Riješavanje problema putne torbe pomoću genetskih algoritama

Gal, Leon

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Rješavanje problema putne torbe pomoću genetskih algoritama

Završni rad

Leon Gal

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 01.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Leon Gal
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4343, 22.07.2019.
OIB Pristupnika:	64871659897
Mentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Riješavanje problema putne torbe pomoću genetskih algoritama
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	[Rezervirano: Leon Gal] Uvod u problem putne torbe, klasicno rješenje te analiza slozenosti. Vrste genetskih algoritama, primjena te usporedba performansi vise modela genetskih algoritama (single/multy crossover, razlicite fitness funkcije, itd...) za rjesavanje. Analiza slozenosti takvih metoda te njihova tocnost u pronalasku optimalnog rjesenja.
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	01.09.2022.
Datum potvrde ocjene od strane Odbora:	07.09.2022.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 08.09.2022.

Ime i prezime studenta:	Leon Gal
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4343, 22.07.2019.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Riješavanje problema putne torbe pomoću genetskih algoritama**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Problem putne torbe je popularni optimizacijski NP-teški problem kod kojeg se pokušava naći maksimalna vrijednost predmeta koji se mogu staviti u torbu, pod uvjetom da se ne premaši kapacitet torbe. Iako je problem proučavan više od tisuću godina, za velike količine predmeta vrlo je teško, gotovo nemoguće razviti algoritam koji bi bio u stanju pronaći optimalno rješenje. Dakako, problem ima stvarne primjene poput financijskog upravljanja te upravljanja proizvodnjom i resursima. U ovom radu se s ciljem pronalaženja što boljih rješenja problema putne torbe koriste genetski algoritmi. Genetski algoritmi su vrsta metaheurističkih, prirodom inspiriranih algoritama koji nizom iterativnih postupaka pretražuju danu domenu za optimalnim rješenjem. Algoritmi su najčešće relativno male vremenske složenosti s obzirom na onu čiji problem rješavaju te su kao takvi pogodni za vrlo efikasan pronalazak dovoljno dobrih, odnosno zadovoljavajućih rješenja. Iako su potencijalne primjene genetskih algoritama široke, danas se najčešće koriste u područjima robotike, strojnog učenja, neuronskih mreža te obradi slike.

U drugom poglavlju matematički je definiran problem putne torbe, ponuđen je primitivni algoritam za rješavanje te je odrađena analiza složenosti tog algoritma. U trećem poglavlju nalazi se uvod u genetske algoritme te je detaljno objašnjena terminologija nužna za razumijevanje rada algoritma. Predstavljen je svaki od operatora osnovnog genetskog algoritma, njegove varijacije te primjena. U zadnjem poglavlju su prikazana rješenja jedne varijante algoritma za zadane testne podatke te je odrađena analiza složenosti danog rješenja.

1.1. Zadatak završnog rada

Uvod u problem putne torbe, klasično rješenje te analiza složenosti. Vrste genetskih algoritama, primjena te usporedba performansi vise modela genetskih algoritama (single/multy crossover, različite fitness funkcije, itd...) za rješavanje. Analiza složenosti takvih metoda te njihova točnost u pronalasku optimalnog rješenja.

2. PROBLEM PUTNE TORBE

Problem putne torbe (engl. *knapsack problem*) je optimizacijski problem u kombinatorici koji glasi: „S obzirom na skup predmeta, svaki s pripadnom težinom i vrijednošću, odredite koje predmete treba uključiti u kolekciju, tako da je ukupna težina manja ili jednaka određenom ograničenju, a ukupna vrijednost što je moguće veća“ [1]. Kod ovog problema se pokušava naći maksimalna vrijednost predmeta koje možemo staviti u torbu, ali pod uvjetom da se ne premaši kapacitet putne torbe.

Neka je zadan prirodni broj N koji predstavlja broj predmeta na raspolaganju, koji definiramo i kao dimenziju problema [2]. Zatim, W je niz od N prirodnih brojeva, težina svakog predmeta te niz V od N prirodnih elemenata, vrijednost svakog predmeta. Broj C predstavlja kapacitet torbe, tj. maksimalnu težinu predmeta koja se može staviti u torbu. Potrebno je naći kombinaciju predmeta takvu da je suma svih vrijednosti predmeta u torbi maksimalna, a ipak da njihova težina ne prekorači kapacitet torbe.

Matematički zapis problema opisan je jednadžbom (2-1) gdje x_i predstavlja broj instance *i*-tog predmeta koji je sadržan u torbi, što znači da x_i iznosi 1 samo kada je *i*-ti predmet odabran, a u suprotnom iznosi 0.

$$maksimizirati \sum_{i=1}^{N} V_i * x_i \; tako\; da \sum_{i=1}^{N} W_i * x_i \leq C, x_i \in \{0,1\} \tag{2-1}$$

Programskih načina rješavanja ovog problema ima jako puno. Najjednostavniji za shvatiti je klasični algoritam koji je objašnjen u poglavlju 2.1, dok su mnogo bolji pristupi rješavanje dinamičkim programiranjem, evolucijskim algoritmima (diferencijalna evolucija, genetski algoritmi i sl.) ili pohlepnim algoritmima (engl. *greedy algorithms*).

U industriji i financijskom menadžmentu postoji mnogo stvarnih problema koji su izravno povezani s problemom putne torbe. Neki od njih su: nabavka zaliha, utovar tereta, raspoređivanje proizvodnje i proračun kapitala. S obzirom da je NP-težak, problem putne torbe je osnova za sustave enkripcije javnim ključem.

2.1. Klasični pristup rješavanja

Klasični algoritam za rješavanje ovog problema se oslanja na algoritam iscrpnog pretraživanja (engl. *bruteforce*) što znači da testira sve moguće kombinacije predmeta sve dok ne pronađe točno rješenje. Za realizaciju ovog rješenja koristimo programsku strukturu stog (engl. *stack*), koja predstavlja putnu torbu te se u nju redom stavljaju i vade predmeti različitih težina sve dok se ne

dobije tražena kombinacija. Kao reprezentacija predmeta stavlja se na stog indeks predmeta te se njime pristupa informacijama o težini i vrijednosti u poljima. Rezultati su indeksi koji se trenutno nalaze na stogu. Pseudokod ovakvog rješenja je prikazan na slici 2.1. gdje koristimo funkcije ***Push()***, ***Pop()***, ***Clear()***, ***IsEmpty()*** koje rade osnovne operacije sa stogom, funkcije ***StackCopy()*** koja kopira sadržaj stoga za potrebe prikaza sadržaja torbe, te funkcije ***GetSackValue()*** koja računa vrijednost svih predmeta u torbi.

