SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Marija Ilijaš

MEMETIČKI ALGORITMI

Diplomski rad

Voditelj rada:
doc. dr. sc. Nela Bosner

Zagreb, ožujak 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Evolucijsko programiranje je grana računalnih znanosti koja se bavi algoritmima baziranima na principima Darwinove teorije o prirodnoj selekciji, te također nalazi inspiraciju u molekularnoj genetici. Mnoge vrste tijekom povijesti svijeta su evoluirale kako bi se što bolje prilagodile okolini na kojoj obitavaju. Na isti način mi želimo osmisliti evolucijski algoritam koji će inicijalnu populaciju adaptirati na svaku novu okolinu. Često, ali ne i uvijek, okolina će nam predstavljati problem koji želimo riješiti, dok će jedinke predstavljati rješenja. Intuitivno, koliko su dobro jedinke prilagođene okolini na kojoj obitavaju, toliko je dobro rješenje zadanog problema.

Memetički algoritmi, koji su glavna tema ovog rada, zapravo su evolucijski algoritmi kombinirani s drugim tehnikama ili nadograđeni nekom drugom metodom ili strukturom podataka. Njihova upotreba je vrlo uspješna u praksi i posjeduje veliki potencijal za daljnja istraživanja.

U nastavku ćemo prvo opisati evolucijske algoritme i njihove značajke. Nakon toga prelazimo na memetičke algoritme, njihove karakteristike, svojstva i primjere. Na kraju, prilažemo primjer izrađen u MATLAB-u, analizu korištenog algoritma te dobivene rezultate.

Više o teoriji evolucijskih algoritama može se pronaći u knjizi [22] iz koje je proizašla i velika većina ovog rada.

Poglavlje 1

Evolucijski algoritmi

1.1 Motivacija

Postoje različite varijante evolucijskih algoritama, međutim osnovna ideja u pozadini svih tehnika je ista: imamo danu populaciju jedinki na nekoj okolini koja ima ograničene resurse; ograničenost resursa dovodi do borbe što izaziva prirodnu selekciju (opstanak najjačih[1]), a zauzvrat dobivamo bolje sposobnosti cjelokupnog stanovništva.

Za početak, razmotrimo evoluciju u biološkim okvirima. Imamo okolinu naseljenu populacijom nekih jedinki koje nastoje preživjeti i reproducirati se. Podobnost (eng. fitness) tih jedinki određena je okolinom gdje obitavaju i odnosi se na to koliko je svaka od njih uspješna u ostvarivanju svojih ciljeva. Drugim riječima, podobnost predstavlja šansu da jedinka preživi u danoj okolini i razmnoži se.

Temeljni način evolucijskog rješavanja problema oslanja se na metodu pokušaja i pogreške. U tom kontekstu, naš zadatak se zapravo svodi na to da odredimo skup kandidata za rješenja. Njihova kvaliteta, odnosno koliko će dobro riješiti dani problem, određuje vjerojatnost da će preživjeti i sudjelovati u procesu pronalaska budućih kandidata za rješenja. Dakle, ako promatramo problem u okviru algoritma, onda je kvalitetna jedinka zapravo dobra aproksimacija traženog rješenja, koja se u svakoj sljedećoj iteraciji (potomstvo te jedinke) sve više približava točnom rješenju. Analogija između biološke evolucije i njenog korištenja u rješavanju problema dana je tablicom 1.1.

[1]Ovaj termin se najčešće shvaća doslovno, što je zapravo pogrešno. To bi značilo da najjača, tj. najpodobnija jedinka *uvijek* preživi. Međutim, priroda (i evolucijsko programiranje koje je kreirano po uzoru na nju) nije deterministički određena i sadrži mnoge slučajnosti, pa tako nije uvijek istina da najjači opstaje.

EVOLUCIJA		RJEŠAVANJE PROBLEMA
Okolina	$\longleftrightarrow$	Problem
Jedinka	$\longleftrightarrow$	Kandidat za rješenje
Podobnost	$\longleftrightarrow$	Kvaliteta

Tablica 1.1: Osnovni prikaz povezivanja evolucije s rješavanjem problema u kontekstu evolucijskog programiranja

1.2 Što je evolucijski algoritam?

Dakle, imamo skup jedinki na nekom prostoru i želimo svaku od njih zasebno procijeniti s obzirom na njenu kvalitetu. Stoga, traženu kvalitetu svake jedinke možemo promatrati kao funkciju koju želimo maksimizirati. U kontekstu našeg problema, uzmimo mjeru podobnosti (eng. fitness measure) kao funkciju kojom ćemo mjeriti kvalitetu jedinke, tj. što je podobnost viša, to je kvaliteta bolja. Dakle, funkcijom kvalitete koju smo upravo opisali i za čiju domenu uzimamo jedinke iz određene generacije, zapravo biramo najpodobnije kandidate za rješenja koji će sudjelovati u stvaranju sljedeće generacije. Stvaranje nove generacije odvija se primjenom rekombinacije i/ili mutacije na odabrane kandidate.

Rekombinacija ili križanje je operator koji se primjenjuje na dva ili više kandidata (tzv. roditelje) proizvodeći jednog ili više novih kandidata (djecu). Mutacija se pak primjenjuje na jednog kandidata i rezultira ponovno jednim novim kandidatom. Dakle, rekombinacijom i mutacijom roditelja dobivamo skup novih kandidata za rješenja (potomstvo). Novi kandidati se, s obzirom na podobnost i starost, natječu sa starim kandidatima za mjesto u novoj generaciji. Ovaj proces se ponavlja dok god se ne nađe zadovoljavajuća kvaliteta (točno rješenje) ili dok se ne ispuni neki prethodno postavljeni zahtjev (npr. maksimalni broj iteracija i sl.).

Postoje dva glavna čimbenika za osnovu evolucijskih sustava:

- Operatori varijacije (rekombinacija/mutacija)- stvaraju potrebnu raznolikost unutar populacije i samim time omogućavaju pojavu novih obilježja
- Selekcija- djeluje kao sila unutar populacije koja povećava prosječnu kvalitetu novih kandidata za rješenja

Kombiniranje ovih dvaju procesa dovodi do poboljšanja podobnosti iz generacije u generaciju. Dakle, evolucijski proces rezultira populacijom koja je sposobna najbolje se prilagoditi okolini što se i odražava na broj potomaka (koji u tom slučaju raste).

Bitno je naglasiti da su mnoge komponente evolucijskog procesa stohastičke[2]. Tijekom prirodne selekcije najbolje jedinke nisu odabrane deterministički[3] i stoga je moguće da čak i slabije jedinke postanu roditelji ili da prežive. U procesu rekombinacije također nasumično odabiremo koje dijelove roditelja ćemo križati. Slično se događa i kod mutacije, odnosno odabir dijelova koje treba mijenjati, pa i novi dijelovi koji će ih zamijeniti, odabrani su na slučajan način.

Opća shema evolucijskog algoritma dana je pseudokodom u tablici 1.2.

START
INICIJALIZIRAJ *populaciju* sa slučajnim kandidatima za rješenja
PROCIJENI svakog kandidata
DOK (*uvjet zaustavljanja* nije zadovoljen) PONAVLJAJ
1 ODABERI roditelje
2 REKOMBINIRAJ parove roditelja
3 MUTIRAJ dobiveno potomstvo
4 PROCIJENI nove kandidate
5 ODABERI jedinke za novu generaciju
KRAJ

Tablica 1.2: Pseudokod koji daje prikaz glavne sheme evolucijskog algortima

Očito je da ovakva shema spada u kategoriju tzv. generate-and-test algoritama, odnosno kao što smo već i spomenuli, baziramo se na metodu pokušaja i pogreške. Funkcija evaluacije (podobnosti) daje heurističku procjenu[4] kvalitete rješenja, a operatori varijacije i selekcija upravljaju procesom pretraživanja. Evolucijski algoritmi sadrže niz svojstava zbog kojih ih možemo svrstati u generate-and-test metode, a neka od njih su:

- Evolucijski algoritmi obrađuju čitav skup kandidata istovremeno jer se baziraju na cjelokupnoj populaciji
- Većina evolucijskih algoritama koristi rekombinaciju, odnosno križaju informacije dvaju ili više kandidata za rješenja kako bi stvorili jednog novog
- Evolucijski algoritmi su stohastički[5]

[2]Stohastički proces (grč. *στοχαστικός*: koji pogađa)- proces koji u sebi sadrži elemente neuređenosti ili slučajnosti. [33]

[3]Determinstički- uvjetovano i nužno određeno, neslobodno, neslučajno. [11]

[4]Heuristički (prema grčkoj riječi *εὑρίσχειν*: nalaziti, otkrivati)- to je bilo koji pristup rješavanju problema koji koristi praktične metode koje ne garantiraju optimalno rješenje, ali daju dovoljno dobre aproksimacije i obično su te metode efikasnije od onih koje daju optimalno rješenje.

[5]Stohastički algoritam- algoritam koji barem u jednom dijelu izvršavanja donese neku odluku slučajnim odabirom. [31]

1.3 Komponente evolucijskog algoritma

U ovom odjeljku ćemo detaljno razraditi evolucijske algoritme. Postoje brojne komponente, procedure i operatori koje moramo navesti i objasniti njihovo značenje i ulogu kako bismo mogli definirati određene evolucijske algoritme. Krenimo od pseudokoda u tablici 1.2 i navedimo najvažnije komponente:

- reprezentacija jedinki
- funkcija evaluacije, tj. procjene (ili funkcija podobnosti)
- populacija
- mehanizam odabira roditelja
- operatori varijacije- rekombinacija(križanje) i mutacija
- mehanizam odabira jedinki za sljedeću generaciju (odabir nasljednika)

Kako bismo stvorili kompletan algoritam koji dobro radi i izvršava se, potrebno je specificirati svaku komponentu i definirati način na koji se inicijalizira. Ukoliko želimo da algoritam prestane s izvršavanjem u nekom trenutku, moramo odrediti uvjet zaustavljanja.

1.3.1 Reprezentacija jedinki

Prvi i osnovni korak u definiranju evolucijskog algoritma je napraviti poveznicu između originalnog problema i evolucijskog načina rješavanja problema. To često znači da trebamo pojednostavniti ili sažeti neka gledišta iz stvarnog svijeta kako bismo dobili dobro definirani 'opipljivi' problem unutar čijeg konteksta postoji rješenje koje se može procijeniti. Najprije treba odlučiti na koji način ćemo specificirati moguća rješenja te kako ćemo ih pohranjivati u računalo da bismo mogli njima manipulirati. Reći ćemo da se objekti koji tvore moguća rješenja originalnog problema odnose na fenotip, a njihove iskodirane podatke unutar evolucijskog algoritma nazivat ćemo genotipom. Ovaj korak naziva se reprezentacija jer pridružuje fenotip s genotipom koji ga predstavlja (reprezentira). Za ilustraciju, uzmimo neki optimizacijski problem čija su moguća rješenja cijeli brojevi. Dakle, dani skup cijelih brojeva je zapravo skup fenotipova. Ukoliko bi netko odlučio zapisati te brojeve u njihovom binarnom zapisu, onda bi npr. genotip 10010 predstavljao broj 18 iz skupa fenotipova. Važno je uočiti da se prostor fenotipova može uvelike razlikovati od prostora genotipova i da se čitavo evolucijsko pretraživanje odvija u prostoru genotipova. Rješenje, odnosno dobar fenotip, dobiva se dekodiranjem najboljeg genotipa nakon završetka izvršavanja algoritma. Stoga je poželjno da optimalno rješenje problema (fenotip), bude prikazano u danom prostoru genotipova. U stvari, budući da općenito ne možemo

unaprijed znati kako izgleda rješenje, težimo tome da sva moguća rješenja možemo prikazati u tom skupu[6].

1.3.2 Funkcija evaluacije (funkcija podobnosti)

Uloga funkcije evaluacije jest iznositi zahtjeve koje bi populacija trebala zadovoljiti. Ona čini osnovu za selekciju i samim time omogućava pojavu poboljšanja. Iz perspektive rješavanja problema u evolucijskom smislu, ona predstavlja zadatak koji treba riješiti. Tehnički gledano, to je funkcija ili postupak kojim se genotipu dodjeljuje vrijednost kvalitete. U pravilu ta se funkcija sastoji od inverzne reprezentacije (od genotipa želimo dobiti odgovarajući fenotip) praćene mjerenjem kvalitete u prostoru fenotipova. U kontekstu našeg primjera, ako je zadatak naći cijeli broj x koji maksimizira x^2, onda podobnost genotipa 10010 možemo odrediti tako da nađemo odgovarajući fenotip ($10010 \rightarrow 18$) i kvadriramo ga : $18^2 = 324$.

U evolucijskom programiranju funkciju evaluacije uobičajeno zovemo funkcija podobnosti.

1.3.3 Populacija

Uloga populacije jest da sadrži (reprezentira) moguća rješenja. Populacija je multiskup[7] genotipova te ona čini jedinicu evolucije. Intuitivno, jedinke su statički objekti koji se ne mijenjaju niti prilagođavaju, već to radi cijela populacija kao takva. S obzirom na zastupljenost, populaciju definiramo na način da odredimo broj jedinki u njoj, odnosno njenu veličinu. U nekim sofisticiranijim evolucijskim algoritmima, populacija može sadržavati dodatne prostorne značajke, npr. kriterij udaljenosti ili relaciju susjedstva. Međutim, to baš i ne odgovara načinu na koje stvarna populacija evoluira na pojedinom geografskom području, pa u takvim slučajevima mora biti definirana neka dodatna struktura kako bi se populacija u potpunosti mogla odrediti.

U skoro svim primjenama evolucijskih algoritama, veličina populacije je konstantna i ne mijenja se tijekom evolucijskog pretraživanja. To stvara ograničenost resursa koja je potrebna da bi se stvorila borba i konkurencija među jedinkama. Operatori selekcije (odabir roditelja i odabir nasljednika) djeluju na nivou populacije. Općenito, oni procjenjuju cjelokupnu trenutnu populaciju i rade izbor na temelju onoga što je trenutno prisutno. Npr., najbolja jedinka *u trenutnoj populaciji* će biti odabrana za sudjelovanje u stvaranju nove generacije, dok će najgora jedinka *u trenutnoj populaciji* biti zamijenjena novom. Ovakvo

[6] U pogledu generate-and-test algoritama, to zapravo znači da je generator *potpun*

[7] Multiskup je skup u kojem se elementi mogu ponavljati. Npr. $M = \{1, 1, 1, 2, 3, 3, 4\}$ je konačni multiskup

djelovanje odvija se na razini populacije za razliku od operatora varijacije koji djeluju na jednog ili više roditelja.

Raznolikost populacije ili raznovrsnost je mjera koja broji prisutnost različitih rješenja, no bitno je reći da se ta mjera nikad ne odnosi samo na jednu značajku, nego uvijek na više njih.

