj generaciji GA nalaziti jedinka čija vrijednost kriterijalne funkcije je za praktičnu primjenu dovoljno blizu optimumu. Iz prethodnog razloga se u GA pri rješavanju problema optimizacije uvodi *elitizam*. Pod elitizmom se podrazumijeva zadržavanje na svakoj generaciji jednog ili više hromosoma sa najboljim vrijednostima kriterijalne funkcije i njihovo *bezuslovno prenošenje u narednu generaciju*. Time se bitno povećava performansa GA jer se nerijetko dešava da se bez elitizma najbolja vrijednost kriterijalne funkcije ne mijenja i po nekoliko desetina generacija, što u nekim primjenama može biti nedopustivo sporo.

3 Primjer korištenja pyeasyga biblioteke

Kao primjer korištenja **pyeasyga** biblioteke bit će pokazano kako se ta biblioteka može koristiti za potrebe minimizacije *Drop-Wave* funkcije. Ova funkcija je izrazito kompleksna i multimodalna, te nalaženje globalnog minimuma, koji se nalazi u tački $(x_1,\, x_2) = (0,\, 0)$, ne predstavlja jednostavan zadatak. Zbog toga se često koristi kao testna funkcija za mnoge heurističke algoritme[3]. Drop-Wave funkcija je zadana izrazom 3:

$$f(x_1, x_2) = \frac{-1 - \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{0.5(x_1^2 + x_2^2) + 2} \tag{3}$$

U Google Colaboratory okruženju instalaciju **pyeasyga** biblioteke možemo učiniti koristeći sljedeće dvije naredbe:

```
import sys
!{sys.executable} -m pip install pyeasyga
```

Nakon instalacije potrebno je učitati `GeneticAlgorithm` klasu, te je za potrebe ovog primjera potrebno učitati i `numpy` i `random` biblioteke. Nakon učitavanja potrebnih biblioteka, definisana je sama kriterijalna funkcija, čiji globalni minimum se traži, te je instanciran objekat `ga` klase `GeneticAlgorithm`. Dopustiva oblast je definisana varijablom `RANGE` koja sadrži opseg dozovoljenih vrijednosti duž x i y ose. Uloge korištenih parametara konstruktora su jasne iz njihovog naziva. Tako npr. `seed_data` predstavlja podatke koji se koriste za definisanje optimizacionog problema, `maximize_fitness` određuje da li je potrebno maksimizirati ili minimizirati fitness i sl. Iako možda djeluje besmisleno minimizirati fitness, ovaj parametar omogućava da se koristi metoda selekcije koja ne zahtijeva transformaciju kriterijalne funkcije u fitness funkciju (kao što je npr. turnirska selekcija).

[3] Nekoliko drugih često korištenih testnih funckija se može pronaći na linku https://www.sfu.ca/~ssurjano/optimization.html

```
from pyeasyga.pyeasyga import GeneticAlgorithm
import random
import numpy as np

# dropwave function, xmin = [0,0] (coordinates of global minimum), fmin = -1 (value
    of global minimum)
def criteria_function(x):
    x1 = x[0]
    x2 = x[1]
    return (-1 - np.cos(12*np.sqrt(x1**2 + x2**2))) / (0.5*(x1**2 + x2**2) + 2)

# x is in range [-5, 5], y is in range [-5, 5]
RANGE = np.array([[-5,5], [-5,5]])

ga = GeneticAlgorithm(seed_data = criteria_function,
                      population_size = 30,
                      generations = 200,
                      crossover_probability = 0.8,
                      mutation_probability = 0.02,
                      elitism = True,
                      maximise_fitness = False)
```