Linija ***Kod***

```
1:      rjesenje = NULL
2:      indeks = -1, uk_tezina = 0, max_vrijednost = 0, n = 0;
3:      Clear()
4:      Ponavljati
5:          ako je uk_tezina < C i indeks < N onda
6:              indeks++
7:              uk_tezina += W[indeks]
8:              Push(indeks)
9:              temp = getSackValue()
10:             ako je temp > max_vrijednost i uk_tezina <= C onda
11:                 max_vrijednost = temp
12:                 rjesenje = StackCopyContent()
13:             Završi
14:         u suprotnom
15:             indeks = Pop()
16:             uk_tezina -= W[indeks]
17:             ako je indeks < N onda
18:                 indeks++
19:                 uk_tezina += W[indeks]
20:                 Push(index)
21:                 temp = getSackValue()
22:                 ako je temp > max_vrijednost i uk_tezina <= C onda
23:                     max_vrijednost = temp
24:                     rjesenje = StackCopyContent()
25:                 Završi
26:             završi
27:         Završi
28:     sve dok !IsEmpty()
```

Sl. 2.1. Pseudokod klasičnog algoritma putne torbe

2.2. Analiza složenosti klasične metode

Kao što je ranije rečeno algoritam isprobava sve moguće kombinacije predmeta unutar putne torbe tražeći najbolje rješenje. Ovaj algoritam će zasigurno naći najbolje moguće (optimalno) rješenje

budući da je testirao svaki mogući podskup predmeta. Kako se ovaj algoritam sastoji od kombinacija bez ponavljanja za svaki podskup predmeta od 1,2...,n elemenata, to se može zapisati kao $O(\binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n})$ što na kraju rezultira ukupnom vremenskom složenosti od $O(2^n)$. Dakle, vremenska složenost pristupa iscrpnog pretraživanja rješavanju problema putne torbe je eksponencijalna te zato ovaj problem spada u NP[1]-teške probleme. Kod eksponencijalne vremenske složenosti vrijeme potrebno da se algoritam izvrši se mijenja eksponencijalno s povećanjem ulaznih podataka. Upravo zbog toga ovaj algoritam nije dovoljan za rješavanje problema putne torbe s većim brojem predmeta. Na današnjem računalu, vidljivo iz tablice 2.1. već za samo 22 predmeta vrijeme izvršavanja se podiglo na skoro 4 sekunde. Vrlo brzo se dolazi do vremena izvršavanja koja nitko ne bi doživio, štoviše za samo 77 predmeta, bilo bi potrebno čekati čak 5 milijardi godina prije nego bi računalo izbacilo točno rješenje.

Tablica 2.1. Vrijeme izvršavanja pristupom iscrpnog pretraživanja

Broj predmeta	*Broj kombinacija*	*Vrijeme [s]*
5	32	0.000003
10	1,024	0.000807
20	1,048,576	1.000000
21	2,097,152	1.940000
22	4,194,304	3.910000

Prostorna složenost klasične metode rješavanja ovisi o broju predmeta. S obzirom da se testiraju sve kombinacije predmeta, granični slučaj je da je torba dovoljno velika kako bi stali svi predmeti. S obzirom da je torba programski prikazana stogom, on će rasti maksimalno do broja predmeta. Prema tome, iz tog slijedi da veličina stoga linearno ovisi o broju premeta pa samim time i zauzeće memorije. Iz ove kratke prostorne analize vidljivo je da se radi o $O(n)$ tj. linearnoj prostornoj složenosti.

[1] NP – nedeterministički polinomijalni

3. GENETSKI ALGORITMI

Evolucija je optimizacijski proces čija je svrha poboljšanje mogućnosti organizama ili sustava da preživi u dinamički promjenjivim i nepogodnim okruženjima. Osnovna ideja evolucije je da se pojedinac prilagodi tokom svog životnog ciklusa te prenese ta znanja na svoje potomke. Nakon čega potomci nastavljaju daljnju prilagodbu na način da gube one osobine koje im nisu potrebne i razvijaju one njima bitne.

Na gore opisanom procesu zasnivaju se svi evolucijski algoritmi (EA) (engl. *Evolutionary Algorithms*). Evoluciju kroz prirodnu selekciju nasumično odabranih pojedinaca iz populacije, možemo promatrati kao traženje najbolje od svih mogućih vrijednosti kromosoma. U tom slučaju, EA nisu ništa drugo nego nasumično traženje optimalnog rješenja za zadani problem. Na kvalitetu procesa pretrage utječe koliko dobro su definirane slijedeće komponente EA:

- Način kodiranja rješenja problema kao kromosom (binarni zapis, decimalni zapis)
- Definiranje funkcije koja vrednuje snagu za preživljavanje (engl. *fitness*) pojedinca
- Definiranje početne populacije
- Operatori selekcije i reprodukcije

Genetski algoritmi (GA) su podklasa EA. GA modeliraju genetsku evoluciju u računalu na način da binarnom reprezentacijom (bitovima) predstavljaju genetske stanice živih organizama te osobine pojedinaca izražavaju preko genotipova (engl. *Genotypes*). U kontekstu GA kada se govori o kromosomima misli se na pojedince određene populacije, koji se nadalje sastoje od gena, tj. njihove genetske reprezentacije. Osnovni operatori GA su:

- selekcija (engl. *selection*) – operator koji predstavlja model opstanka najsposobnijih (engl. *survival of the fittest*), objašnjen detaljnije u poglavlju 3.1.4
- rekombinacija (engl. *crossover*) – operator preko kojeg se modelira reprodukcija kromosoma, objašnjen detaljnije u poglavlju 3.1.5
- mutacija (engl. *Mutation*) – operator koji omogućava unošenje novog genetskog materijala, objašnjen detaljnije u poglavlju 3.1.6

Način na koji se ove komponente spajaju u smislenu cjelinu pokazuje opis na slici 3.1. gdje se osnovni koraci ponavljaju sve dok uvjeti zaustavljanja nisu postignuti (najčešće određen broj generacija ili dok se ne dođe do željene preciznosti rješenja).

Linija ***Kod***

```
1:      Postaviti brojač generacija na nula, gen=0
2:      Inicijalizirati populaciju C(n), gdje je n broj početne populacije
3:      Sve dok zaustavni uvjet(i) nisu zadovoljeni
4:      Ponavljati
5:          Vrednuj kvalitetu f(x_i(gen)), gdje je x_i pojedinac populacije
6:          Napraviti reprodukciju s ciljem dobivanja potomaka
7:          Odabrati novu populaciju, C(gen+1)
8:          Nastaviti na novu generaciju, t=t+1
9:      Završi
```

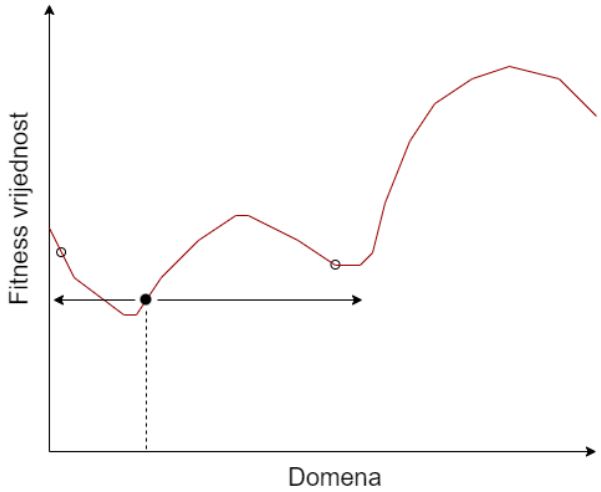
Sl. 3.1. Opis osnovnog genetskog algoritma

3.1. Osnovni pojmovi

U ovom poglavlju detaljnije su objašnjeni najbitniji dijelovi svakog genetskog algoritma, kao što su početna populacija, funkcija dobrote te operatori selekcije, rekombinacije i mutacije. Prvo je potrebno objasniti dva najosnovnija pojma koja omogućuju pretragu složene domene za optimalnim rješenjem:

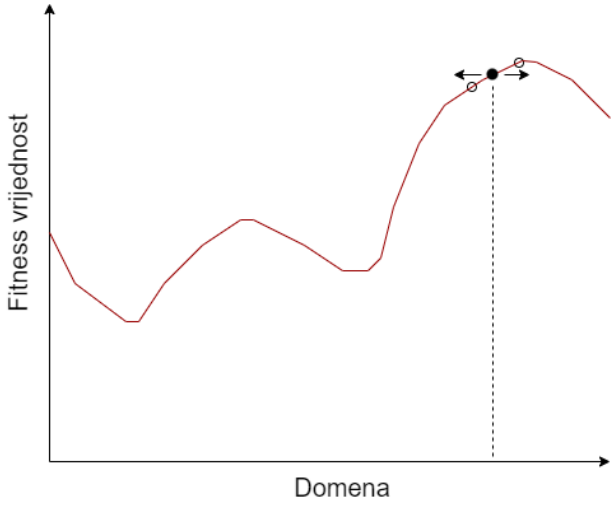
- istraživanje (engl. *exploration*) – koncept preko kojega se modelira potreba za nasumičnom pretragom domene s ciljem otkrivanja što većeg dijela domene.
- eksploatacija (engl. *explotation*) – naziv za način na koji se istražuje mala okolina poznatog dijela domene.