Važno je također napomenuti da prisutnost samo jedne vrijednosti podobnosti nužno ne ukazuje na postojanje samo jednog fenotipa, jer više fenotipova može imati istu podobnost. Analogno, prisutnost samo jednog fenotipa ne ukazuje na postojanje isključivo jednog genotipa. S druge pak strane, prisutnost samo jednog genotipa implicira postojanje isključivo jednog fenotipa i jedne vrijednosti podobnosti.

1.3.4 Mehanizam odabira roditelja

Svrha odabira roditelja ili odabira partnera (eng. mate selection) jest da bi se unijele razlike među jedinkama na osnovi njihovih kvaliteta, a osobito kako bi se boljim pojedincima omogućilo da postanu roditelji sljedećoj generaciji. Jedinka postaje roditelj ako je odabrana da se podvrgne operacijama varijacije kako bi se stvorilo potomstvo. Zajedno s mehanizmom odabira nasljednika, odabir roditelja je zaslužan za nametanje poboljšanja kvalitete. U evolucijskom programiranju odabir roditelja je u pravilu probabilistički[8]. Prema tome, visokokvalitetne jedinke imaju veću šansu da postanu roditelji, od onih manje kvalitetnih. Unatoč tome, za jedinke lošije kvalitete ipak postoji mala, ali pozitivna vjerojatnost da postanu roditelji. Kada one ne bi dobile nikakvu šansu da postanu roditelji, cijeli sustav bi postao prepohlepan (eng. too greedy) i populacija bi zaglavila u lokalnom optimumu.

1.3.5 Operatori varijacije: rekombinacija i mutacija

Uloga operatora varijacije je stvaranje novih jedinki od starih. U pogledu odgovarajućeg prostora fenotipova, to se odnosi na stvaranje novih kandidata za rješenja. Iz perspektive generate-and-test pretraživanja, operatori varijacije preuzimaju korak generiranja. S obzirom na koliko jedinki djeluju, razlikujemo dvije vrste takvih operatora: unarni (mutacija) i n-arni (rekombinacija).

1.3.5.1 Mutacija

Unarni operator varijacije često se naziva mutacija. On se primjenjuje na jedan genotip i rezultira (blago) modificiranim mutantom- djetetom ili potomkom. Operator mutacije je uvijek stohastički: rezultat koji daje (dijete) ovisi o ishodu niza slučajnih odabira.

[8]Probabilistički odabir- za svakog člana populacije mora biti poznata vjerojatnost da bude izabran u uzorak; izbor svih elemenata mora biti lišen bilo kakvog utjecaja namjere ili odluke istraživača [20]

1.3.5.2 Rekombinacija (križanje)

Binarni operator varijacije naziva se rekombinacija ili križanje. Kao što samo ime govori, takav operator spaja informacije genotipova dvaju roditelja i time stvara jedno ili dva genotipa potomstva. Isto kao i mutacija, rekombinacija je stohastički operator: odabir koji dijelovi roditelja će biti kombinirani i na koji način će se to odvijati je slučajan.

n-arni operatori rekombiniranja za $n > 2$, dakle oni koji djeluju na više od dva roditelja, su matematički mogući i lako se implementiraju, ali nemaju biološke ekvivalente. Možda se upravo zbog tog razloga ne upotrebljavaju često, iako su već neke studije pokazale da takvi operatori imaju pozitivan učinak na evoluciju.

Princip rekombinacije je jednostavan: križajući dvije jedinke koje imaju različite, ali poželjne osobine, možemo proizvesti potomka koji sadrži kombinaciju obje od tih osobina. Ipak, operatori rekombiniranja u evolucijskim algoritmima se najčešće primjenjuju probabilistički, tj. vjerojatnost da se ne izvrše je jednaka nuli.

1.3.6 Mehanizam odabira jedinki za sljedeću generaciju (odabir nasljednika)

Slično kao kod mehanizma odabira roditelja, uloga odabira jedinki za sljedeću generaciju jest da se uvedu razlike među jedinkama s obzirom na njihovu kvalitetu. Ipak, ovaj mehanizam se provodi u jednom drugom stadiju evolucijskog ciklusa i to nakon što odabrani roditelji stvore potomstvo. Kao što smo već ranije spomenuli, veličina populacije je skoro uvijek konstantna kod evolucijskog programiranja i upravo to je razlog zašto moramo odabrati koje će jedinke moći tvoriti sljedeću generaciju. Odabir tih jedinki bazira se na tome koliko je svaka od njih podobna, što naravno ide u korist kvalitetnijim jedinkama, ali ne treba zanemariti niti dob koja je također bitan faktor. Za razliku od odabira roditelja koji je u pravilu stohastički, odabir nasljednika je često deterministički.

Odabir nasljednika često se naziva strategija izmjene generacija. U većini slučajeva možemo koristiti oba naziva, ali češće koristimo odabir nasljednika zbog dosljednosti terminologije.

1.3.7 Inicijalizacija

U većini evolucijskih algoritama inicijalizacija je jednostavna: prva populacija nastala je od slučajno odabranih jedinki. Međutim, početnu populaciju možemo zadati takvom da npr. ima višu podobnost, ali isplati li nam se zbog toga ulagati veći trud u stvaranje algoritma ili ne, uvelike ovisi o samom problemu kojeg treba riješiti.

1.3.8 Uvjet zaustavljanja (eng. termination condition)

Razlikujemo dva slučaja za odabir uvjeta zaustavljanja. Ako problem ima poznatu optimalnu razinu podobnosti, dobivenu pomoću optimuma dane funkcije cilja, tada bi (u idealnom svijetu) naš uvjet za prestanak izvršavanja bio pronalazak rješenja s upravo tom podobnošću. Ako pak znamo da naš problem, modeliran u stvarnom svijetu, sadrži određena pojednostavljenja ili šumove, onda možemo prihvatiti rješenje koje odstupa od optimalne podobnosti za neki $\varepsilon > 0$.

Međutim, budući da su evolucijski algoritmi stohastički i da najčešće nema garancije da se može dosegnuti takav optimum, ovakav uvjet možda nikada ne bude ispunjen i može se dogoditi da algoritam nikad ne prestane s izvršavanjem. Stoga, ono što treba učiniti jest dodati još jedan uvjet za kojeg smo sigurni da će u nekom trenutku zaustaviti algoritam. Neki od sljedećih kriterija se obično koriste u tu svrhu:

1. Dok ne istekne maksimalno CPU[9] vrijeme
2. Dok ukupan broj procjena podobnosti ne dostigne unaprijed postavljenu granicu
3. Ukoliko poboljšanje podobnosti ne prelazi neki određeni prag duži period izvršavanja (tj. unutar nekog većeg broja generacija)
4. Ukoliko raznolikost populacije padne ispod nekog zadanog praga

Tehnički gledano, kriterij zaustavljanja u takvim slušajevima je disjunkcija: nađena optimalna vrijednost *ili* zadovoljen dodatni uvjet. Ako problem nema poznati optimum, onda je dovoljno samo odabrati jedan od nabrojanih dodatnih uvjeta (ili neki drugi) za koje smo sigurni da će u jednom trenutku zaustaviti algoritam.

1.4 Simulacija rada evolucijskog ciklusa

Za ilustraciju načina rada evolucijskog algoritma prikazat ćemo detalje jednog ciklusa odabira i reprodukcije na jednostavnom problemu preuzetom od Goldberga [6]. Dakle, želimo maksimizirati vrijednosti od x^2 za cijele brojeve u rasponu $0-31$. Kako bismo mogli odvrtiti cijeli evolucijski ciklus, trebamo objasniti i obraditi svaku od komponenti evolucijskog algoritma: reprezentaciju, odabir roditelja, rekombinaciju, mutaciju i odabir nasljednika.

Što se reprezentacije tiče, koristimo jednostavno 5-bitno binarno kodiranje cijelih brojeva (fenotipova) u nizove bit-ova (genotipova). Za odabir roditelja uzet ćemo mehanizam proporcionalnih podobnosti, gdje je p_i vjerojatnost da je jedinka i iz populacije P odabrana

[9] CPU (central processing unit; hrv. "središnja jedinica za obradu", procesor), glavni je dio računala koji vođen zadanim programskim naredbama izvodi osnovne radnje nad podacima. [26]

da bude roditelj, i ta vjerojatnost je dana formulom $p_i = \frac{f(i)}{\sum_{j \in P} f(j)}$. Štoviše, možemo odlučiti zamijeniti cijelu populaciju u jednom potezu s potomstvom stvorenim od odabranih roditelja. U tom slučaju je naš operator odabira nasljednika jako jednostavan: sve postojeće jedinke su uklonjene iz populacije i sve nove jedinke su dodane u istu bez obzira na njihovu podobnost. To implicira da ćemo formirati onoliko potomaka koliko populacija ima članova.

S obzirom na našu odabranu reprezentaciju, operatore rekombinacije i mutacije ćemo uzeti što jednostavnijima. Mutaciju ćemo izvesti generiranjem slučajnog broja (iz uniformne distribucije na segmentu $[0, 1]$) za svaku poziciju bit-a. Svaki dobiveni slučajni broj uspoređivat ćemo s fiksnim pragom kojeg zovemo stopa mutacije (eng. mutation rate). Ukoliko je neki broj ispod tog praga, to znači da je vrijednost gena na odgovarajućoj poziciji obrnuta. Rekombinaciju ćemo implementirati klasičnim križanjem u jednoj točki. Takav operator se primjenjuje na dva roditelja i daje dva djeteta na način da odabere nasumičnu točku križanja negdje na danom nizu i sve bit-ove roditelja nakon te točke zamjenjuje.

Nakon što smo odredili esencijalne komponente, možemo izvršiti cijeli ciklus odabira i

Br. niza	Početna populacija	x	Podobnost $f(x) = x^2$	Vjerojatnost p_i	Očekivani broj	Stvarni broj
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
ZBROJ			1170	1.00	4.00	4
PROSJEK			293	0.25	1.00	1
MAX			576	0.49	1.97	2

Tablica 1.3: Prvi primjer za x^2; inicijalizacija, procjena i odabir roditelja

reprodukcije. Tablica 1.3 daje prikaz slučajne populacije s 4 genotipa, odgovarajućim fenotipima i podobnostima. Ciklus počinje odabirom roditelja koji će 'biti sjeme' za sljedeću generaciju. Šesti stupac u tablici 1.3 pokazuje očekivani broj kopija svake jedinke nakon odabira roditelja, što je dano formulom $\frac{f_i}{\bar{f}}$, gdje $\bar{f}$ označava prosječnu podobnost. Ovi brojevi predstavljaju vjerojatnosnu distribuciju iz koje slučajnim odabirom nekog uzorka stvaramo tzv. *bazen potencijalnih roditelja (eng. mating pool)*[10]. Stupac 'Stvarni broj' predstavlja broj kopija u tom bazenu potencijalnih roditelja, tj. pokazuje jedan od mogućih

[10]Ako iz populacije (kao jedinice evolucije) izdvojimo samo jedinke koje će proizvesti potomke, onda skup tih jedinki nazivamo *bazen potencijalnih roditelja (eng. mating pool)* [10]

ishoda.

Nadalje, odabrane jedinke su nasumično uparene i za svaki par odabrana je jedna slučajna

Br. niza	Bazen potencijalnih roditelja	Točka križanja	Potomstvo nakon križanja	x	Podobnost $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
ZBROJ					1754
PROSJEK					439
MAX					729

Tablica 1.4: Drugi primjer za x^2; križanje i procjena potomstva

točka na nizu. Tablica 1.4 daje rezultate križanja na danom bazenu potencijalnih roditelja i to za izmjenu točaka nakon četvrtog i drugog gena, zajedno s odgovarajućim podobnostima. Mutaciju primjenjujemo na potomstvo koje smo dobili križanjem.

Tablica 1.5 prikazuje 'ručno dobivene mutante'. U ovom slučaju mutacije su uzrokovale

Br. niza	Potomstvo nakon križanja	Potomstvo nakon mutacije	x	Podobnost $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
ZBROJ				2354
PROSJEK				588.5
MAX				729

Tablica 1.5: Treći primjer za x^2; mutacija i procjena potomstva

pozitivne promjene u podobnosti, ali treba naglasiti da u kasnijim generacijama, kako će broj jedinica rasti, mutacija će u prosjeku (ali ne uvijek) biti štetna.

Iako smo ovaj primjer ručno simulirali, on ipak prikazuje tipični napredak: prosječna podobnost porasla je s 293 na 588.5, a maksimalna s 576 na 729 nakon križanja i mutacije.

1.5 Primjer: Problem osam kraljica

Ideja ovog problema je kako smjestiti osam kraljica na šahovsku ploču 8x8 tako da se ni jedna od njih međusobno ne napada. Taj problem zapravo možemo i generalizirati u smislu da gledamo n kraljica na ploči dimenzija $n \times n$. Postoji više klasičnih pristupa ovom problemu zasnovanih na umjetnoj inteligenciji koji rade u konstruktivnom ili inkrementalnom[11] modu. Oni kreću tako da smjeste prvo jednu kraljicu, pa drugu, itd., te nakon smještanja n kraljica, pokušavaju smjestiti $(n + 1)$.-u kraljicu na mjesto gdje ne može napasti preostale. Ovdje koristimo 'backtracking' [12] metodu: ako ne postoji pozicija takva da na njoj $(n + 1)$.-a kraljica ne napada ostale, onda treba n-tu kraljicu pomaknuti na neku drugu poziciju.

Evolucijski pristup se uvelike razlikuje od maloprije opisanog, a ponajviše zato jer nije inkrementalni. Naši kandidati za rješenja su već gotove konfiguracije, tj. razmještaj svih 8 kraljica na ploči. Neka je prostor fenotipova P skup svih takvih konfiguracija (dakle, svi mogući razmještaji 8 kraljica na šahovskoj ploči). Očito je da većina elemenata tog skupa krši uvjet o nenapadanju među kraljicama. Kvaliteta svakog fenotipa $q(p)$, $p \in P$ najjednostavnije se može definirati kao broj parova kraljica koje se međusobno napadaju. Dakle, što manji je taj broj, to je bolji fenotip (konfiguracija), a vrijednost nula, $q(p) = 0$, ukazuje da smo našli dobro rješenje. Iz ove perspektive možemo formulirati prikladnu funkciju cilja koju treba minimizirati, a uz to znamo i optimalnu vrijednost. Iako nismo uopće definirali prostor genotipova, možemo zaključiti da pogodnost (koju želimo maksimizirati) genotipa g koji reprezentira fenotip p je nekakav inverz od $q(p)$.

Neka je genotip (ili kromosom) permutacija brojeva 1,2,...,8 i neka dobiveni $g = (i_1, ..., i_8)$ označava (jedinstvenu) konfiguraciju, gdje n-ti stupac sadrži točno jednu kraljicu smještenu u i_n-tom retku. Stoga, prostor genotipova G je skup svih permutacija brojeva 1,2,...,8.