S obzirom da se u primjeru koristi GA sa binarnim kodiranjem, definisana je i varijabla `BINARY_CHROMOSOME_LENGTH` koja predstavlja dužinu binarnog hromosoma. Sljedeći korak je definisanje metode koja će generisati jedinku. Ova metoda se koristi prilikom inicijalizacije populacije, te je realizovana tako što je nasumično generisan binarni string. S obzirom da se za ovu metodu kao parametar `data` interno unutar klase prosljeđuje parametar konstruktora `seed_data`, to je moguće iskoristiti te podatke za bolju inicijalizaciju populacije, ali to je za potrebe ovog primjera izostavljeno.

```
# first half => x, second half => y
BINARY_CHROMOSOME_LENGTH = 26

# randomly generate candidate solution, used in generating initial population
def create_individual(data):
    return [random.randint(0, 1) for _ in range(BINARY_CHROMOSOME_LENGTH)]

ga.create_individual = create_individual
```

Sljedeći korak je definisanje ukrštanja i mutacije. U ovom primjeru je kao ukrštanje korišteno ukrštanje u dvije tačke, te je mutacija realizovana negiranjem bita na nasumičnoj lokaciji.

```
# two-point crossover
def crossover(parent_1, parent_2):
    crossover_index1 = random.randrange(1, len(parent_1)-1)
    crossover_index2 = random.randrange(crossover_index1+1, len(parent_1))
    child_1 = parent_1[:crossover_index1] + parent_2[crossover_index1:
    crossover_index2] + parent_1[crossover_index2:]
    child_2 = parent_2[:crossover_index1] + parent_1[crossover_index1:
    crossover_index2] + parent_2[crossover_index2:]
    return child_1, child_2

ga.crossover_function = crossover

# randomly select a bit and invert it
def mutate(individual):
    mutate_index = random.randrange(len(individual))
    if individual[mutate_index] == 0:
        individual[mutate_index] = 1
    else:
```

```
        individual[mutate_index] = 0

ga.mutate_function = mutate
```

Kao mehanizam selekcije je odabran *default* mehanizam selekcije koji se koristi u sklopu **pyeasyga** biblioteke - turnirska selekcija. Kao atribut klase se može postaviti tip turnirske selekcije, te je postavljen na vrijednost 2 (*default* vrijednost ovog atributa je jedna desetina veličine populacije). Naravno, moguće je definisati i vlastiti tip selekcije koji kao parametar prima populaciju.

```
# define selection function here if necessary
# def selection(population):
# and assign it to the selection_function method
# ga.selection_function = selection

# using tournament selection by default

# set tournament size to 2
ga.tournament_size = 2
```

Nakon toga su definisane pomoćne funkcije:

- `decimal` - vrši konverziju liste binarnih vrijednosti u decimalnu vrijednost,
- `decode` - vrši dekodiranje jedinke, tj. binarni hromosom dekodira u realne vrijednosti koje odgovaraju koordinatama,
- `is_in_range` - provjerava da li je proslijeđena koordinata unutar odgovarajućeg opsega

Nakon toga je definisana fitness funkcija koja u ovom slučaju može biti jednaka kriterijalnoj funkciji jer je parametar `maximizeFitness` postavljen na `False`.

```
# list of 1s and 0s to decimal ([1,0,0,1] -> 9)
def decimal(binary):
    sum = 0
    for i in range(0, len(binary)):
      sum += binary[len(binary)-1-i]*2**(i)
    return sum

# decoding binary representation of individuals
def decode(individual):
    x_binary = individual[:BINARY_CHROMOSOME_LENGTH // 2]
    y_binary = individual[BINARY_CHROMOSOME_LENGTH // 2:]
    x_range = RANGE[0]
    y_range = RANGE[1]
    x_min = x_range[0]
    x_max = x_range[1]
    y_min = y_range[0]
    y_max = y_range[1]
    x_decode = x_min + ((x_max - x_min)*decimal(x_binary))/(2**(len(x_binary)) - 1)
    y_decode = y_min + ((y_max - y_min)*decimal(y_binary))/(2**(len(y_binary)) - 1)
    return x_decode, y_decode

def is_in_range(x, range_x):
  return x >= range_x[0] and x<= range_x[1]

def fitness (individual, data):
    x_decode, y_decode = decode(individual)
    while not (is_in_range(x_decode, RANGE[0]) and is_in_range(y_decode, RANGE[1])):
        individual = create_individual(data)
        x_decode, y_decode = decode(individual)
    fitness = data([x_decode, y_decode])
```

```
    return fitness

ga.fitness_function = fitness
```