Kao što je vidljivo na slici 3.2. kod istraživanja elementi proširuju svoje potomstvo na udaljene dijelove domene za pretragu te tako omogućavaju uniformnije pretraživanje i samim time veće šanse za pronalazak globalnog maksimuma.



Sl. 3.2. Grafički prikaz istraživanja

Na slici 3.3. je vidljivo da kod eksploatacije, potomstvo neće biti jako udaljeno od roditelja, upravo to nam omogućava pronalazak lokalnog maksimuma.



Sl. 3.3. Grafički prikaz eksploatacije

Ukratko možemo reći da istraživanje raspršuje elemente po domeni i pomaže pronalasku globalnog maksimuma, dok eksploatacija sužava pretragu na malu pod-domenu oko roditelja te omogućava pronalazak lokalnog maksimuma. Istraživanje i eksploatacija uvijek dolaze u kombinaciji kako bi se postigli najbolji rezultati pretrage [3].

3.1.1. Prikaz kromosoma u računalu

Kromosom (genom) ili pojedinac su nazivi koji će se koristiti dalje u radu. To je fiksni set gena koji se smatra matematičkim objektom, nad kojim su definirani određeni operatori koji omogućavaju njegovo modificiranje. Dvije su verzije prikaza kromosoma u računalu, binarna reprezentacija te decimalna (engl. *floating-point*) reprezentacija. Za potrebe ovog algoritma bitno je objasniti samo binarnu reprezentaciju koja se koristi.

U binarnoj reprezentaciji, kromosomi se prikazuju kao niz binarnih znakova u memoriji računala. Prednosti ovakvog pristupa su jednostavnost izrade, mnoge informacije se lako kodiraju binarno, jednostavna implementacija genetski operatori [4]. Primjer ovako kodiranog pojedinca sa fiksnom duljinom od 6 gena prikazan je na slici 3.4. Svaki kromosom predstavlja jedno rješenje, dok bitovi unutar njega predstavljaju gene. Vrijednost svakog gena se naziva alel.

Sl. 3.4. Binarna reprezentacija kromosoma

3.1.2. Početna populacija

Kao što je ranije rečeno genetski algoritmi su zapravo stohastički, populacijski algoritmi pretraživanja, prema tome svaki GA mora sadržavati populaciju kandidata za rješenje. Prvi korak zato mora biti stvaranje početne populacije.

Najjednostavniji i najčešće korišten način generiranja početne populacije je dodjeljivanje nasumičnih vrijednosti iz dozvoljene domene svakom genu u svakom pojedincu. Cilj nasumičnog odabira je osigurati da je početna populacija uniformna reprezentacija cijele domene pretrage (engl. *search space*). Takvim načinom odabira odmah u startu postoje veće šanse biti bliže optimalnom rješenju jednim od kromosoma. U suprotnom slučaju, pokrijemo li samo jedan dio domene pretrage, šanse su da nikada nećemo istražiti određen dio domene te potencijalno propustiti optimalno rješenje [5].

Veličina početne populacije također igra veliku ulogu u točnosti i brzini ukupnog algoritma. Velika količina pojedinaca povećat će raspršenost što rezultira poboljšanje istraživačkih mogućnosti populacije. Naravno, što je više pojedinaca veća je računalna složenost po generaciji. Iako se vrijeme izvršavanja po generaciji povećalo zbog velike populacije, sada je potrebno manje iteracija da bi se postiglo prihvatljivo rješenje.

Uzme li se premala početnu populaciju ograničit će se istraživačke mogućnosti, odnosno populacija će sadržavati mali dio domene pretrage. U ovom slučaju iako je vrijeme po generaciji manje, algoritmu će biti potrebno više iteracija da konvergira, za razliku od velika populacije. U slučaju manjih populacija GA može se prisiliti na veću pretragu domene povećavanjem stope mutacije (poglavlje 3.1.6).

3.1.3. Funkcija dobrote

Općenito u prirodi vrijedi pravilo da će pojedinci najboljih obilježja imati najveće šanse za preživljavanje i daljnje širenje. Kako bi se odredila sposobnost pojedinca za preživljavanje, koristi se matematička funkcija koja računa koliko je dobro rješenje za određen kromosom te je nazivano funkcija dobrote (engl. *Fitness function*). Funkcija dobrote (jednadžba 3-1) kao i svaka druga matematička funkcija vrši preslikavanje, točnije, preslikavanje iz reprezentacije kromosoma u

$$f : \Gamma^{n_x} \rightarrow \mathbb{R} \tag{3-1}$$

skalarnu vrijednost.

Gdje Γ predstavlja vrstu podataka elemenata n_x-dimenzionalnog kromosoma. Vrijednosti koje vraća funkcija dobrote su realni brojevi koje je nekada potrebno normalizirati u raspon [0, 1] dijeljenjem s najvećom kvalitetom, najčešće za potrebe proporcionalne selekcije. Funkcija dobrote je dio GA koji vrednuje kvalitetu rješenja te je ona jedini dio algoritma koji je potrebno ponovno napisati ukoliko bismo htjeli rješavati neki drugi problem.

U slučaju putne torbe, ukoliko je kumulativna masa elemenata veća od mase torbe funkcija dobrote vraća vrijednost nula, dok u suprotnom vraća zbroj vrijednosti svih elemenata u torbi.

3.1.4. Operator selekcije

Selekcijski operator je jedan od glavnih operatora GA te on izravno povezuje koncept opstanka najsposobnijih (engl. *survival of the fittest*). Glavni zadatak selekcije je da pokušava dobiti što bolja rješenja, a to se postiže na dva načina:

Selekcija nove populacije – nova populacija s kandidatima za rješenje selektira se na kraju svake generacije, kako bi služila kao roditeljska populacija za sljedeću generaciju. Nova populacija selektira se isključivo iz potomaka ili iz potomaka i roditelja. Pravilno postavljen selekcijski operator bi trebao osigurati da kvalitetni pojedinci prežive u sljedeću generaciju.

Reprodukcija – Potomci se kreiraju isključivo kroz primjenu operatora rekombinacije te mutacije. U slučaju rekombinacije, bolji pojedinci trebaju imati veće šanse za razmnožavanje kako bi osigurali da potomak sadrži genetski materijal najboljeg pojedinca. U slučaju mutacije, selekcijski mehanizam se treba fokusirati na slabije pojedince, nadajući se da će mutacija slabog pojedinca dovesti do dobrih osobina te da će tako povećati šanse za opstanak.

Postoji mnogo operatora selekcije, no svima je ista zadaća opisana iznad. Osnovni operatori su:

Nasumična selekcija – Najjednostavniji oblik selekcijskog operatora, gdje svaki pojedinac ima jednaku šansu $\frac{1}{n_s}$ (gdje je n_s veličina populacije) biti izabran. Ne koristi se informacija o kvaliteti, što znači da najbolji i najlošiji pojedinac imaju jednake šanse preživjeti u iduću generaciju, ali zato je brži od ostalih selekcijskih operatora.