Sljedeći korak je definiranje operatora mutacije i rekombinacije za našu reprezentaciju. Za mutaciju uzmimo operator koji nasumično bira dvije pozicije u danom kromosomu i zamijeni njihove vrijednosti. Dobar operator rekombinacije u ovom slučaju permutiranja i nije tako očit, ali u tablici 1.6 prikazan je mehanizam koji će stvoriti dva djeteta od dva roditelja.

Operacije koje smo opisali i dali u tablici 1.6 nisu nužno efikasne, nego tu služe kao primjeri primjenjivih operacija s obzirom na danu reprezentaciju. Nadalje, mehanizmi odabira roditelja i nasljednika dani su pseudokodom u tablici 1.2, ali su detaljnije opisani za naš problem u tablici 1.7. Roditelje biramo na način da iz populacije nasumično odaberemo 5 jedinki od kojih uzmemo 2 najbolje. Odabir nasljednika pak gleda koje su jedinke najlošije kako bi ih eliminirao iz populacije, jer želi napraviti mjesta za 2 nove (bolje) jedinke. Dakle, strategija je ta da gledamo cijelu populaciju zajedno s potomcima, rangiramo je prema podobnosti, te 2 najlošije jedinke izbacimo.

[11] Inkrementalni- onaj koji se povećava, npr. x++ je pokrata za inkrement x=x+1 u programskom smislu

1.	Nasumično odaberi poziciju, točku križanja, $i \in \{1, ..., 7\}$
2.	Podijeli oba roditelja na dva segmenta s obzirom na tu poziciju
3.	Kopiraj 1. segment 1. roditelja na 1. dijete i 1. segment 2. roditelja na 2. dijete
4.	Skeniraj 2. roditelja s lijeva na desno i popuni 2. segment 1. djeteta vrijednostima iz 2. roditelja i preskoči one koje već sadrži
5.	Napravi isto za 1. roditelja i 2. dijete

Tablica 1.6: Tzv. 'cut-and-crossfill' rekombinacija

Još nam samo preostaje da popunimo inicijalnu populaciju s nasumičnim permutacijama i odredimo uvjet zaustavljanja. Dakle, budući da znamo da točno rješenje postoji (jer znamo optimum funkcije cilja), uvjet zaustavljanja je pronalazak točnog rješenja. Postavit ćemo i dodatni uvjet: u slučaju da se izvrši 10.000 izvrednjavanja podobnosti (prije nego dođe do točnog rješenja), algoritam će se zaustaviti.

Pretpostavimo još dodatno da operaciju rekombinacije uvijek apliciramo, a mutaciju primjenjujemo samo na 80% potomstva. Opisani evolucijski algoritam dan je u tablici 1.7.

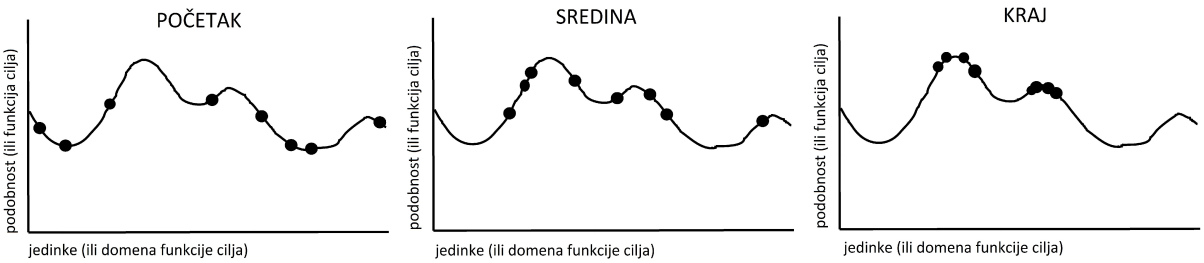
Reprezentacija	Permutacija
Rekombinacija	'Cut-and-crossfill' križanje
Mogućnost rekombinacije	100%
Mutacija	Zamjena
Mogućnost mutacije	80%
Odabir roditelja	Najbolja 2 od nasumično odabranih 5
Odabir nasljednika	Rangira populaciju i potomke te 2 najgora rješenja izbaci
Veličina populacije	100
Broj potomaka	2
Inicijalizacija	Nasumična
Uvjet zaustavljanja	Rješenje ili 10.000 izvrednjavanja podobnosti

Tablica 1.7: Opis evolucijskog algoritma za problem osam kraljica

1.6 Djelovanje evolucijskog algoritma

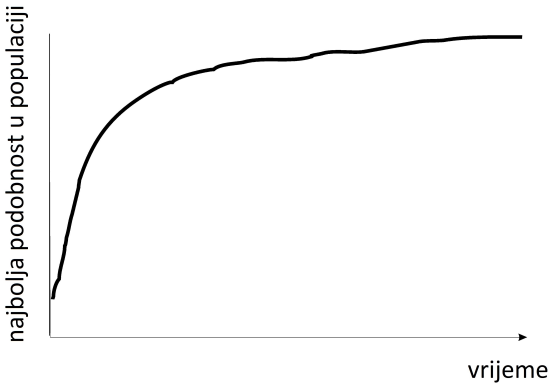
Za ilustraciju kako evolucijski algoritam uobičajeno radi, pretpostavit ćemo da imamo jednodimenzionalnu funkciju cilja koju želimo maksimizirati.

Slika 1.1 prikazuje tri faze evolucijskog pretraživanja: distribuiranost jedinki na početku, u sredini i na kraju evolucije. U prvom stadiju, odmah nakon inicijalizacije, jedinke su na-



Slika 1.1: Kretanje podobnosti populacije kroz evoluciju

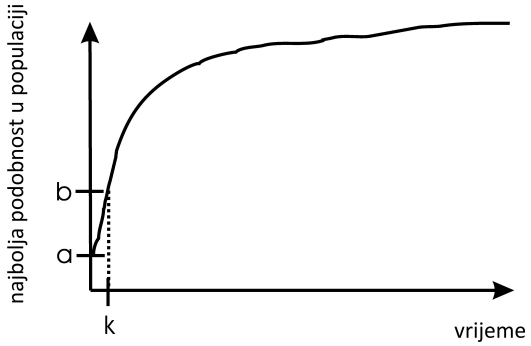
sumično rasprostranjene po cijelom prostoru pretraživanja (slika 1.1, POČETAK). Samo nekoliko generacija kasnije distribucija se mijenja: operatori selekcije i varijacije uzrokovali su porast podobnosti (slika 1.1, SREDINA). Na kraju, ako je uvjet zaustavljanja dobro postavljen, cijela je populacija koncentrirana oko više vrhova, od kojih su neki suboptimalni (slika 1.1, KRAJ). Nekada se čak može dogoditi da se cijela populacija zadrži na nekom lokalnom optimumu i nikad ne dostigne globalni.



Slika 1.2: Rast najbolje vrijednosti podobnosti kroz vrijeme

Istraživanje (eng. exploration) je pojam za stvaranje novih jedinki na još neprovjerenom području prostora pretraživanja. Iskorištavanje (eng. exploitation) pak označava koncentraciju pretraživanja u blizini već poznatih dobrih rješenja. Bitno je održavati ravnotežu između ova dva postupka jer previše istraživanja dovodi do neefikasnosti, dok previše iskorištavanja vodi do tzv. prerane konvergencije (eng. premature convergence), odnosno prebrzo se gubi raznovrsnost i populacija ostaje zarobljena u lokalnom optimumu.

Evolucijski algoritmi imaju još jednu odliku. Između ostalog, oni spadaju u tzv. *anytime* algoritme. To su algoritmi koji će dati rješenje, pa makar i suboptimalno, iako ih se prekine prije kraja izvršavanja [27]. Upravo ta značajka vidljiva je na slici 1.2 koja prikazuje rast najbolje vrijednosti podobnosti kroz vrijeme. Krivulja rapidno raste u prvih par generacija, a sve ostalo vrijeme stagnira.



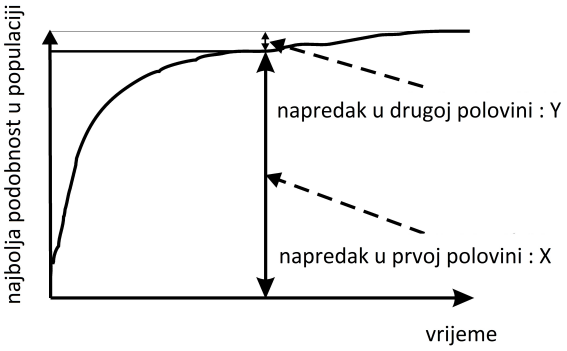
Slika 1.3: Populacija koja kreće od *a* razine podobnosti dostiže *b* razinu nakon *k* vremena

Baš zbog tog *anytime* svojstva, postavlja se pitanje isplati li se ulagati više truda u inicijalizaciju populacije s boljom podobnošću, kad će ionako kroz par generacija (broj *k* na slici 1.3) dostići željenu podobnost.

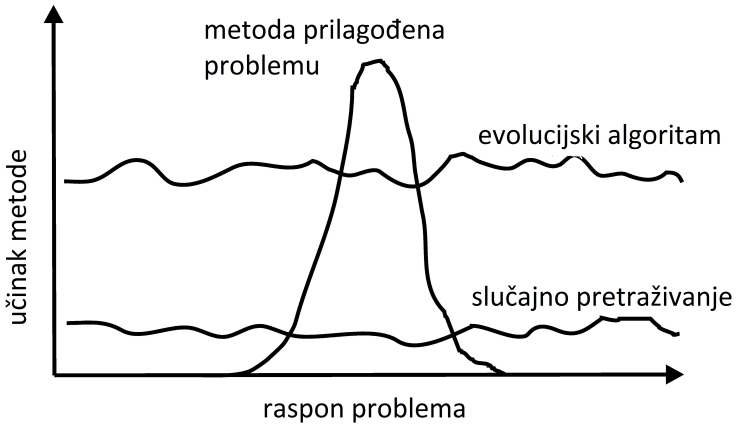
Navedena značajka utječe i na uvjet zaustavljanja evolucijskog algoritma. Na slici 1.4 podijelili smo vrijeme trajanja evolucije na dva jednako dugačka dijela X i Y. Kao što se i vidi, porast u prvoj polovici (X) je značajno veći nego u drugoj polovici (Y) što nam ukazuje na to da duže izvođenje algoritma ne daje nužno bolji rezultat, tj. možemo pretpostaviti da nakon određenog vremena (odnosno, nakon određene vrijednosti podobnosti) novim iteracijama nećemo dobiti bitno bolje rješenje.

Osvrt na rad evolucijskog algoritma završit ćemo gledanjem njegove izvedbe iz globalne perspektive, tj. gledat ćemo koliko je učinkovit za veći raspon problema.

Slika 1.5 prikazuje Goldbergovu [6] viziju učinka evolucijskog algoritma nastalu 1980-ih godina. Očito, metode osmišljene za određeni problem bolje funkcioniraju od evolucijskog algoritma, ali one nemaju nikakvog učinka za druge vrste problema. Graf pokazuje da se evolucijski algoritam bolje nosi s rješavanjem više različitih problema u odnosu na neko drugo slučajno pretraživanje. Što se tiče teorijskog razmatranja, tzv. *No Free Lunch* teorem, kojim ćemo se više pozabaviti u poglavlju 2, tvrdi da niti jedan evolucijski algoritam ne može nadmašiti slučajnu šetnju kada uzme u obzir 'sve' probleme [32]. Stoga je



Slika 1.4: Duži rad algoritma ne daje nužno bolje rješenje



Slika 1.5: Goldbergov [6] prikaz o učinku evolucijskog algoritma iz 1980-ih

generalno netočno da je evolucijski algoritam uvijek učinkovitiji od slučajnog pretraživanja kako pokazuje graf. Suvremeni pristup potvrđuje mogućnost kombiniranja evolucijskog i slučajnog pretraživanja u svrhu stvaranja novog hibridnog algoritma (čime ćemo se baviti u sljedećem poglavlju).

Poglavlje 2

Memetički algoritmi

U ovom poglavlju bavit ćemo se sustavima koji su bazirani na evolucijskom programiranju, ali imaju određenu nadogradnju. Oni mogu biti dio nekog većeg sustava, ili pak neke metode i strukture podataka mogu biti dio njih. Ovakva vrsta algoritama je vrlo uspješna u praksi i čini istraživačko područje koje rapidno raste te ima veliki potencijal. Algoritmi koji su predmet proučavanja tog područja nazivaju se memetički algoritmi. U nastavku ćemo obrazložiti zašto nam uopće trebaju takvi algoritmi i kako smo došli do njih; opisat ćemo brojne mogućnosti kombiniranja evolucijskih algoritama s drugim tehnikama i dati neke smjernice za uspješno osmišljavanje hibridnih algoritama.

2.1 Motivacija za hibridizaciju evolucijskih algoritama

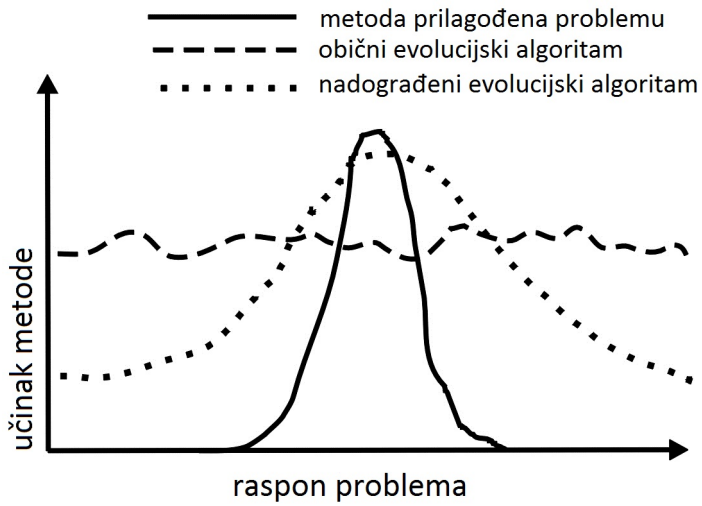
Postoji više faktora koji pobuđuju potrebu za hibridizacijom evolucijskih algoritama s drugim tehnikama, a mi ćemo spomenuti najistaknutije od njih. Većina složenih problema može se razložiti na manje dijelove koje znamo riješiti. U takvim slučajevima, ima smisla kombinirati najpogodnije metode za različite potprobleme u svrhu rješavanja glavnog problema.

Međutim, uspješan i efikasan način za generalno rješavanje problema ne postoji. U prilog ovoj činjenici idu empirijski dokazi i neki teorijski rezultati, kao npr. *No Free Lunch* teorem (NFL)[1]. Dakle, iz perspektive evolucijskog programiranja, NFL implicira da učinak evolucijskog algoritma nije onakav kakvim ga se predstavljalo 1980-ih na slici 1.5 u odjeljku 1.6.