Na kraju je potrebno samo pokrenuti genetički algoritam, te se ispisuje kodirana vrijednost najbolje jedinke i njena vrijednost fitnessa (prostor genotipa), ali i dekodirana vrijednost jedinke i vrijednost kriterijalne funkcije (prostor fenotipa). Pored toga, ispisane su i sve jedinke iz posljednje generacije.

```
ga.run()

print("Best individual fitness and best individual binary chromosome:")
print(ga.best_individual())

print("Best individual decoded:")
print(decode(ga.best_individual()[1]))

print("Best individual function value:")
print(criteria_function(decode(ga.best_individual()[1])))

print("\n---------------------------\n")

print("Last generation:")
for individual in ga.last_generation():
    print(individual)
```

Iz prethodnog primjera vidimo da GA imaju jako visok nivo apstrakcije, tj. moguće je po potrebi definisati sve neophodne operatore za neki optimizacioni problem. Upravo to je omogućilo da se GA mogu primijeniti na jako veliki broj klasa optimizacionih problema.

4 Zadaci za rad u laboratoriji

Zadatak 1 - Minimizacija Rastriginove funkcije

Modifikovati prethodni primjer tako da vrši minimizaciju Rastriginove funkcije unutar opsega $x_1, x_2 \in [-5.12, 5.12]$. Rastriginova funkcija je data izrazom 4:

$$f(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(\cos(2\pi x_1) + \cos(2\pi x_2)) \tag{4}$$

(a) Iscrtati funkciju unutar dopustive oblasti. Šta možete reći o obliku funkcije? Da li je funkcija multimodalna?

(b) Promijeniti dužinu hromosoma na 6. Da li je postignuta veća preciznost u pronađenom rješenju?

(c) Mijenjati parametar `mutation_probability` od vrijednosti bliskih 0 do vrijednosti bliskih 1. Uporediti rezultate.

(d) Mijenjati parametar `generations` i `population_size`. Da li je bolje za manje populacije koristiti više ili manje generacija?

(e) Postaviti `population_size` na vrijednost 5 i ne koristiti elitizam. Da li je postignuto bolje rješenje?

(f) Uporediti rezultate kada se kao mehanizam selekcije koristi rulet-točak selekcija i selekcija na bazi ranga.

Zadatak 2 - Problem 8 kraljica / 8 Queens problem

U dokumentaciji **pyeasyga** biblioteke se nalazi primjer rješavanja problema 8 kraljica (eng. *8 Queens Problem*)[4]. Problem se sastoji od pronalaska rasporeda 8 kraljica na šahovskoj ploči tako da nijedna kraljica ne napada preostale. Jedno moguće rješenje problema je prikazano na slici 7. Analizirati primjer i odgovoriti na pitanja:

(a) Kako je definisan mehanizam kodiranja?

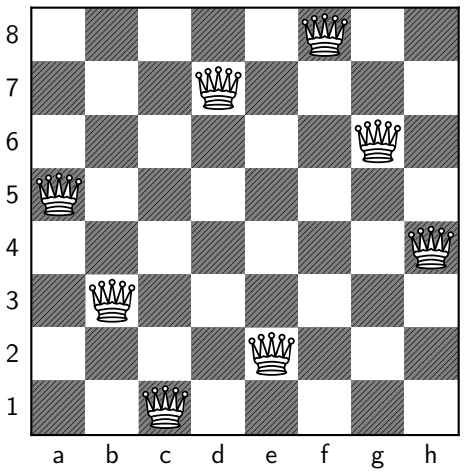
(b) Kako je definisan mehanizam selekcije?

(c) Kako je definisana fitness funkcija?

(d) Kako je definisan operator ukrštanja?

(e) Kako je definisan operator mutacije?

(f) Koje izmjene bi trebalo napraviti da se rješava generalizovani problem N kraljica?



Slika 7: Jedno moguće rješenje problema 8 kraljica.

[4] https://pyeasyga.readthedocs.io/en/latest/examples.html#queens-puzzle