Proporcionalna selekcija – oblik selekcije koji favorizira najboljeg pojedinca. Stvara se distribucija vjerojatnosti proporcionalna kvaliteti pojedinca, te su šanse svakog pojedinca definirane jednadžbom (3-2).

$$\varphi_s(x_i(t)) = \frac{f_Y(x_i(t))}{\sum_{l=1}^{n_s} f_Y(x_l(t))} \tag{3-2}$$

Gdje je n_s ukupan broj pojedinaca populacije, $\varphi_s(x_i)$ je vjerojatnost da će x_i-ti pojedinac biti odabran, $f_Y(x_i)$ je skalirana funkcija dobrote x_i-tog pojedinca kako bi vraćala isključivo pozitivne brojeve.

Na slici 3.5. vidi se da su pojedinci odabrani na način da se odabire nasumičan broj r, te se šanse za izvlačenje sumiraju i onaj koji prvi svojim šansama prekorači broj r bit će odabran za razmnožavanje [5].

Linija ***Kod***

```
1:      i = 1
2:      Izračunati φs(xi) koristeći jednadžbu 3-2
3:      suma = φs(xi)
4:      Odabrati r ~ U(0,1)
5:      Sve dok suma < r ponavljati
6:          i = i + 1
7:          suma = suma + φs(xi)
8:      Završi
```

Sl. 3.5. Algoritam proporcionalne selekcije

Zbog toga što je proporcionalan odabir s kvalitetom pojedinca, može se dogoditi da snažni pojedinci dominiraju u proizvodnji potomstva dovodeći do prerane konvergencije, slabe raspršenosti po domeni ili čak sub-optimalnog rješenja.

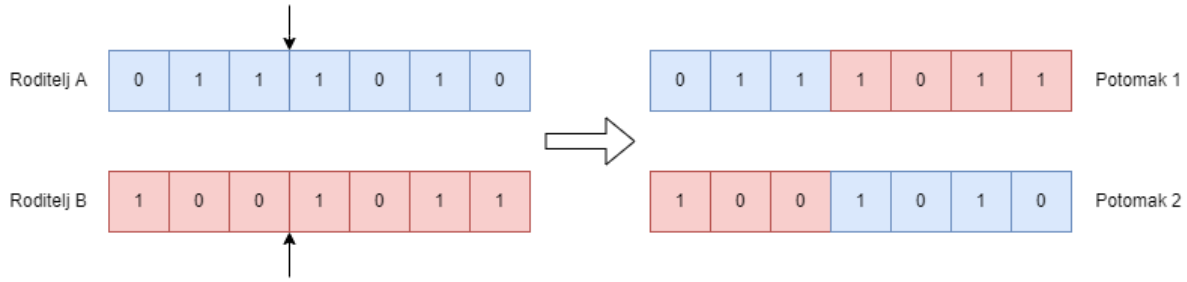
Selekcija natjecanjem (engl. *tournament selection*) – Oblik selekcije koji nasumično odabire grupu od n_{ts} pojedinaca, tako da je $n_{ts} < n_s$. Tada se odabrani pojedinci međusobno uspoređuju (natječu) te je najbolji pojedinac iz grupe odabran [6]. Za rekombinaciju s dva roditelja, selekcija natjecanjem mora se izvršiti dvaput, jednom za odabir svakog od roditelja. Ovakav pristup sprječava najsnažnijeg pojedinca da dominira zato što, iako koristi informaciju o kvaliteti za usporedbu pojedinaca, koristi i nasumičan odabir grupe pojedinaca što smanjuje učestalost pojavljivanja najboljeg. Drugim riječima

ako se odabere grupa od $n_{ts} = n_s$ tj. cijela populacija, tada će svaki put biti odabran najbolji pojedinac, dok u drugom slučaju, ako je $n_{ts} = 1$ tada se postiže nasumična selekcija.

3.1.5. Operator rekombinacije

Rekombinacija (engl. *crossover*) je proces stvaranja jednog ili više novih pojedinaca kroz kombinaciju genetskih materijala koji su nasumično odabrani od dva ili više roditelja. Razmatrajući samo *seksualne rekombinacije*[2], dvije su podvrste:

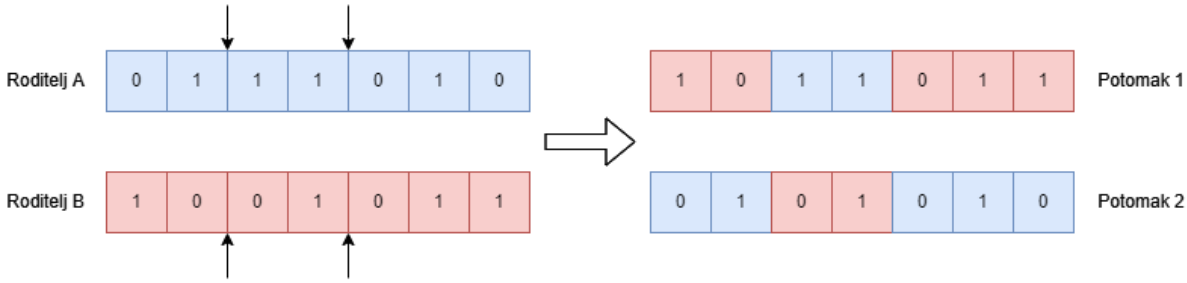
Rekombinacija u jednoj točki (engl. *Single point crossover*) – Potomak se stvara tako što odabiremo jednu nasumičnu točku (indeks) u kojoj će se ukrstiti genetski materijali



Sl. 3.6. Prikaz rekombinacije u jednoj točki

roditelja. Tada prekopiramo dio genetskih materijala do te točke od jednog roditelja, a nakon te točke od drugoga. Slika 3.6. prikazuje primjer rekombinacije u jednoj točki na mjestu strelica.

Rekombinacija u više točaka (engl. *Multy point crossover*) – Potomak se stvara tako što odabiremo minimalno dvije nasumične točke (indeksa) u kojima će se ukrstiti genetski

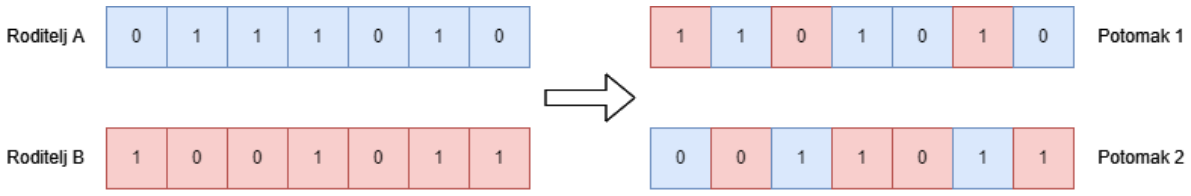


Sl. 3.7. Prikaz rekombinacije u dvije točke

[2] Klasa rekombinacije koja podrazumijeva dva roditelja, postoje i aseksualne gdje je samo jedan roditelj, te multi-rekombinacije koje koriste više roditelja za stvaranje potomstva.

materijali roditelja. U ovom slučaju izmjenično se kopiraju genetski materijali oba roditelja sve dok se ne popune sve regije određene točkama. Primjer dobivanja potomaka za rekombinaciju s dvije točke prikazan je na slici 3.7. gdje su točke prikazane strelicama.