Stoga, na slici 2.1 vidimo alternativni prikaz učinkovitosti. Graf uzima u obzir mogućnost dobivanja hibrida kombiniranjem evolucijskog algoritma s metodom specificiranom za određeni problem. Ovisno o tome koliko hibridni algoritam u sebi sadrži informacija o

[1]NFL tvrdi da su svi stohastički algoritmi isti, odnosno da je njihov učinak na sve diskretne probleme jednak. Više o samom teoremu može se naći u knjizi [32]

određenom problemu, krivulja učinkovitosti će se postepeno mijenjati od ravne vodoravne (obični evolucijski algoritam) do krivulje s vrhom (metoda prilagođena problemu).



Slika 2.1: Michalewiczov [16] prikaz o učinku evolucijskog algoritma iz 1990-ih

U praksi evolucijske algoritme primjenjujemo češće na probleme koji nisu opsežno istraženi i za koje nema dovoljno korisničkog iskustva. Pokazalo se da u je takvim slučajevima kombinacija evolucijskog algoritma i metode prilagođene problemu, tj. hibridni evolucijski algoritam, puno uspješniji nego njegovi 'roditelji' zasebno. Uočimo da je u tom kontekstu slika 2.1 pogrešna jer ne ukazuje na taj efekt.

Evolucijski algoritmi su dobri u brzom pronalaženju povoljnih područja za traženje rješenja (istraživanje), ali su manje dobri u završnoj fazi gdje bi trebali profiniti rješenje (iskorištavanje). Stoga, dodavanjem poboljšanog mehanizma za lokalno pretraživanje u evolucijski ciklus, dobili bismo efikasnije pretraživanje u blizini dobrih rješenja. Dakle, na taj bismo način *nadogradili* evolucijski algoritam i on bi postao bolji.

Konačno, motivacija koja se često koristi u ovom polju je Dawkinsova ideja o memima [3]. Memi su jedinice prijenosa kulture analogno kao što su geni jedinice biološkog prijenosa. Za umnožavanje (repliciranje) se odabiru oni najpopularniji, zatim se kopiraju i prenose putem međuljudske komunikacije.

"*Primjeri mema su melodije, ideje, slogani, moda u odijevanju, načini izrade lončarije ili zidanja lukova. Kao što se geni u zalihi gena šire prijelazom, putem spermija i jaja, iz jednog tijela u drugo, tako se i memi u zalihi mema šire prijelazom iz mozga u mozak pomoću procesa koji se, u najopćenitijem smislu, može nazvati oponašanje. Čuje li neki znanstvenik neku dobru zamisao ili pročita nešto o njoj, prenijet će je svojim kolegama i studentima. Spominjat će je u svojim člancima i predavanjima. Bude li zamisao*

prihvaćena, možemo reći da se ona proširila, šireći se od mozga do mozga." [4, str. 220]

Sa stajališta proučavanja prilagodljivih sustava i tehnika optimizacije, memi služe za transformaciju kandidata za rješenje koji nam je od izravnog interesa. Dakle, možemo razmotriti ideju da dodamo fazu učenja u evolucijski ciklus koja bi predstavljala interakciju između mema i gena, pri čemu bi reprezentacija problema (genotip) postala 'plastična', a mehanizam učenja (mem) bi predstavljao razvojni proces.

Postoje brojni razlozi zašto je hibridizacija evolucijskih algoritama s drugim tehnikama od interesa i istraživačima i krajnjim korisnicima. Upotreba drugih tehnika i znanja u poboljšanju evolucijskih algoritama dobila je kroz razne znanstvene radove mnoga imena: hibridni genetički algoritmi, Baldwinovski evolucijski algoritmi, Lamarkovski evolucijski algoritmi, genetički algoritmi za lokalno pretraživanje itd. Termin memetički algoritam prvi je upotrijebio Moscato [17] kako bi obuhvatio sve tehnike pod koje spadaju evolucijske metode poboljšane dodavanjem informacija o određenom problemu ili implementiranjem jedne ili više faza lokalnog pretraživanja.

2.2 Kratak uvod u lokalno pretraživanje

U najopćenitijim crtama, lokalno pretraživanje je iterativni proces koji ispituje skup točaka oko trenutnog rješenja, i ako nađe bolje rješenje u tom okolnom skupu, onda ga postavi za novo trenutno rješenje. U ovom odjeljku dajemo kratak uvod u lokalno pretraživanje iz perspektive memetičkih algoritama. Algoritam za lokalno pretraživanje možemo ilustrirati pseudokodom u tablici 2.1. Tri su glavne komponente koje utječu na rad ovog algoritma.

Prvo biramo pivotno pravilo koje može imati strmi uspon ili pohlepni uspon (također poznat kao prvi uspon). Unutarnja petlja se kod strmog uspona prekida nakon što ispitamo cijelo susjedstvo $n(i)$, tj. $brojač = |n(i)|$, dok se kod pohlepnog uspona zaustavlja čim nađemo bolje rješenje, tj. $((brojač = |n(i)|)$ ili $(najbolji \neq i))$. Nekad je nužno u praksi uzeti manji uzorak za ispitivanje $(N << |n(i)|)$ u slučaju da je susjedstvo preveliko.

Druga komponenta je dubina lokalnog pretraživanja, tj. uvjet zaustavljanja vanjske petlje. Ta dubina leži u rasponu između primjene samo jednog koraka $(br.iteracija = 1)$, do izvođenja onoliko koraka koliko je potrebno da dođemo do lokalne optimalnosti $((brojač = |n(i)|)$ i $(najbolji = i))$. Velika pozornost dana je proučavanju učinka mijenjanja ovog parametra unutar memetičkih algoritama i pokazalo se da utječe na izvedbu algoritma lokalnog pretraživanja, kako u pogledu vremena izvršavanja tako i u kvaliteti nađenog rješenja.

Treći i najvažniji faktor koji utječe na ponašanje lokalnog pretraživanja je izbor funkcije koja generira susjedstvo. U praksi se takva funkcija $n(i)$ najčešće definira kao operator pomaka. Ekvivalentno tome bilo bi definiranje grafa $G = (V, E)$, gdje je V skup vrhova, a E skup bridova koji bi zapravo predstavljao operator pomaka na način da vrijedi: $(i, j) \in E \iff j \in n(i)$. Merz i Freisleben [14] pokazali su da odabir operatora pomaka može imati dramatičan učinak na efikasnost lokalnog pretraživanja, a samim time i na rezultirajući

```
START
/* imamo početno rješenje i i funkciju susjedstva n
najbolji = i;
br.iteracija = 0;
DOK (uvjet dubine nije zadovoljen) PONAVLJAJ {
   brojač = 0;
   DOK (pivotno pravilo nije zadovoljeno) PONAVLJAJ {
      generiraj sljedećeg susjeda j ∈ n(i);
      brojač = brojač + 1;
      AKO (f(j) je bolji od f(najbolji)) ONDA {
         najbolji = j;
      }
   }
   i = najbolji;
   br.iteracija = br.iteracija + 1;
}
KRAJ
```

Tablica 2.1: Pseudokod algoritma za lokalno pretraživanje

memetički algoritam.

2.2.1 Lamarkizam i Baldwinov efekt

Okosnica algoritma kojeg smo upravo opisali bazira se na pretpostavci da trenutno rješenje uvijek zamijeni bolji susjed ako postoji. U kontekstu memetičkog algoritma možemo zamišljati da se faza lokalnog pretraživanja pojavljuje kao poboljšanje ili razvojna faza učenja unutar evolucijskog ciklusa. Pritom se postavlja pitanje (uzimajući u obzir biološku stranu) trebaju li se promjene na određenoj jedinki (*stečene osobine*) zadržati u genotipu ili rezultat bolje podobnosti treba biti dodijenjen originalnom članu (predlokalno pretraživanje) populacije.

Mogu li se stečene osobine prenijeti na potomstvo ili ne, bilo je glavno pitanje u 19. stoljeću. Lamarck[2] je tvrdio da je to moguće i po njemu je čak nazvana ta ideja- *lamarkizam*. Suprotno tome, Baldwinov efekt[3] [1] kaže da ako neka jedinka stekne prilagodljivu osobinu, ona će je prenijeti na potomstvo i s vremenom će cijela vrsta usvojiti tu osobinu.

[2]Jean-Baptiste de Monet de Lamarck, francuski prirodoslovac (Bazentin-le-Petit,1.8.1744 – Pariz,18.12.1829) [34]

[3]James Mark Baldwin, američki filozof i psiholog (Columbia, Južna Carolina,12.1.1861 – Pariz,8.11.1934)

Moderne teorije o genetici snažno podupiru ovo drugo gledište jer znamo koliko je mapiranje od DNA do proteina složeno i nelinearno, a kamoli onda razvojni proces kojim se stvara zreli fenotip. U tom pogledu, zaista je teško povjerovati da cijeli obrnuti proces može funkcionirati, odnosno da se stečene osobine u fenotipu mogu prenijeti na genotip.

Srećom, nas zanima okruženje kompjuterskih algoritama pa nismo ograničeni ovim biološkim pretpostavkama. Dakle, u praksi su obje sheme moguće i lako se implementiraju u memetičke algoritme. Ako rezultat faze lokalnog pretraživanja zamjenjuje jedinku u populaciji, onda takav memetički algoritam smatramo Lamarkovskim. S druge strane, Baldwinovskim memetičkim algoritmom smatramo slučaj kada uvijek imamo originalnu jedinku kojoj samo mijenjamo podobnost čiju vrijednost dobivamo kao ishod lokalnog pretraživanja. U najnovijim radovima najčešće se koristi čisti Lamarkovski pristup ili probabilistička kombinacija dvaju pristupa na način da je poboljšana podobnost uvijek bitna i da se originalna jedinka zamijeni boljom s određenom vjerojatnošću.

2.3 Struktura memetičkog algoritma

Postoji više načina za spajanje evolucijskih algoritama s drugim operatorima i/ili informacijama specificiranima za neki problem kao što je i prikazano na slici 2.2.

2.3.1 Heuristička (istraživačka) ili pametna inicijalizacija

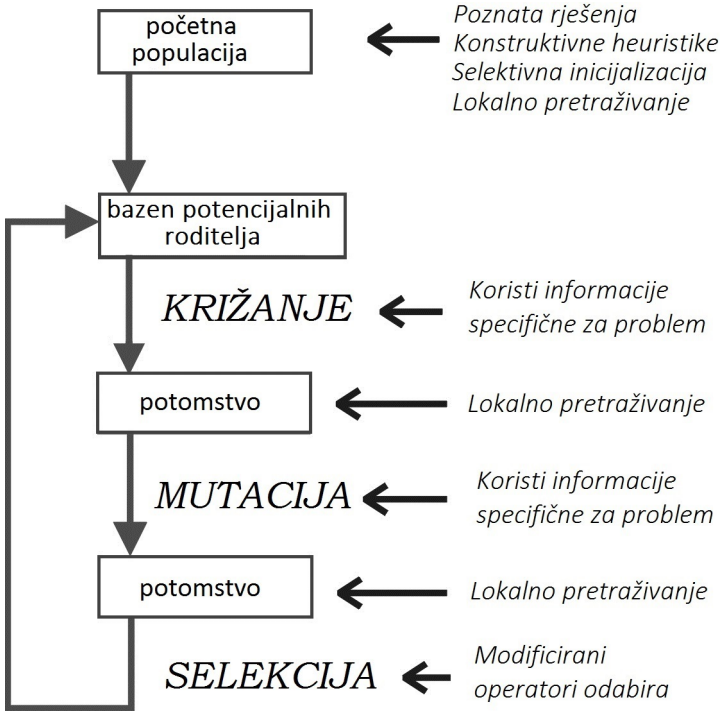
Najočitije mjesto gdje bismo mogli umetnuti postojeće znanje o strukturi problema ili informacije o potencijalnom rješenju je faza inicijalizacije. U odjeljku 1.6 smo već raspravljali o temi inicijalizacije i dali smo razloge zašto nema smisla ulagati previše truda u nju (u prilog tome ide i slika 1.3). Međutim, pokretanje evolucijskog algoritma s već postojećim rješenjima ima zanimljive prednosti:

1. Ako koristimo postojeća rješenja možemo izbjeći tzv. *ponovni izum kotača*[4]. Sprječavanje traćenja računalnih napora može povećati učinkovitost algoritma (npr. ubrzati ga)

2. Ciljano (nenasumično) odabiranje početne populacije može usmjeriti potragu u dijelove prostora pretraživanja koji sadrže dobra rješenja. Usmjeravanje pretraživanja može rezultirati povećanom učinkovitošću (npr. kvalitetnije krajnje rješenje)

3. Neka određena ukupna količina računalnog napora podijeljena između heurističke inicijalizacije i evolucijskog pretraživanja sigurno daje bolje rezultate nego da smo sav taj isti napor uložili samo u evolucijsko pretraživanje ili samo na heuristiku

[4]Ponovni izum kotača (eng. reinventing the wheel)- odnosi se na dupliciranje osnovne metode koja je već napravljena ili optimizirana [30]



Slika 2.2: Mogući načini za implementiranje drugih operatora ili specifičnih znanja u evolucijski ciklus

U sljedećim točkama navodimo neke od metoda kojima možemo nadograditi jednostavnu nasumičnu funkciju inicijalizacije:

- Možemo uzeti skup već poznatih rješenja koje smo dobili pomoću nekih drugih tehnika (od jednostavne metode pokušaja i pogreške do visoko specijaliziranih istraživačkih konstrukcija) i njima '*posaditi*' populaciju. Ovakav način koristi se kod problema trgovačkog putnika (pronalazak najbližih susjeda), za probleme planiranja (najefikasnije stvaranje rasporeda) i sl.

- *Selektivna inicijalizacija* predstavlja postupak u kojem proizvedemo velik broj nasumičnih rješenja (jedinki) i onda od njih izabiremo početnu populaciju. To uključuje odabir skupa jedinki koji se temelji ne samo na podobnosti već i na različitosti, kako bi se obuhvatio veći prostor pretraživanja.

- Možemo izvoditi lokalno pretraživanje tako da počnemo od svakog člana početne populacije i na taj način dobivamo inicijalnu populaciju koja se sastoji od skupa točaka koje su lokalno optimalne s obzirom na neki operator pomaka.

- Možemo iskoristiti neke od nabrojanih metoda i pomoću njih dobiti dobra rješenja, zatim ta rješenja možemo klonirati i mutirati kako bismo dobili više jedinki u blizini početne točke.

Sve ove metode su već isprobane i za određene probleme su polučile dobre rezultate. Međutim, važno je uzeti u obzir da evolucijskom algoritmu treba pružiti dovoljno raznolikosti da bi se cijeli proces dobro odvio. U [24] su Surry i Radcliffe ispitivali kakav učinak na genetički algoritam ima ako u početnu populaciju dodamo određen omjer već poznatih dobrih rješenja. Uočili su da malo dodavanje izvedenih rješenja pomaže u genetičkom pretraživanju, a povećavanjem tog udjela poboljšava se *prosječna* učinkovitost. Ipak, pokazalo se da se *najbolja* učinkovitost javlja iz nasumičnije odabrane početne populacije. Drugim riječima, povećavanjem omjera izvedenih rješenja u početnoj populaciji povećava se srednja vrijednost učinkovitosti izvedbe algoritma, ali smanjuje se varijanca. To zapravo znači da nije bilo puno jako loših izvođenja, ali zato je bilo i manje onih izrazito dobrih, što je za neke tipove problema nepoželjno svojstvo.

2.3.2 Hibridizacija unutar operatora varijacije: Pametno križanje i mutacija

Mnogi stručnjaci su predlagali tzv. pametne operatore varijacije koji bi sadržavali znanja i informacije o određenom problemu. Primjer toga možemo vidjeti u [23] gdje je objašnjeno kako se kodiraju geni za mikroprocesorske naredbe koje se onda prirodno grupiraju u skupove na temelju sličnosti. Operator mutacije je u tom slučaju sadržavao u sebi informaciju o grupiranju, pa su se mutacije više odvijale među naredbama iz istog skupa, nego iz različitih skupova.

Nešto drugačiji primjer od ovoga je iz [25] i govori o računalnom predviđanju strukture bjelančevina. U tu svrhu koristi se operator križanja u jednoj točki i u njemu je implementirano znanje i informacije o danom problemu. Ono što je uočeno jest da su nasljedna svojstva nastala križanjem zapravo nabori ili fragmenti trodimenzionalne strukture. Svojstvo ovog problema je da se strukture bjelančevina tijekom presavijanja mogu slobodno rotirati oko peptidnih veza. Modificirani operator je dobro iskoristio ovu informaciju i to na način da je testirao sve moguće orijentacije dvaju fragmenata kako bi našao one najpogodnije. Ako nije pronađena niti jedna izvediva konformacija[5], onda se odabire druga točka križanja i proces se ponavlja. Ovo možemo smatrati jednostavnim primjerom uključivanja faze lokalnog pretraživanja u operator rekombinacije.

Za kraj ovog odjeljka odabrali smo najkompleksniji primjer modifikacije operatora-Merzov i Freislebenov *operator križanja koji čuva udaljenosti* u svrhu rješavanja problema

[5]Konformacije su različiti oblici molekula koji su posljedica rotacije dijela molekule oko jednostruke veze. [21]

trgovačkog putnika. Ovaj operator čine dva načela: dobro iskorištavanje informacija usko vezanih uz sami problem te u isto vrijeme očuvanje raznolikosti unutar populacije kako ne bi došlo do prerane konvergencije[6]. Raznovrsnost ćemo sačuvati ako osiguramo da potomstvo naslijedi sve osobine koje su zajedničke za oba roditelja, i nijednu osobinu koju ima samo jedan od njih. To omogućava da potomci budu jednako udaljeni od svakog roditelja kao i jedan od drugoga. Pametni dio ovog operatora je korištenje heuristike za pronalaženje najbližeg susjeda kako bi se pospajali dijelovi puta nasljeđeni od roditelja, i na taj se način eksplicitno iskorištavaju informacije vezane uz sami problem. Lako se vidi kako je ova shema primjenjiva na razne probleme: djelomična rješenja koja smo dobili nasljeđivanjem osobina od oba roditelja samo treba upotpuniti pomoću odgovarajućih postupaka.

2.3.3 Djelovanje lokalnog pretraživanja na rezultate operatora varijacije

Najčešća upotreba hibridizacije unutar evolucijskih algoritama, a ujedno i ona koja najbolje odgovara Dawkinsovu konceptu mema, je primjena jedne ili više faza poboljšanja na neku jedinku tijekom ciklusa evolucijskog algoritma, tj. djelovanje lokalnog pretraživanja na sva rješenja koja su nastala mutacijom ili rekombinacijom. Iz slike 2.2 vidimo da se to može odviti u različitim dijelovima ciklusa, ali najčešća implementacija dana je pseudokodom u tablici 2.2.

Jedan rezultat, i to nedavno dobiven, nam je od posebnog interesa, a to je Krasnogorov formalni dokaz koji kaže da ako želimo smanjiti najlošija izvođenja, onda je nužno odabrati metodu lokalnog pretraživanja čiji operator pomaka ne smije biti isti kao kod operatora mutacije i rekombinacije [7]. Ovakve preinake najbolje se postižu agresivnom stopom mutacije ili upotrebom operatora varijacije koji imaju različite strukture susjedstva.

2.3.4 Hibridizacija tijekom mapiranja genotipa u fenotip

Hibridizacija memetičkih algoritama s drugim tehnikama se često događa tijekom mapiranja genotipa u fenotip (prije procjene). Dobar primjer za to je upotreba specifičnih informacija za određeni problem s nekom metodom za dešifriranje ili s funkcijom oporavka. Pristup u kojem se evolucijski algoritam koristi kako bi pružio ulazne podatke i time kontrolirao primjenu neke druge heuristike, često se koristi za postizanje velikog učinka u npr. rješavanju problema planiranja i stvaranja rasporeda.

Kao što se lako vidi postoji poveznica između svih pristupa, a to je korištenje postojećih metoda i informacija o domeni gdje god je to moguće. Uloga evolucijskog algoritma je omogućavanje primjene manje pristrane heuristike ili dekompozicije problema kako bi se

[6]Prerana konvergencija objašnjena je u odjeljku 1.6

START
INICIJALIZIRAJ populaciju;
PROCIJENI svakog kandidata;
DOK (*uvjet zaustavljanja* nije zadovoljen) PONAVLJAJ {
ODABERI roditelje;
REKOMBINIRAJ roditelje za stvaranje potomstva;
MUTIRAJ potomstvo;
PROCIJENI potomstvo;
[nije obavezno] IZABERI koju metodu za lokalno pretraživanje želiš primijeniti;
POBOLJŠAJ potomstvo pomoću lokalnog pretraživanja;
[nije obavezno] Dodijeli ocjenu metodama lokalnog pretraživanja (s obzirom na to kolika su poboljšanja izazvale u podobnosti);
ODABERI jedinke za novu generaciju;
NAGRADI trenutne uspješne metode za lokalno pretraživanje (povećanjem vjerojatnosti da se ubuduće koriste);
}
KRAJ

Tablica 2.2: Pseudokod za jednostavni memetički algoritam

dopustilo korištenje sofisticiranih, ali loše skaliranih metoda[7] kada ukupna veličina problema isključi njihovu upotrebu.

2.4 Prilagodljivi memetički algoritmi

Najvažniji faktor u osmišljavanju memetičkog algoritma je način na koji generiramo skup susjednih točaka (unutar kojeg tražimo bolje rješenje). U tu svrhu dostupan je velik dio empirijske i teorijske analize za određivanje težine problema. Već smo ranije spomenuli Merzovo i Freislebenovo otkriće da odabir operatora pomaka ima velik utjecaj na lokalno pretraživanje, pa i na rezultirajući memetički algoritam (odjeljak 2.2). Također, ponovno spominjemo Krasnogorov rezultat koji kaže da je za smanjenje složenosti najgoreg vremena izvođenja algoritma poželjno definirati operator pomaka unutar lokalnog pretraživanja takvim da bude drugačiji od operatora mutacije i rekombinacije (pododjeljak 2.3.3).

Dakle, kada osmišljavamo memetički algoritam, važno je pažljivo razmotriti koji operator pomaka ćemo koristiti. U nekim slučajevima za izbor strukture susjedstva mogu

[7]Metoda je loše skalirana ako se složenost algoritma drastično povećava povećanjem broja ulaznih podataka.

pomoći informacije vezane za domenu problema.

Jedan jednostavan način za svladavanje ovakvih problema je upotreba *više (eng. multiple)* operatora lokalnog pretraživanja u tandemu. Krasnogor i Smith [9] uveli su prvi put ideju tzv. '*multimemetičkih*' algoritama, gdje je evolucijski algoritam bio povezan ne s jednom, već s nekoliko metoda za lokalno pretraživanje; zajedno s tim metodama bio je implementiran mehanizam koji je ovisno o stadiju pretraživanja odlučivao koja je od metoda najkorisnija, i nju odabirao. Taj mehanizam odnosio se na *samoprilagodljivost* u smislu da svaki kandidat za rješenje nosi gen koji pokazuje koju metodu lokalnog pretraživanja treba koristiti. To svojstvo je naslijeđeno od roditelja i podložno je mutaciji. Primjer toga dan je u [8] gdje se za problem predviđanja strukture bjelančevina koristio cijeli raspon operatora pomaka od kojih je svaki bio prilagođen za određenu fazu procesa presavijanja (npr. lokalno proširivanje, rotiranje, reflektiranje, itd.).

Ong i Keane razvijali su slične ideje, ali koristili različite mehanizme odabira, i to su nazivali '*meta-lamarkovsko učenje*'. Nije trebalo dugo da se uoči sličnost između ta dva pristupa i u [19] je predstavljena izvrsna recenzija rada na polju kojeg su nazvali '*prilagodljivi memetički algoritmi*'. Taj termin obuhvaća multimemetičke algoritme, razvojne, meta-lamarkovske i samogenerirajuće memetičke algoritme te meta-heuristike[8].

U suštini svi ovi pristupi čine skup operatora lokalnog pretraživanja koji je na raspolaganju algoritmu, u smislu da u svakom trenutku odluke algoritam odabere koji će operator koristiti. Algoritme svrstavamo u kategorije ovisno o načinu na koje donose odluke. Ako koriste fiksnu strategiju onda ih svrstavamo u 'statične'. S druge strane, ako koriste povratne informacije o tome koji su operatori osigurali najbolji napredak do sad, onda ih kategoriziramo kao 'prilagodljive' algoritme. S obzirom na prirodu informacija koje koriste, prilagodljive algoritme dijelimo na: vanjske, globalne (cijela populacija) i lokalne (određeni dio prostora pretraživanja). Nadalje, uočeno je da operatori lokalnog pretraživanja mogu biti povezani s kandidatima za rješenja i u tom slučaju govorimo o 'samoprilagodljivim' algoritmima. Puno istraživanja usmjereno je na to kako najbolje *odrediti* koji je trenutno koristan mehanizam za lokalno pretraživanje, treba li *nagraditi* korisne meme povećavajući vjerojatnost da ih se opet upotrijebi, i kako *prilagoditi* definiciju mehanizma lokalnog pretraživanja.

S obzirom na navedene pojmove, Meuth i ostali u [15] uvode podjelu na:

- Memetičke algoritme prve generacije (1G)- u [15] su definirani kao "Globalno pretraživanje upareno s lokalnim pretraživanjem"
- Memetičke algoritme druge generacije (2G)- "Globalno pretraživanje s više lokalnih optimizatora. Memetičke informacije (odabir optimizatora) se prenose na potomstvo (Lamarkovska evolucija)."

[8]Meta-heuristika je metoda (najčešće implementirana tehnikama strojnog učenja) koja odabire, koristi i kombinira jednostavnije heuristike za rješavanje nekog problema računalnog pretraživanja.

- Memetičke algoritme treće generacije (3G)- ”Globalno pretraživanje s više lokalnih optimizatora. Memetičke informacije (odabir lokalnog optimizatora) se prenose na potomstvo (Lamarkovska evolucija). Usvojeno je mapiranje između evolucijske putanje i odabira lokalnog optimizatora.”

Napomenuli su da su u vrijeme pisanja [15] jedino razvojni i samogenerirajući memetički algoritmi spadali u 3G klasu, i predložili su još jednu kategoriju (ali nisu je implementirali): memetičke algoritme četvrte generacije (4G) koja je po njima predstavljala ”Korištenje memorije i mehanizama prepoznavanja, generalizacije, optimizacije.”. Nedvojbeno je da razvojni memetički algoritmi koji sadržavaju meme zasnovane na uzorcima (eng. pattern-based memes) spadaju u tu zadnju opisanu kategoriju.

2.5 Kreiranje memetičkih algoritama

Dosada smo raspravljali o razlozima za uvođenje informacija vezanih za određeni problem i upotrebu heuristika u evolucijskim algoritmima, te objasnili moguće načine kako bismo to napravili. Međutim, kao i uvijek, moramo prihvatiti činjenicu da memetički algoritmi nisu nekakvo ’magično rješenje’ za probleme optimizacije i moramo, kao i sa svim drugim tehnikama, uložiti puno truda i jako paziti na to kako ih implementiramo. U nastavku ovog odjeljka ukratko raspravljamo o nekim pitanjima koja su proizašla iz iskustva i teorijskog razmišljanja.

2.5.1 Očuvanje raznovrsnosti

Problem prerane konvergencije, u kojem populacija konvergira oko neke suboptimalne točke, javlja se općenito kod evolucijskih algoritama, ali se zbog učinka lokalnog pretraživanja pogoršava kada su u pitanju memetički algoritmi. Ako faza lokalnog pretraživanja traje dok god svaka točka ne bude premještena na lokalni optimum, onda to vodi do neizbježnog gubitka raznolikosti unutar populacije[9]. Navodimo neke od pristupa kojima bi se riješio ovaj problem:

- Ako se populacija inicijalizira na način da ubacimo već poznate dobre jedinke, onda to treba biti razmjerno mali udio istih
- Upotreba operatora rekombinacije koji su osmišljeni tako da čuvaju raznolikost
- Modificiranje operatora odabira tako da se spriječe duplikati

[9] Osim iznimnog slušaja kada svakog člana populacije privlači drukčiji lokalni optimum

- Modificiranje operatora odabira ili kriterija za lokalno pretraživanje tako da koriste Boltzmannovu metodu[10] kako bi se sačuvala raznolikost

2.5.2 Primjena informacija i stečenih znanja

Završna točka koju razmatramo pri kreiranju memetičkog algoritma je upotreba i recikliranje informacija i znanja koja smo stekli tijekom procesa optimizacije. U određenoj mjeri to se automatski postiže rekombinacijom, ali generalno govoreći ne postoji neki izričiti mehanizam koji to radi.

Jedna moguća hibridizacija je tzv. *tabu pretraživanje* [5] koje na eksplicitan način koristi informacije o točkama koje su već pretražene kako bi vodio optimizaciju u dobrom smjeru. U ovom algoritmu čuva se 'tabu' popis koji sadržava listu već posjećenih točaka u koje je zabranjeno vraćanje. Pokazalo se da su ovakve metode obećavajuće i za održavanje raznolikosti. Slično, možemo lako zamisliti poboljšanje Boltzmannove sheme odabira koji koristi informacije o širenju genotipova u sadašnjoj populaciji (ili čak u prošlim populacijama) pri odlučivanje hoće li prihvatiti nova rješenja ili ne.