Uniformna rekombinacija (engl. *Uniform crossover*) – U ovom postupku se ne koristi



Sl. 3.8. Prikaz uniformne rekombinacije

točka (indeks) za odabir područja, nego se potomci stvaraju tako što za svaki gen koji se umeće u potomka „bacamo novčić“ odnosno nasumično se odabire gen kojeg roditelja biramo te postupak ponavljamo za sve gene pojedinca, kako je prikazano na slici 3.8.

Ovim postupcima dobiju se dva potomka. Uvrste li se oba izravno u sljedeću generaciju tada je potrebno manje postupaka rekombinacija dok se ne napuni populacija novim naraštajem. Ovaj način dovodi do raspršivanja pojedinaca s obzirom na domenu pretrage zato što se u sljedeću generaciju uzima i bolji i lošiji potomak. Pametniji način je da se uzme uvijek samo bolji te se on postavi u sljedeću generaciju. Ovaj slučaj ipak ima mane ako je najbolji potomak svejedno lošiji od svojih roditelja, treba razmotriti trebali ga uopće uvrstiti u sljedeću generaciju.

Korištenjem rekombinacije u više točaka dodatno povećavamo raznolikost genetskog materijala pojedinaca, zato što je povećan broj mogućih kombinacija slaganja gena. Kod rekombinacije u jednoj točki potomci će biti sličniji roditeljima te se može dogoditi da se počnu pojavljivati novi pojedinci s identičnim genetskim materijalima kao roditelji.

3.1.6. Mutacijski operator

Generalna ideja mutacije leži iza toga da se u prirodi događaju konstantno mutacijske promjene svih jedinki. Iako one pojedinačno gledano možda nisu korisne te upropaste pojedinca, na velikoj skali takve mutacije pomažu prilagodbi i evoluciji. Cilj mutacije je predstavljanje novog genetskog materijala postojećim pojedincima, što za posljedicu ima raspršivanje genetskih karakteristika populacije po većem dijelu domene pretrage. Mutacija se zadaje kao određena vjerojatnost p_m promjene stanja bita svakom genu potomka. Ako znamo vjerojatnost mutacije svakog gena p_m, možemo izračunati vjerojatnost mutacije svakog pojedinca P_m, prikazano jednadžbom (3-3).

$$P_m\big(x_i(t)\big) = 1 - (1 - p_m)^{n_x} \tag{3-3}$$

Gdje je n_x broj gena od kojih se svaki pojedinac sastoji. Mutacijska stopa p_m najčešće je mala vrijednost kako se dobra rješenja ne bi bespotrebno izobličavala.

3.2. Vrste genetskih algoritama

Kada se radi na realnim problemima male su šanse da postoji nekakav *zlatni čekić*, odnosno varijanta algoritma koja će podjednako dobro rješavati različite probleme. GA su iznimno uspješni u velikom rasponu domena, ali to ne znači da će trenutno ili uspješno rješavati sve domene problema. Upravo iz tog razloga su se GA grupirali na četiri osnovne varijante opisane u daljnjem tekstu [7].

3.2.1. Generacijski GA

Generacijski GA (engl. *Generational GA*), skraćeno (GGA), varijanta je genetskog algoritma koji započinje s nasumičnom početnom populacijom veličine *p* koja potom stvara i mutira *p* potomaka. Strategija zamjene generacija zamjenjuje sve roditelje s potomcima. Sukladno tome nema preklapanja među starom i novom populacijom pa je elitizam[3] jednak nuli. GGA koristi selekciju natjecanjem objašnjenu u poglavlju 3.1.4 za nasumičan odabir kandidata iz populacije koji potom kreiraju jednog potomka. Proces se ponavlja sve dok broj potomaka ne bude jednak roditeljskoj populaciji, tada se cijela roditeljska populacija briše. Algoritam se tako izvršava iterativno sve dok ne bude zadovoljen neki od postavljenih uvjeta zaustavljanja.

Svaka mutacija i rekombinacija mogu rezultirati smanjenjem ili poboljšanjem kvalitete nastalog pojedinca. Zato što uvažavamo najveću kvalitetu pojedinaca prilikom selekcije, njihov doprinos u sljedećim generacijama omogućuje globalni napredak. [8]

3.2.2. Stabilno-Stanje GA

Genetski algoritam stabilnog stanja (engl. *Steady-State GA*), skraćeno (SSGA), vrsta je algoritma koja nema generacije. Razlikuje se od GGA po tome što ne koristi selekciju natjecanjem za odabir pojedinaca, te umjesto dodavanja djece u novu generaciju samo izmjenjuje postojeću. Algoritam prvo odabire nasumično dva roditelja te oni stvaraju potomka. Ukoliko potomak ima veću kvalitetu od najslabijeg (engl. *worst fit*) pojedinca tada se oni zamjenjuju. Najčešće se dva potomka stvaraju

[3] Elitizam – označava broj superiornih pojedinaca u populaciji koji se prenose u sljedeću generaciju zbog očuvanja kvalitetnog genetskog materijala.

po ciklusu. SSGA je brži od GGA zato što za svaki ciklus napravi samo jedno vrednovanje funkcije po potomku, dok GGA mora napraviti *p* (veličina populacije) vrednovanja svaku generaciju.

3.2.3. Stabilni-Generacijski GA

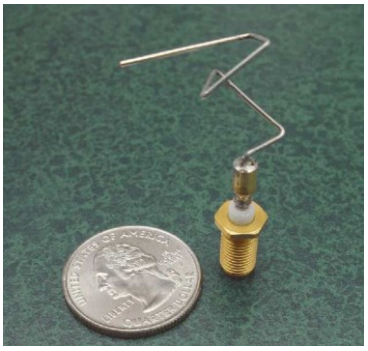
Stabilni generacijski GA (engl. *Stady-Generational GA*), skraćeno (SGGA), sličan je SSGA s obzirom da također nema generacije. Na početku odabire dva roditelja iz populacije kako bi stvorio potomka. Umjesto da potomak zamjenjuje najslabijeg pojedinca ili jednog od roditelja kao što je slučaj kod SSGA, ovdje potomak zamjenjuje nasumičnog pojedinca unutar populacije koji nije najsnažniji (engl. *best fit*). Prednost ove varijante algoritma je da također radi samo dva vrednovanja funkcije po ciklusu.

3.2.4. (μ+ μ) GA

(μ+ μ) GA iz populacije veličine *p* odabire nasumično dva roditelja binarnom selekcijom (engl. *binary tournament selection*) stvarajući tako potomke sve dok broj potomaka ne bude jednak broju *p*. Tada algoritam kreira novu populaciju spajajući potomke i roditelje u zajedničku populaciju. Potom odabire najbolje pojedince redom sve dok novonastala populacija ne naraste na broj *p*. Na ovaj način algoritam sastavlja populaciju samo sa najboljim pojedincima iz dvije generacije. Iako postoji dosta vrednovanja funkcije po ciklusima, prednost ove varijante je to što su najbolji pojedinci garantirani u novoj populaciji, za razliku od nasumičnih odabira i zamjena gdje je moguće čak završiti sa populacijom koja ima nižu kvalitetu.

3.3. Primjena genetskih algoritama

Genetski algoritmi imaju široku primjenu u današnjici, najviše zbog njihove jednostavnosti te zadovoljavajućih performansi. Jedna od čestih primjena je dizajniranje automobila u automobilskoj industriji, gdje takvi algoritmi dizajniraju najbolje vrste materijala koji će zadovoljiti određene zahtjeve (čvrstoća, mala masa i sl.) te oblike karoserija kako bi automobili bili što aerodinamičniji [9]. U vidu bilo kakve optimizacije, kao npr. usmjeravanje prometa i



Sl. 3.9. Evolucijski razvijen odašiljač (izvor: [10])

opreme (problem trgovačkog putnika) koriste se genetski algoritmi. Na slici 3.9. prikazan je odašiljač kojeg je NASA razvila 2006. godine [10] genetskim algoritmima s ciljem stvaranja odašiljača što većeg dometa. Nadalje, robotika, strojno učenje, neuronske mreže, obrada slike, enkripcija podataka, DNK analiza i još puno drugih područja su iznimno pogodna za primjenu genetskih algoritama.