2.6 Primjer iz primjene: Memetički algoritam za izradu rasporeda sati

Za ilustraciju načina na koje se evolucijski algoritmi mogu kombinirati s drugim tehnikama, uzet ćemo problem rasporeda ispita (eng. Examination Timetabling Problem) opisan u [2]. Izrada rasporeda je općenito NP[11]-potpuni problem[12] i od osobitog je interesa unutar akademskih zajednica. Opći oblik problema sastoji se od skupa ispita E, od kojih svaki ispit ima određen broj sjedala s_i na koje se student može smjestiti u rasporedu s obzirom na skup vremena P. U pravilu imamo na raspolaganju i matricu susjedstva C gdje c_{ij} predstavlja broj studenata koji polažu oba ispita i i j. Ako postoji izvedivo rješenje onda se radi o *problemu zadovoljavanja ograničenja* (eng. CSP- constraint satisfaction problem), ali općenito takvo rješenje ne mora postajati, pa se u tom slučaju tretira kao *problem optimizacije s ograničenjima* kroz upotrebu tzv. *kaznene funkcije* (eng. penalty function). Ta funkcija obuhvaća neka od sljedećih ograničenja:

- Ispiti mogu biti zakazani samo u učionicama s odgovarajućim kapacitetom

[10] Boltzmannova metoda je metoda analogna simuliranom prekaljivanju gdje se potezi koji nose pogoršanja događaju s ne-nul vjerojatnošću, što pomaže pri izbjegavanju lokalnog optimuma.

[11] NP je kratica za nedeterminističko polinomijalno vrijeme

[12] NP-potpuni problemi imaju svojstvo da im se već gotova rješenja mogu brzo provjeriti, ali do samog rješenja općenito nije lako doći [29].

- Ne smiju se zakazati dva ispita i i j u isto vrijeme ako je $c_{ij} > 0$
- Trebalo bi izbjegavati da studenti imaju više ispita u istom danu
- Poželjno je da studenti nemaju ispite u uzastopnim razdobljima (čak i ako je noć između njih)

Iako je problem izrade rasporeda sati i planiranja već dobro proučen i mnogi heuristički pristupi su osmišljeni, problem je u tome što se oni često ne podudaraju. Pristup koji su dokumentirali Burke i Newell u [2] je osobito zanimljiv i izrazito važan za ovo poglavlje jer ima sljedeće značajke:

- Koristi se pristup dekompozicije problema na način da heuristički raspoređivač razlaže problem na manje dijelove koje se lako daju riješiti tehnikama optimizacije
- Evolucijski algoritam je zapravo heuristička optimizacija koju koristimo
- Evolucijski algoritam sam inkorporira druge heuristike, tj. to je zapravo (po definiciji) memetički algoritam

Heuristički raspoređivač dijeli skup E na nekoliko manjih jednako velikih podskupova koji se naizmjence raspoređuju. Dakle, pri raspoređivanju elemenata n-tog podskupa, elementi $(n-1)$.-og podskupa su već raspoređeni i ne mogu se mijenjati. Skup E particioniramo pomoću metrike kojom procjenjujemo težinu raspoređivanja svakog ispita te zatim rangiramo ispite tako da prvo rasporedimo one najteže (s obzirom na metriku). Razmatramo tri različite metrike: metrika koja broji koliziju s drugim događajima, metrika koja broji koliziju s *prethodno zakazanim* događajima, metrika koja broji slobodne termine u rasporedu. Stručnjaci koji su se bavili ovim problemom spominju još i tzv. *tehnike gledanja unaprijed* (eng. look-ahead techniques) koje uzimaju dva podskupa, ali samo jedan od njih svrstaju u raspored.

Odabrana heuristika za ovaj problem je sama po sebi memetički algoritam s parametrima navedenima u tablici 2.3.

Objasnimo par točaka iz tablice. Svaki član početne populacije nastao je nasumičnim permutiranjem ispita nakon čega mu je dodijeljeno prvo slobodno vrijeme održavanja (u poretku kojeg smo dobili nakon permutacije). Algoritam za lokalno pretraživanje uvijek se primjenjuje dok god nije pronađeno lokalno optimalno rješenje, ali koristi mehanizam pohlepnog uspona tako da postoji varijabilnost u izlaznim podacima. Primjenjuje se na svako inicijalno rješenje i na svakog potomka i stoga evolucijski algoritam uvijek radi s rješenjima koja su lokalno optimalna (barem u odnosu na ovaj operator).

Stručnjaci koji su obavljali ove eksperimente uočili su da ne treba koristiti rekombinaciju. Umjesto toga je svaki potomak stvoren putem uporabe jednog od dva operatora mutacije koji su specifični za problem. 'Blagi' operator je zapravo verzija *mutacije premetanjem*

Reprezentacija	Skup povezanih popisa ispita, svaki kodiran za jedno razdoblje (unutar skupa vremena P)
Rekombinacija	Ne koristi se
Mutacija	Nasumičan odabir između 'blage' i 'jake' mutacije
Mogućnost mutacije	100%
Odabir roditelja	Eksponencijalno rangiranje
Odabir nasljednika	Najboljih 50 od 100 potomaka
Veličina populacije	50
Inicijalizacija	Generirana nasumično i na nju primijenjeno lokalno pretraživanje
Uvjet zaustavljanja	Pet generacija bez poboljšanja u najboljoj podobnosti
Posebna svojstva	Lokalno pretraživanje (koje vodi do lokalnog optimuma) primjenjuje se nakon mutacije

Tablica 2.3: Tablica koja opisuje memetički algoritam nadograđen algoritmom za višestupanjsko raspoređivanje

koja provjerava izvedivost rješenja koje proizvodi. 'Jaka' mutacija je izrazito specifična i koristi se u određenim slučajevima. Ona promatra roditelje i na temelju kaznene funkcije računa kolika je vjerojatnost da se događaji u određenom vremenu 'poremete'. No, ovaj operator također koristi i informacije od drugih rješenja jer se vjerojatnosti poremećaja mijenjaju s obzirom na najbolje rješenje u trenutnoj populaciji.

Rezultati dobiveni ovim algoritmom su impresivni što se tiče brzine izvršavanja i same kvalitete rješenja. Vrijedi naglasiti sljedeće točke koje su dovele do tog uspjeha:

- Kombinacija evolucijskog algoritma s drugim tehnikama brže daje bolje rezultate nego svaka tehnika ili algoritam zasebno
- Algoritam koristi lokalno pretraživanje tako da je inicijalna populacija u startu bolja od neke nasumično odabrane
- Primjenjuju se jači i bolji mehanizmi odabira: eksponencijalno rangiranje i za odabir roditelja i za odabir nasljednika
- Koriste se pametni operatori mutacije. Jedan koristi informacije specifične za problem kako bi spriječio stvaranje rješenja koja krše najvažnija ograničenja. Drugom je pak svrha izbjegavanje 'loših' vremena u rasporedu
- Operator 'jake' mutacije koristi informacije iz ostatka populacije kako bi uvidio koliko je vjerojatno da se neko vrijeme poremeti
- Dubina lokalnog pretraživanja je uvijek maksimalna, tj. populacija roditelja uvijek će proizaći iz skupa lokalnog optimuma

- Unatoč jačim mehanizmima odabira i prethodnoj stavci, prerani gubitak raznovrsnosti je izbjegnut zbog toga što se mutacija *uvijek* primjenjuje i zato što svi operatori pretraživanja imaju različite mehanizme kretanja
- Postoji širok spektar raznog kodiranja i algoritamskih strategija kako bi se izbjegla puna procjena rješenja i kako bi se ubrzala manipulacija djelomičnih rješenja

Poglavlje 3

Primjer u MATLAB-u

Na kraju ovog rada dajemo primjer memetičkog algoritma napravljenog u MATLAB-u.

3.1 Algoritam skakutanja žaba

Algoritam skakutanja žaba (eng. Shuffled Frog Leaping Algoritm- SFLA) je memetički algoritam koji je razvijen za rješavanje kombinatornih problema optimizacije. Algoritam se sastoji od lokalnog pretraživanja te globalne razmjene informacija i podataka. SFLA uzima virtualnu populaciju žaba koje su u interakciji i koje su podijeljene u različite *memplekse*[1]. Mem je jedinica kulturne evolucije, a virtualne žabe djeluju kao domaćini ili nositelji mema. Algoritam istodobno izvodi lokalno pretraživanje u svakom mempleksu neovisno. Kako bi se osiguralo globalno istraživanje (eng. exploration), virtualne žabe se periodički miješaju i reorganiziraju u nove memplekse. Osim toga, kako bi se pružila prilika za slučajno generiranje poboljšanih informacija, virtualne žabe su nasumično generirane i supstituirane unutar populacije.

Kôd za opisani algoritam preuzet je sa [18] i zamišljen je kao osnova koju treba modificirati ovisno o problemu koji želimo riješiti. U nastavku slijedi detaljna analiza samog algoritma i dobivenih rezultata.

3.2 Analiza algoritma

Objasnimo podrobnije naš problem. Dakle, imamo nasumično odabranu inicijalnu populaciju od 500 žaba koje obitavaju na području $[-10, 10] \times [-10, 10]$. Za funkciju podobnosti

[1] Mempleksi ili kompleksi mema su zapravo grupe mema koje nalazimo u istoj jedinki. S obzirom na univerzalni darvinizam, mempleksi nastaju zato što se memi repliciraju i kopiraju uspješnije ako su 'udruženi'. [28]

uzeli smo zbroj kvadrata koordinata, tj. ako je pozicija neke žabe (x, y), onda je njena podobnost $x^2 + y^2$. Svaka žaba implementirana je kao strukturno polje koje se sastoji od dvije varijable: pozicija i podobnost. Ono što mi želimo jest maksimizirati našu funkciju cilja $f(x, y) = x^2 + y^2$. Pseudokod algoritma dan je u tablici 3.1.

START
funkcija_cilja$(x, y) = x^2 + y^2$;
okolina = $[-10, 10] \times [-10, 10]$;
max_br_iteracija = 500;
br_mempleksa = 5;
veličina_mempleksa = 100;
veličina_populacije = br_mempleksa * veličina_mempleksa;
br_roditelja = max $\{round(0.3 * veličina\_mempleksa), 2\}$;
% gdje je $round(x) = \lfloor x + \frac{1}{2} \rfloor$, odnosno tzv. najbliže cijelo
br_potomaka = max $\{round(0.1 * br\_roditelja), 1\}$;
profinjenje = 5;
step = 2;
INICIJALIZIRAJ NASUMIČNU POPULACIJU:
ZA $i = 1 : veličina\_populacije$ {
  populacija(i).Pozicija = (1, 1) * slučajna_uniformna_distribucija_na_intervalu $[-10, 10]$;
  populacija(i).Podobnost = funkcija_cilja(populacija(i).Pozicija);
  }
Sortiraj populaciju u uzlaznom poretku s obzirom na podobnost;
GLAVNA PETLJA:
ZA $it = 1 : max\_br\_iteracija$ {
  Nasumično rasporedi žabe po mempleksima;
  U svakom mempleksu neovisno izvrši lokalno pretraživanje (**pseudokod u tablici 3.2**);
  Vrati žabe iz mempleksa u populaciju;
  Sortiraj populaciju u uzlaznom poretku s obzirom na podobnost;
}
KRAJ

Tablica 3.1: Pseudokod za algoritam skakutanja žaba

Prijeđimo sada na glavni dio programa. Naš algoritam u svakom koraku gleda cjelokupnu populaciju žaba i uvijek ih na slučajan način raspodijeli u memplekse. Zatim, u svakom od tih mempleksa nezavisno izvodi lokalno pretraživanje. Dakle, prvo odabire 30 roditelja unutar svakog mempleksa i to s obzirom na vektor vjerojatnosti odabira pojedine jedinke. Njih zatim sortira uzlazno s obzirom na podobnosti i kreće u potragu za potomcima.

n = *veličina_populacije*;
$P = 2 * (1 : n)/(n * (n + 1))$; % vektor vjerojatnosti odabira jedinke
% $(1 : n)$ je MATLAB-ovska notacija za vektor $[1, 2, ..., n]$
ZA $i = 1$: *profinjenje* {
  Sortiraj žabe unutar mempleksa uzlazno s obzirom na podobnost;
  Odaberi roditelje s obzirom na P i izvuci ih iz mempleksa (**vidi tablicu 3.3**);
  ZA $k = 1$: *br_potomaka* {
    1. Sortiraj roditelje uzlazno s obzirom na podobnost;
    2. *Korak* = *step* * (a, b). * (pozicija najboljeg - pozicija najlošijeg roditelja);
       % gdje su a i b nasumične konstante između (0, 1), a *step* je unaprijed zadani pomak
    3. Potomak = najlošiji roditelj + *korak*;
    AKO je pozicija potomka unutar zadane okoline $[-10, 10] \times [-10, 10]$ ONDA {
      Izračunaj podobnost potomka;
      AKO je podobnost potomka veća od podobnosti najlošijeg roditelja ONDA {
        Zamijeni najlošijeg roditelja s potomkom za sljedeću generaciju;
        }
      INAČE {
        AKO ovo radiš prvi put ONDA ponovno idi na 2.;
        INAČE Najlošijeg roditelja zamijeni nasumičnom jedinkom iz zadane okoline;
      }
    }
    INAČE {
      AKO ovo radiš prvi put ONDA ponovno idi na 2.;
      INAČE Najlošijeg roditelja zamijeni nasumičnom jedinkom iz zadane okoline;
    }
  }
  Vrati preživjele roditelje i dobivene potomke u mempleks;
}

Tablica 3.2: Pseudokod za odabir potomaka unutar pojedinog mempleksa

Vektor vjerojatnosti odabira računali smo na sljedeći način: razmatramo formulu za sumu prvih n prirodnih brojeva $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$. Za broj n uzeli smo veličinu mempleksa, dakle $n = 100$. Sada nije teško vidjeti da ako u vektor poredamo brojeve $[1, 2, ..., 99, 100]$ i pomnožimo taj vektor s $\frac{2}{n(n+1)}$, dobivamo da je zbroj komponenti jednak 1, odnosno dobivamo vektor vjerojatnosti kojeg ćemo nazvati P. Nadalje, odabir roditelja se vrši pomoću MATLAB-ove funkcije $y = randsample(n, k, true, w)$ gdje je n veličina mempleksa, k broj

roditelja unutar svakog mempleksa, a w vektor vjerojatnosti. Ta funkcija vraća vektor dimenzija $k \times 1$ u kojem je vjerojatnost da broj i bude odabran kao komponenta tog vektora dana s $\frac{P(i)}{\sum_{l=1}^{n} P(l)} \overset{(*)}{=} P(i)$, gdje vrijedi jednakost $(*)$ jer smo P odabrali takvim da bude vjerojatnosni vektor, odnosno $\sum_{l=1}^{n} P(l) = 1$ (detaljnije o samoj funkciji na [13]). Međutim, budući da ta funkcija može vratiti više puta isti indeks, mi je pozivamo na način opisan u tablici 3.3. Dakle, svaki put kada opisana funkcija odabere neki indeks, automatski tu komponentu u vektoru P postavljamo na nulu kako ne bi mogla ponovno biti odabrana. Na taj način sprječavamo višestruke indekse, odnosno naših će $k = 30$ roditelja svi biti različiti.

```
L = zeros(k, 1);    % gdje je k broj roditelja;
temp = P;
ZA i = 1 : br_roditelja {
   L(i) = randsample(n, 1, true, temp);    % gdje je n veličina mempleksa
   temp(L(i)) = 0;
}
```

Tablica 3.3: Pseudokod za odabir roditelja pomoću vektora vjerojatnosti

Dobiveni vektor označava indekse žaba u mempleksu koje su odabrane da budu roditelji. Budući da žabe u svakom mempleksu najprije sortiramo uzlazno s obzirom na podobnost i s obzirom na to da su vjerojatnosti u vektoru P više za veće brojeve nego za manje, zaista omogućavamo odabir roditelja koji je *vjerojatnosno* skloniji podobnijim jedinkama.

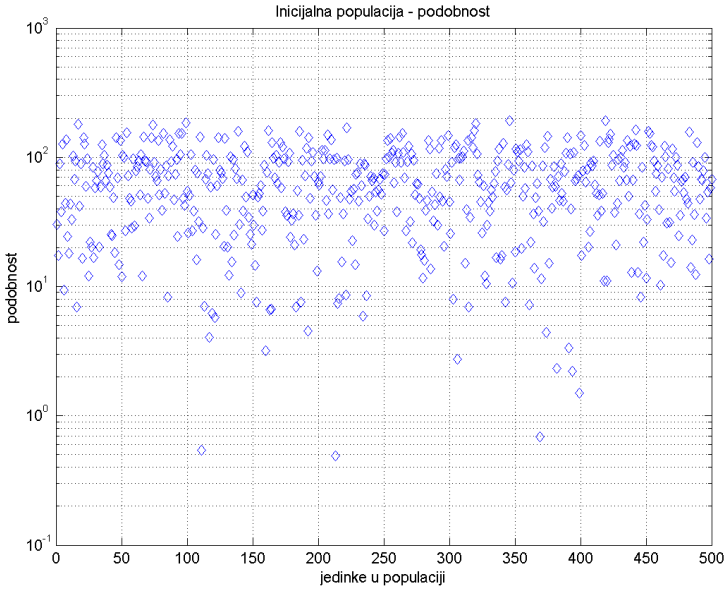
Nadalje, objasnimo što je *profinjenje* i zašto ga uopće koristimo. Naime, u svakoj iteraciji na nasumičan način odabiremo žabe koje će tvoriti svaki od mempleksa. Upravo zbog toga, koristimo *profinjenje* kako bismo zaista u svakoj generaciji dobili što bolje rješenje. Podrobnije, uzmimo npr. prvu iteraciju i njenih nasumično odabranih 5 mempleksa i nazovimo ih *A,B,C,D,E*. Bez smanjenja općenitosti, promatrajmo mempleks *A*. Vjerojatnost da baš sve žabe iz memepleksa *A* završe kasnije opet u istom mempleksu je skoro pa jednaka 0, i upravo to je razlog zašto želimo izvući najbolje moguće rješenje *profinjivanjem*. Ukratko, iz svakog kreiranog mempleksa želimo izvući najbolje jer se isti takav najvjerojatnije više neće kreirati.

Ono što je zanimljivo jest da ovaj algoritam gleda roditelja s najlošijom podobnošću i njega pokušava poboljšati. To radi na način da gleda razliku pozicija najboljeg i najlošijeg roditelja, zatim tu razliku množi po točkama s uređenim parom slučajno odabranih konstanti između $(0, 1)$, te s konstantom *step* (koja je u našem slučaju odabrana da bude jednaka 2). Dakle, ako je (x, y) razlika pozicija najboljeg i najlošijeg roditelja, a (a, b) uređeni par dviju nasumičnih konstanti iz $(0, 1)$, onda je $korak = (step * a * x, step * b * y)$. Konačno,

traženi potomak je najlošiji roditelj *pomaknut* s trenutnog mjesta za taj dobiveni *korak*. Ukoliko je pozicija potomka unutar zadane okoline ($[-10, 10] \times [-10, 10]$), izračunamo mu podobnost i usporedimo s najlošijim roditeljem: ako je podobnost potomka veća od podobnosti najlošijeg roditelja, onda potomak preživljava, a roditelj otpada. U suprotnom algoritam ponavlja još jednom postupak gdje traži *korak*, odnosno novog potomka. Ako niti taj drugi put ne nađe bolje rješenje, onda najlošijeg roditelja zamijeni nasumično odabranom jedinkom iz tražene okoline. Ovaj dio algoritma dan je pseudokodom u tablici 3.2.

3.3 Dobiveni rezultati

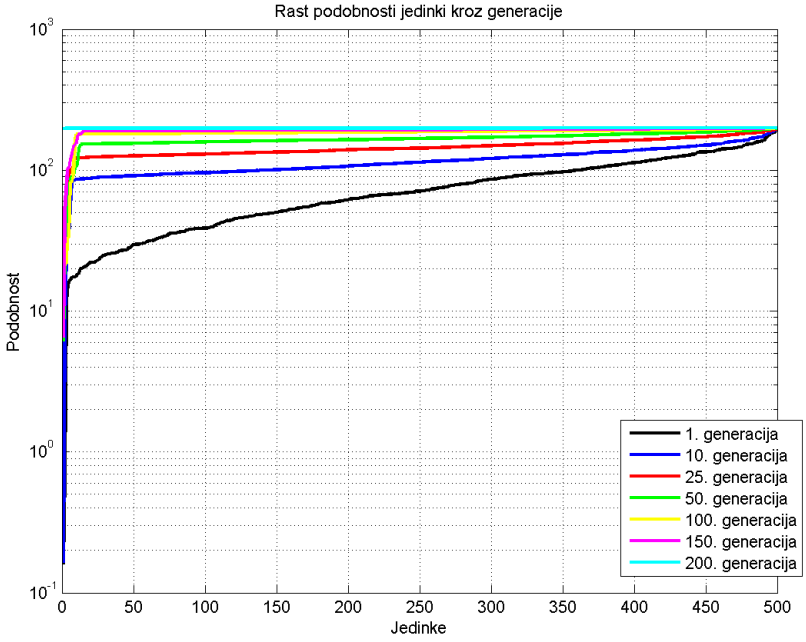
Konačno, preostalo je još samo interpretirati dobivene rezultate. Ponovimo samo ukratko



Slika 3.1: Podobnost inicijalne populacije

tehničke detalje: uzeli smo populaciju od 500 žaba koje u svakom koraku nasumično dijelimo u 5 mempleksa po 100 žaba. Unutar svakog mempleksa biramo 30 roditelja s obzirom na vektor vjerojatnosti odabira, zatim neovisno vršimo lokalno pretraživanje u svakom od njih, te na kraju 3 najlošija roditelja budu zamijenjena s 3 dobivena potomka i to nekoliko puta (*profinjenje*). Za maksimalni broj iteracija uzeli smo 500, što se pokazalo i više nego dovoljno. Mjerili smo i vrijeme svake iteracije pomoću MATLAB-ovih funkcija *tic* i *toc*.

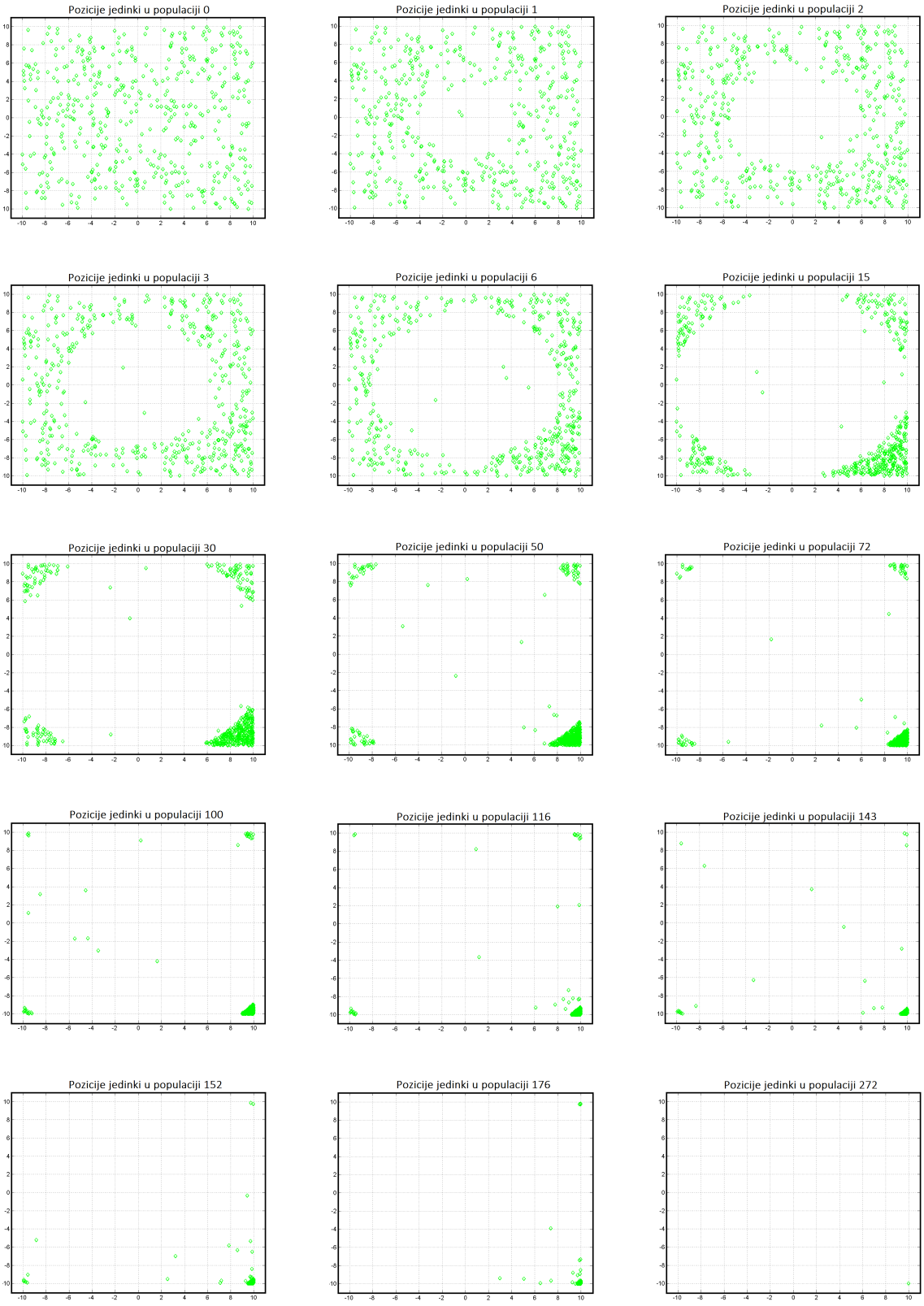
Pogledajmo sliku 3.3 koja prikazuje kretanje žaba u jednom pokretanju algoritma. Lako se vidi da se već nakon par koraka žabe grupiraju oko 4 lokalna optimuma, iako na kraju završe u samo jednom od njih. Također se vidi da količinski više žaba otpočetka gravitira tom krajnjem, mogli bismo reći globalnom, optimumu. Štoviše, uočeno je da u svakom pokretanju algoritma dobivamo neki drugi globalni optimum. No, to ne iznenađuje s obzirom na to da je inicijalna populacija uvijek nasumična.



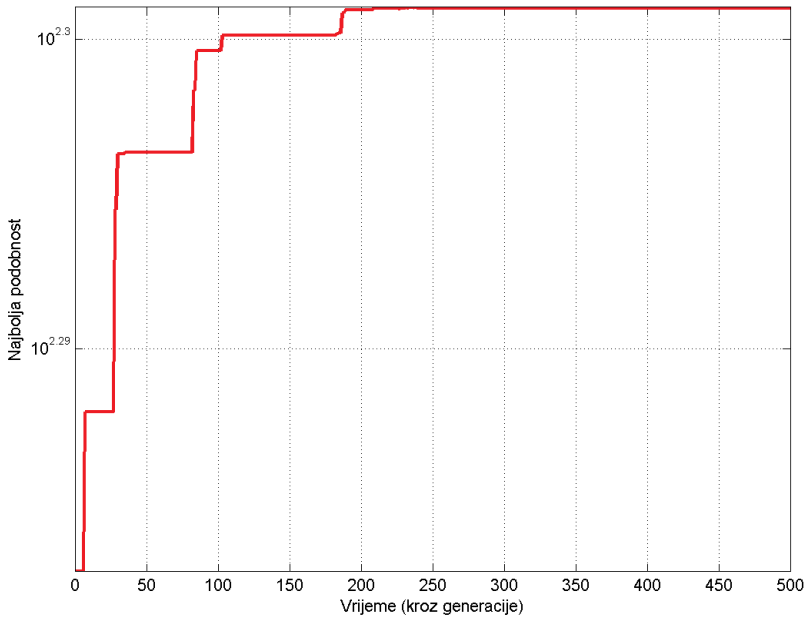
Slika 3.2: Podobnost populacije u raznim generacijama

Nadalje, proučimo malo podobnost. Na slici 3.1 prikazana je podobnost početne populacije žaba- očekivano imamo razne vrijednosti jer smo nasumično birali populaciju. S druge pak strane, graf na slici 3.2 prikazuje podobnost populacije u određenim generacijama. Uspoređujući graf 3.1 s linijom podobnosti prve generacije na slici 3.2 vidimo izrazito veliki napredak, i to nakon samo jednog koraka algoritma. Ovo svojstvo već smo spominjali ranije, u odjeljku 1.6. Radi se o tzv. *anytime* svojstvu evolucijskog, odnosno memetičkog algoritma, a to znači da se podobnost rapidno poboljšava u prvih par generacija, a ostalo vrijeme u pravilu stagnira. Ta značajka je najuočljivija na grafu 3.4 koji prikazuje rast najbolje podobnosti kroz vrijeme. Zadovoljavajuće rezultate dobivamo već nakon 50 iteracija, a u 272. generaciji dostižemo globalni optimum.

Na kraju dajemo uvid u trajanje izvršavanja naših iteracija. Najbolje vrijeme izvršavanja jedne iteracije bilo je 0.0977 sekundi, dok je najlošije bilo 0.1872 sekundi. Za postizanje



Slika 3.3: Kretanje žaba s obzirom na funkciju cilja $f(x, y) = x^2 + y^2$



Slika 3.4: Rast najbolje podobnosti kroz vrijeme

globalnog optimuma, tj. za 272 iteracije bilo je potrebno 15.862254 sekundi.