4. OSTVARENO PROGRAMSKO RJEŠENJE

Programsko rješenje ovog rada pisano je u C programskom jeziku, korištenjem samo osnovnih biblioteka propisanih standardom. Korišteni IDE program je Microsoft Visual Studio 2022 Community Edition.

4.1. Princip rada programskog rješenja

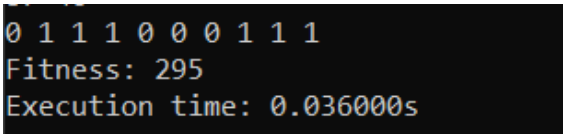
Korištena je verzija generacijskog genetskog algoritma. Stanje unutar torbe predstavljeno je strukturom *genom* (slika 4.1.) koja sadrži polje u kojem svaki element polja (*gen*) predstavlja jedan predmet unutar torbe. Ukoliko se predmet nalazi u torbi vrijednost gena je 1, a u suprotnom 0. Veličina polja jednaka je zbroju svih predmeta koje možemo staviti u torbu (numberOfItems).

```
typedef struct
{
    int sequence[numberOfItems];
}genome;
```

Sl. 4.1. Prikaz osnovne strukture algoritma

Nadalje, broj pojedinaca u svakoj generaciji (veličina generacije) je fiksna te je proizvoljno stavljena na 64. Bitno je da broj ne bude premal, jer je tada mala raspršenost pojedinaca, a ni prevelik, jer tada ispaštaju performanse. Algoritam koristi selekciju natjecanjem (poglavlje 3.1.4 - *tournament selection*) kod koje je veličina grupe od n_{ts} pojedinaca jednaka veličini generaciji podijeljeno sa 10. Što znači ako se generacija sastoji od 64 pojedinca, $n_{ts} = 6$. Algoritam koristi rekombinaciju u dvije točke (*two point crossover*) i stopa elitizma je nula. Ostali parametri su promjenjivi te njihova vrijednost ovisi o broju predmeta za koje se pokušava naći rješenje, pa će oni biti definirani neposredno prije svakog testnog primjera. Uvjet zaustavljanja algoritma je fiksan broj generacija koje se zadaju preko makro konstante na početku programa.

Ulazne vrijednosti programa su broj predmeta koji se mogu staviti u torbu, zatim u svakom novom redu vrijednost te masa svakog od predmeta odvojenih razmakom.



Sl. 4.2. Izlazne vrijednosti programa

Izlazne vrijednosti programa su redom prikaz najboljeg pojedinca u završnoj generaciji, vrijednost njegove funkcije dobrote te vrijeme izvršavanja algoritma u sekundama kao što je prikazano na slici 4.2.

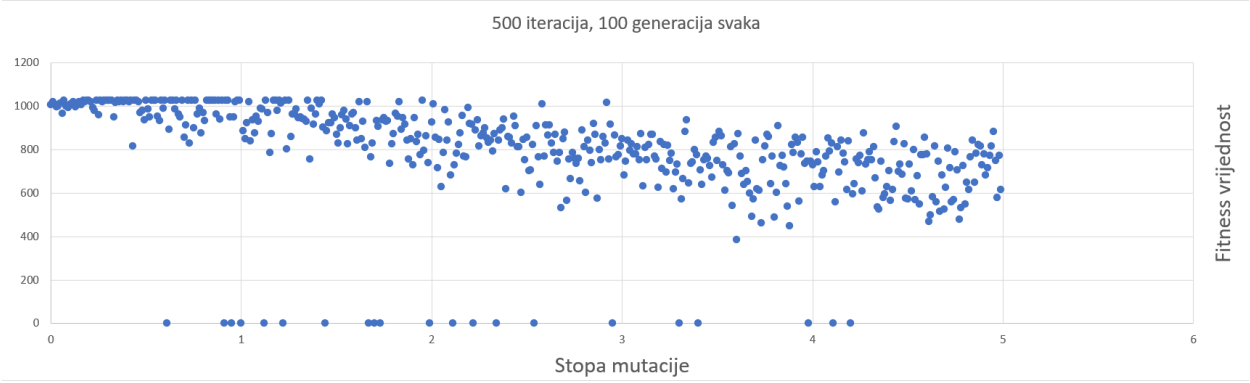
4.2. Usporedba točnosti rješenja

Za potrebe vrednovanja točnosti algoritma definirat ćemo uz par dobro poznatih rješivih testnih primjera i neke koji su nerješivi, tj. ne postoje načini kako ih egzaktno riješiti. Set testnih primjera sastojat će se od 3 primjera koji variraju u veličini kako bi što bolje testirali algoritam u različitim režimima rada. Svi rezultati su prikazani grafički, a kako bi se postigla što bolja dosljednost, svaki testni slučaj se pokreće veći broj puta te se mjere statistička obilježja svih pokretanja. Grafovi su prikazani tako da na x-osi leži broj generacija ili stopa mutacije, a na y-osi je vrijednost funkcije dobrote. Sve ostale vrijednosti su fiksne za sva pokretanja na određenom primjeru te su zapisana neposredno uz graf (stopa mutacije, veličina populacije...).

4.2.1. Prvi testni primjer

Prvi testni primjer sastoji se od 20 predmeta, s obzirom da je vrlo mal broj predmeta njega se može riješiti egzaktno metodom iscrpnog pretraživanja pa ga isprva nema smisla razmatrati, ali tu je zbog usporedbe vremenske složenosti s klasičnom metodom te zbog provjere radi li GGA na malim primjerima optimalno.

Najprije, potrebno je definirati stopu mutacije svakog pojedinca, pri kojoj se dobivaju optimalna rješenja. S ciljem pronalaska najbolje stope mutacije pokreće se 500 iteracija algoritma, ali s različitom stopom mutacije koja svaku iteraciju raste za 0.01% pa bude u rasponu od 0% do 5% što je prikazano grafički na slici 4.3. Važno je napomenuti da se uzima što manji broj generacija prilikom traženja optimalne mutacijske stope kako ne bi sva rješenja natjerali na konvergenciju ogromnim brojem generacija. Broj generacija u ovom primjeru je 100. Optimalna vrijednost za ovaj primjer iznosi 1024. Kao što je vidljivo iz slike vrlo male stope mutacije prilično točno rješavaju zadani problem, dok kasnije kod većih stopa (2% na više) dobiva se raspršenje rješenja



Sl. 4.3. Vrijednost kvalitete u ovisnosti o stopi mutacije

te se točnost smanjuje daljnjim povećavanjem mutacije. Također iz slike je vidljivo da se najbolje kvalitete pojedinaca postižu za raspon mutacija od 0.1% do 0.5% ovisno o veličini ulazna, prema tome prvi testni primjer je poslužio kao pokazatelj raspona kvalitetne stope mutacije za ovakav tip algoritma.

Kod usporedbe vremena potrebnog za izvršavanje u odnosu na klasičnu metodu već i na ovako malim primjerima postoji značajan napredak u brzini. U poglavlju 2.2 vidljivo je iz tablice 2.1. da je za 20 predmeta klasičnom pristupu potrebna jedna sekunda, dok GGA pronalazi točno rješenje u 100 generacija za 0.218 sekundi što je približno 5 puta brže.

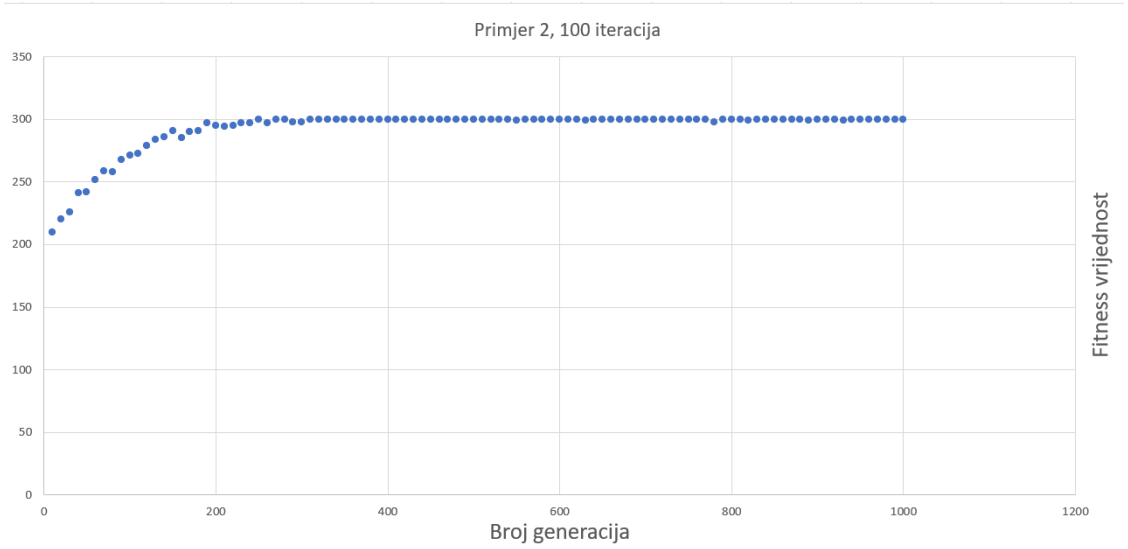
4.2.2. Drugi testni primjer

Logika iza drugog testnog primjera je da se postavi torba s dovoljno velikim kapacitetom tako da unutra stanu svi predmeti te je na taj način moguće i za velike brojeve predmeta znati egzaktno rješenje upravo zato što je trivijalno, suma svih vrijednosti predmeta, no naravno da to algoritam „ne zna“ te mora doći do tog kao i do svakog drugog rješenja - evolucijom.

Drugi testni primjer se sastoji od 300 predmeta s kapacitetom torbe od 301 te svaki predmet ima jednaku težinu i vrijednost koji iznose 1. Vrlo je lako sada odrediti ukupnu vrijednost predmeta koji stanu u torbu tj. optimalno rješenje, ono iznosi 300.

Stopa mutacije svakog pojedinca je 0.1%. Na slici 4.4. prikazana je ovisnost funkcije dobrote o broju generacija te je vidljivo da svi pokušaji nakon 300 generacija daju točno rješenje. To govori da algoritma rješava ovaj testni primjer optimalno u 300 generacija čije vrijeme izvođenja iznosi 0.447s. Dakako da ovo nije pretežak primjer problema putne torbe jer se mogu svi predmete staviti u torbu pa algoritam nikad neće morati micati gene, odnosno uvijek će se u novim generacijama

dodavanjem gena pojedincima povećavati kvaliteta što uvelike olakšava pronalazak točnog rješenja.



Sl. 4.4. Točnost drugog primjera sa porastom broja generacija

4.2.3. Treći testni primjer

Ovaj testni primjer je nasumično generiran te se pouzdano ne zna njegovo optimalno rješenje. Primjer se sastoji od 1,000 predmeta, gdje svaki ima težinu u rasponu od 1 do 10,000 i vrijednost u rasponu od 1,000 do 10,000. Kapacitet torbe je 2,317,485 što je točno pola ukupne težine svih predmeta. Drugim riječima, prosječno bi trebalo stati u torbu pola predmeta gledajući samo kapacitete torbe i težine predmeta. Kako je svaka vrijednost predmeta u gore navedenom rasponu, a korištena je uniformna nasumična distribucija, prosječni predmet bi trebao imati vrijednost od 5,500 te bi u torbu trebalo prosječno stati pola svih predmeta. To dovodi do teorijske prosječne ukupne vrijednosti od $5{,}500 * 500 = 2{,}750{,}000$. Kako je ovo optimizacijski algoritam ovaj broj predstavlja donju granicu i nju se pokušava optimizirati. Gornja granica bi bila suma svih vrijednosti predmeta koja iznosi 5,572,268. Naravno da je gornju granicu nemoguće postići, ali svakako želimo da rješenja budu što bliže gornjoj granici nego prosječnoj, donjoj.

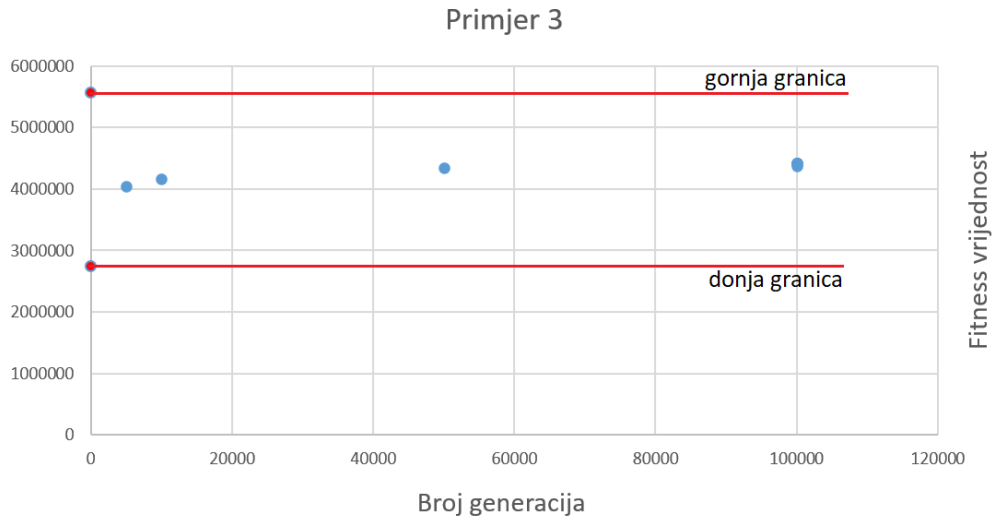
Tablica prikazuje vrijednosti funkcije dobrote najboljih pojedinaca nakon izvršavanja algoritma za u tablici definiran broj iteracija pri definiranoj stopi mutacije te ukupno trajanje izvođenja algoritma. Prvobitno je postavljena stopa mutacije od 0.1% te su prva 4 izvođenja odrađena s tom stopom mutacije. Nakon izvođenja algoritma s brojem iteracija od 100,000 potrebno je postepeno podizati ili smanjivati stopu mutacije kako bi se pronašla najpovoljnija za ovaj primjer. Kao što je vidljivo iz tablice podizanjem mutacijske stope dobivaju se bolja rješenja (primjerice 8. i 9.

Tablica 4.1. Izlazne veličine 10 pokretanja algoritma

Redni broj	*Kvaliteta*	*Vrijeme[s]*	*Broj generacija*	*Stopa mutacije[%]*
1	*4,048,135*	*23.62*	5,000	0.1
2	*4,162,723*	*47.31*	10,000	0.1
3	*4,348,368*	*233.77*	50,000	0.1
4	*4,379,856*	*464.94*	100,000	0.1
5	*4,400,555*	*464.86*	100,000	0.2
6	*4,414,311*	*464.55*	100,000	0.3
7	*4,420,727*	*464.88*	100,000	0.4
8	*4,423,030*	*483.12*	100,000	0.5
9	*4,423,889*	*506.62*	100,000	0.6
10	*4,415,741*	*470.67*	100,000	0.7

pokretanje), analogno, smanjivanjem stope mutacije kvarimo rješenja. S druge strane prevelikom stopom mutacije rješenja opet opadaju (10. pokretanje).

Najbolje rješenje je dobiveno pri stopi mutacije od 0.6% uz 100,000 generacija, čije je izvršavanje



Sl. 4.5. Usporedba dobivenih rješenja s postavljenim granicama

trajalo 506.62s te ono iznosi 4,423,889. Ako se uzmu u obzir granice sa početka poglavlja te se rješenja prikažu grafički kao na slici 4.5. vidljivo je da su rješenja u gornjoj polovici postavljenih granica. Ako se uzme u obzir najbolje rješenje, koje ima 4,423,889 vrijednost predmeta u torbi, naspram idealnog 5,572,268 tj. da sve predmete stavimo u torbu, dobivenim rješenjem se zapravo uspjelo staviti 79.39% svih predmeta u torbu čiji je kapacitet 50% ukupne težine svih predmeta. Prema tome ovo rješenje se može smatrati zadovoljavajuće.

4.3. Analiza složenosti programskog rješenja

Prema [11] malo je vjerojatno da postoji tehnika koja će rješavati sve instance NP-teških problema u vremenu koje raste linearno sa ulaznom veličinom. Prema toj izjavi niti jedan algoritam ne može u linearnom vremenu riješiti problem putne torbe egzaktno, pa tako ni genetski algoritmi.

Tablica 4.2. Složenost osnovnih funkcija algoritma

Naziv funkcije	*Vremenska složenost*
FitFunction()	O(M)
TournamenSelection()	O(M)
TwoPointCrossover()	O(M)
Mutate()	O(M)

Prilikom analize složenosti algoritama uvijek se gleda najslabija karika u lancu, odnosno funkcija ili dio algoritma koji ima najveći vremensku složenost. Konkretno, korišteni algoritam se sastoji od niza funkcija koje su prikazane u tablici gdje je M broj predmeta u torbi.

Kao što je vidljivo iz tablice sve osnovne funkcije (operatori) su linearni, tj. njihova složenost je O(M). Ono što čini pravu složenost algoritma je broj pozivanja svakog od tih operatora. Svaki od operatora poziva se za svaku generaciju pa vrijeme izvršavanja najviše ovisi o broju generacija N. Pri tome neke funkcije se pozivaju za svakom pojedincu, no broj pojedinaca je fiksan i najčešće vrlo mal (64). Kada to uzmemo u obzir, ukupna vremenska složenost je O(N*M) tj. ovisi o broju generacija i o broju predmeta.

5. ZAKLJUČAK

Ovim radom pokazano je korištenje genetskih algoritama za rješavanje NP-teškog problema putne torbe. Koristeći klasičnu metodu rješavanja dobivaju se optimalna rješenja, no samo za mali broj predmeta zbog velike vremenske složenosti. Od osnovna četiri algoritma navedena u radu, praktično je implementiran generacijski algoritam koji je potom testiran te je dao zadovoljavajuće rezultate s obzirom na velik broj predmeta. Prilikom implementacije ovakvog algoritma važno je paziti na vrstu selekcije te način na koji se stvaraju potomci jer ta dva parametra utječu na cjelokupnu točnost algoritma, te je teško testirati koji od dva parametra treba izmijeniti. Stopa mutacije također utječe na preciznost ali ona je samo broj pa ju se može povratnom vezom prilagoditi na najprikladnije vrijednosti, kao što je u radu i učinjeno. Genetski algoritmi nisu zlatni čekić kojim treba napadati sve probleme, posebno zbog toga što mnogo problema nije kompatibilno za implementaciju na takav način te bolja rješenja treba tražiti u drugim porodicama algoritama (pohlepni algoritmi, podjeli i vladaj, dinamičko programiranje i sl.). Isto tako, ne rješavaju uvijek egzaktno problem, odnosno daju suboptimalna rješenja, ali ipak, zbog svoje jednostavnosti implementacije i efikasnosti su vrlo popularni u optimizacijskim problemima.

LITERATURA

[1] S. Martello i P. Toth, Knapsack problems: algorithms and computer implementations, Bologna: John Wiley & Sons, 1990.

[2] H. Kellerer, U. Pferschy i D. Pisinge, "Multidimensional knapsack problems." Knapsack problems., Berlin: Springer, Berlin, Heidelberg, 2004, pp. 235-236.

[3] A. E. Eiben i C. A. Schippers, »On evolutionary exploration and exploitation,« *Fundamenta Informaticae,* svez. 35, br. 1-4, pp. 35-50, 1998.

[4] B. Mishra i R. K. Patnaik, Genetic Algorithm and Its Variants: Theory and Applications, Rourkela: National Institute of Technology, 2009.

[5] A. P. Engelbrecht, Computational intelligence: an introduction, Pretoria: John Wiley & Sons, 2007.

[6] R. Champlin, Selection Methods of Genetic Algorithms., Computer Science at Digital Commons, 2018.

[7] A. Jenkins, V. Gupta, A. Myrick i M. Lenoir, Variations of genetic algorithms, arXiv preprint, 2019.

[8] K. Staats, Genetic Programming Applied to RFI Mitigation in Radio Astronomy. MS thesis, Cape Town: University of Cape Town, 2016.

[9] »Educba,« [Mrežno]. Available: https://www.educba.com/what-is-genetic-algorithm/. [Pokušaj pristupa 29 6 2022].

[10] G. S. Hornby, D. S. Linden i J. D. Lohn, Automated Antenna Design with Evolutionary, California: American Institute of Aeronautics and Astronautics, 2006.

[11] N. F. McPhee, R. Poli i W. B. Langdon, A field guide to genetic programming, Morris, Minnesota, SAD: University of Minnesota Morris Digital Well, 2008.

SAŽETAK

Problem putne torbe (engl. *knapsack problem*) je optimizacijski NP-teški problem kod kojeg se pokušava naći maksimalna vrijednost predmeta koji se mogu staviti u torbu, pod uvjetom da se ne premaši kapacitet torbe. Algoritama za rješavanje je ovog problema je mnogo no pristup rješavanju u ovom radu su genetski algoritmi koji su meta heuristički algoritmi pretraživanja stvoreni po uzoru na prirodnu evoluciju. Konkretno korišten je generacijski algoritam, implementiran u C programskom jeziku, te je odrađena usporedba složenosti i točnosti takvog algoritma s klasičnim pristupom rješavanja.

Ključne riječi: funkcija dobrote, genetski algoritmi, genomi, mutacija, problem putne torbe

ABSTRACT

Solving knapsack problems using genetic algorithms

Knapsack problem is one of most popular NP-hard problems in optimization theory, in which the goal is to find right combination of items that produce maximal value so that they do not exceed given knapsack capacity. There are many algorithms for solving this problem, but main approach to solvnig in this paper are genetic algorithms, which are metaheuristic algorithms created along the lines of natual evolution. In particular, a generational algorithm was used, implemented in C programming language, and a comparison of the complexity and accuracy of such algorithm with the classic bruteforce approach of solving was done.

Key words: fitness function, genetic algorithm, genomes, knapsack, mutation