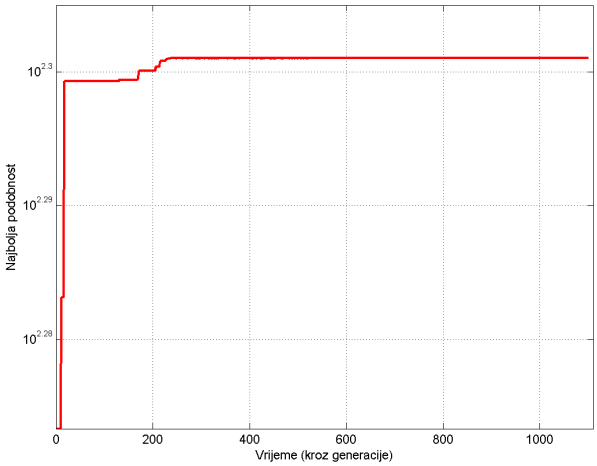
3.4 Variranje parametara

Isti problem promatramo i dalje, tj. imamo populaciju od 500 žaba koje obitavaju na okolini $[-10, 10] \times [-10, 10]$ i želimo maksimizirati funkciju cilja $f(x, y) = x^2 + y^2$ gdje je (x, y) pozicija svake žabe. Ono što ćemo varirati jest broj mempleksa.

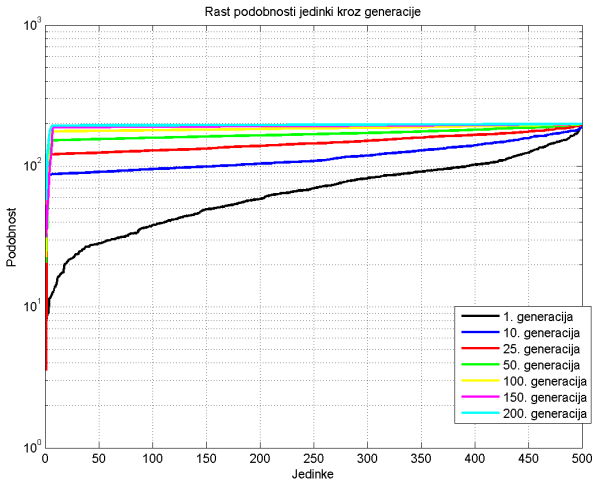
3.4.1 2 mempleksa po 250 žaba

Dakle, sve je isto kao u prethodnom primjeru osim što u svakom koraku našu populaciju žaba dijelimo u dva mempleksa po 250 žaba. U ovom slučaju broj roditelja u svakom mempleksu je 75, a broj potomaka 8.

Ovaj primjer je nešto lošiji od početnog. Kretanje žaba odvija se otprilike isto kao prije, ali s većim brojem iteracija. Slika 3.7 prikazuje kretanje žaba kroz generacije (pozornost obratiti na broj iteracija). Nadalje, prilažemo i graf rasta nabolje podobnosti kroz vrijeme i to na slici 3.5, gdje se vidi sporiji rast krivulje u odnosu na graf sa slike 3.4. Na slici



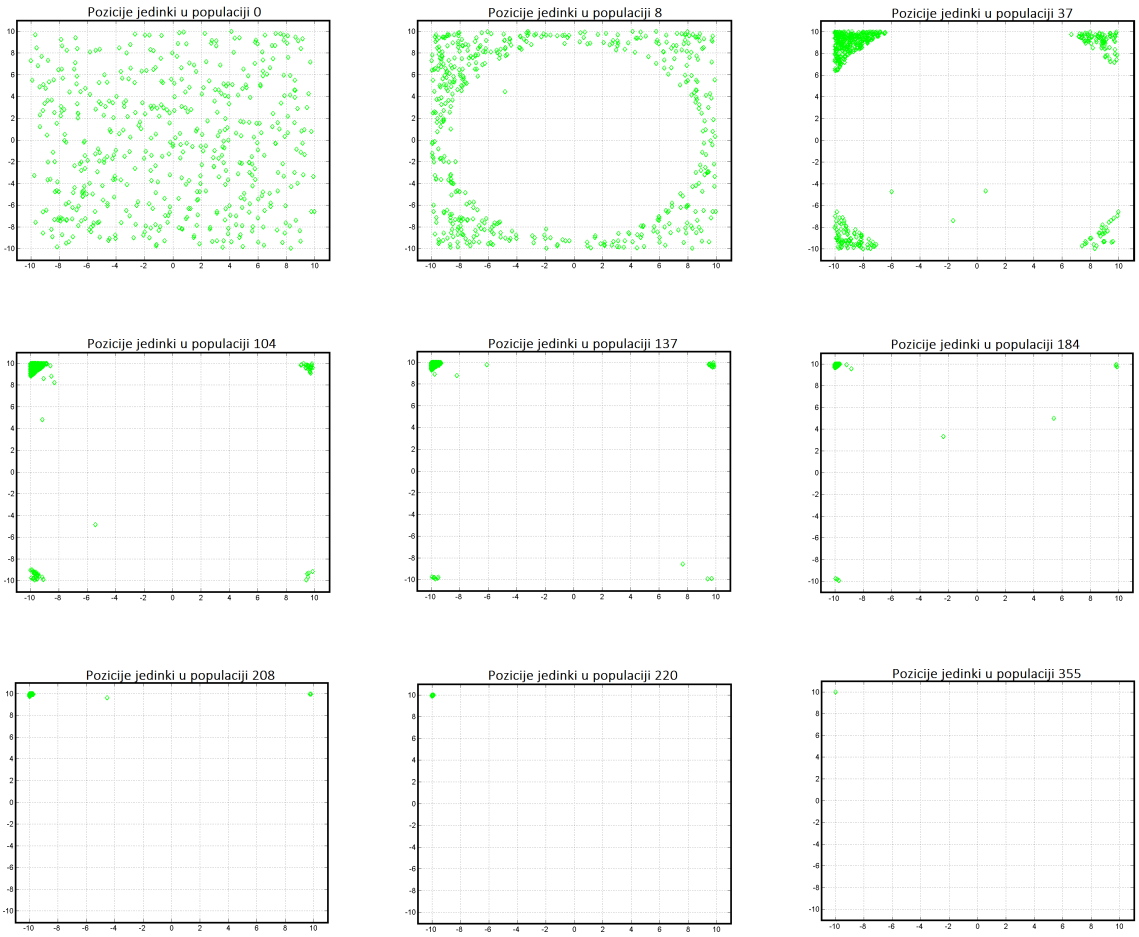
Slika 3.5: *2 mempleksa po 250 žaba*: Rast najbolje podobnosti kroz vrijeme



Slika 3.6: *2 mempleksa po 250 žaba*: Podobnost populacije u raznim generacijama

3.6 prikazane su krivulje podobnosti kroz više generacija. Globalni optimum postižemo u 355.-oj iteraciji, što je za skoro 100 iteracija lošije nego prvi primjer.

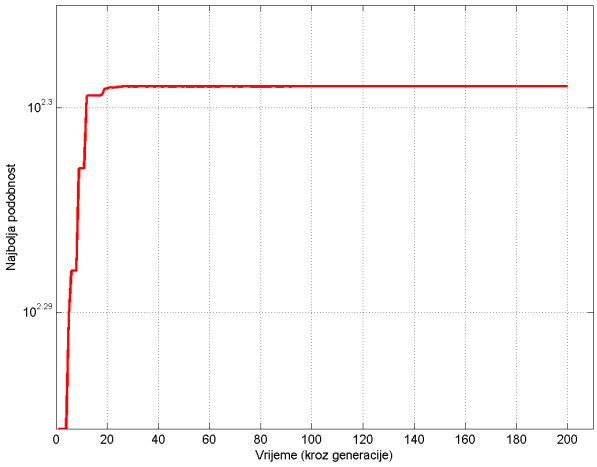
Što se vremena izvršavanja iteracija tiče, prilažemo sljedeće rezultate: najbrže izvršavanje iznosilo je 0.047699 sekundi, najsporije izvršavanje 0.428075 sekundi, a za globalni optimum kojeg smo postigli nakon 355 iteracija bilo je potrebno 23.8683 sekundi.



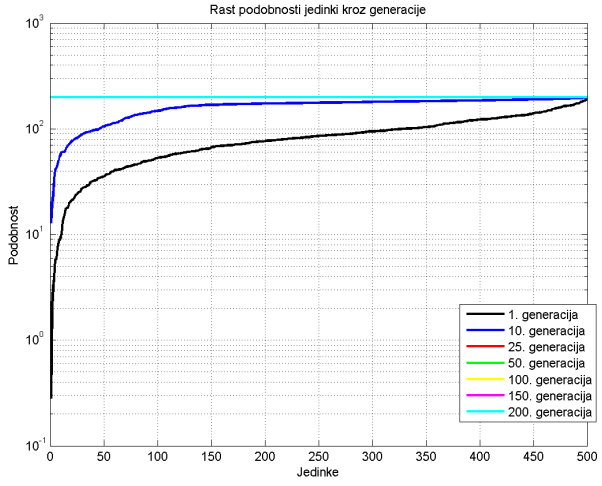
Slika 3.7: *2 mempleksa po 250 žaba*: Kretanje žaba s obzirom na funkciju cilja $f(x, y) = x^2 + y^2$

3.4.2 100 mempleksa po 5 žaba

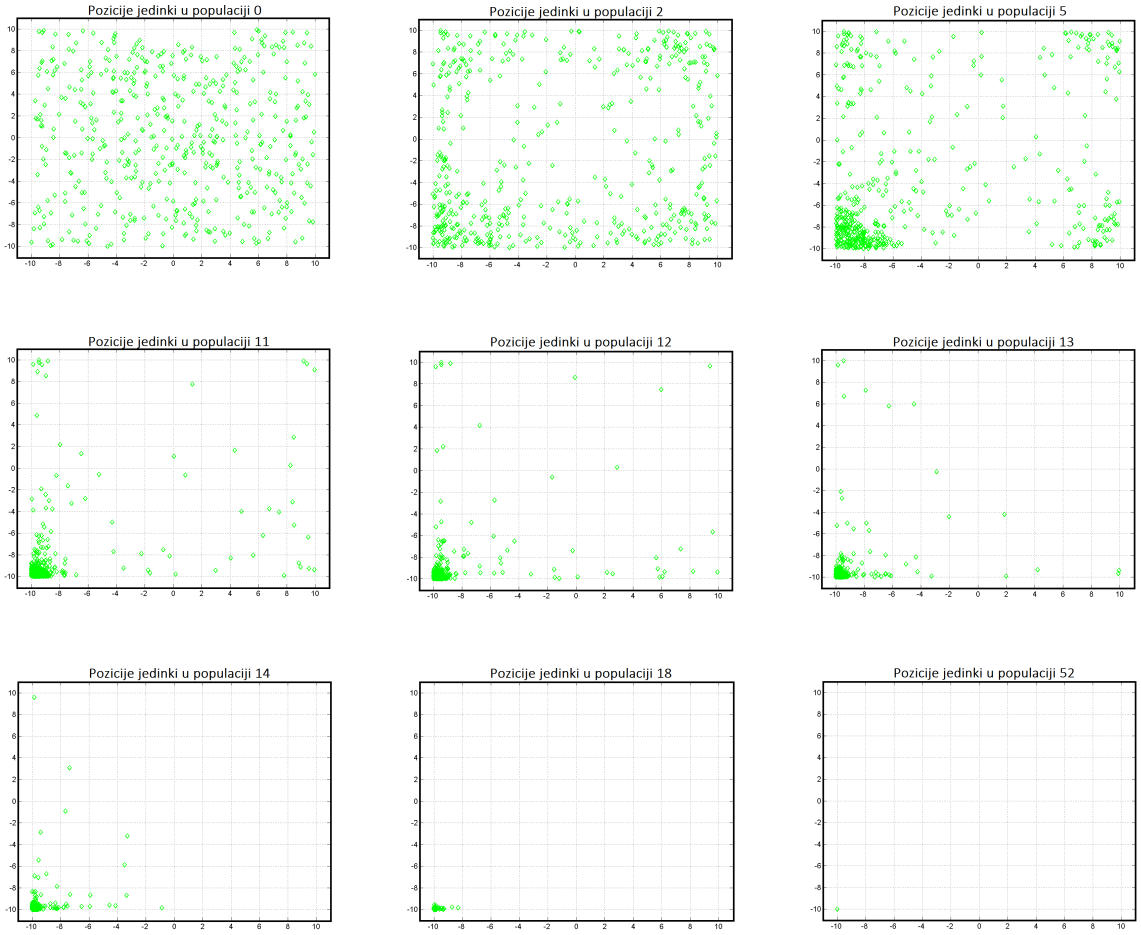
Proučimo slučaj kada imamo puno memepleksa i malo žaba unutar njih, točnije 100 mempleksa po 5 žaba. Tada između 5 jedinki biramo 2 roditelja, a broj potomaka je reduciran na 1.



Slika 3.8: *100 mempleksa po 5 žaba*: Rast najbolje podobnosti kroz vrijeme



Slika 3.9: *100 mempleksa po 5 žaba*: Podobnost populacije u raznim generacijama



Slika 3.10: *100 mempleksa po 5 žaba*: Kretanje žaba s obzirom na funkciju cilja $f(x, y) = x^2 + y^2$

U ovom slučaju jako brzo nalazimo optimalno rješenje. Od samog početka skoro sve jedinke gravitiraju jednom od 4 optimuma, a nakon samo 52 iteracije dostiže se globalni optimum.

Dajemo isti prikaz rezultata: slika 3.10 prikazuje kretanje žaba s obzirom na funkciju cilja, slika 3.9 prikazuje krivulje podobnosti kroz više generacija (s time da generacije 25-200 na grafu imaju skoro istu krivulju), te konačno slika 3.8 pokazuje krivulju rasta najbolje podobnosti.

Za ovaj slučaj također prilažemo dobivene vremenske rezultate: najbolje vrijeme je

0.114372 sekundi, najlošije 0.512488 sekundi, a 52 iteracije uzele su 10.0299 sekundi.

3.4.3 Zaključak

Variranjem broja mempleksa, odnosno broja jedinki na kojima se istovremeno vrši lokalno pretraživanje, uviđamo da je najbolje imati više mempleksa po manje žaba. Iako iz tablice 3.4 vidimo da se najbolje vrijeme izvršavanja jedne iteracije javlja kod manjeg broja mempleksa, a najlošije kod velikog broja mempleksa, sveukupno trajanje algoritma do postizanja globalnog optimuma ipak je najkraće kod velikog broja mempleksa jer je potreban manji broj iteracija. Dakle, istovremeno izvršavanje puno neovisnih lokalnih pretraživanja iziskuje dosta vremena, ali na kraju ipak daje najbolje rezultate.

Također, što se tiče broja iteracija opet pobjedu odnosi slučaj od 100 mempleksa po 5 žaba. Ostala dva slučaja nisu niti približno dobra i brza u dostizanju globalnog optimuma.

Vrijeme (sekunde)	Najbolje vrijeme	Najlošije vrijeme	Globalni optimum	u iteraciji
5 m. $\times$ 100 žaba	0.0977	0.1872	15.862254	272
2 m. $\times$ 250 žaba	0.047699	0.428075	23.8683	355
100 m. $\times$ 5 žaba	0.114372	0.512488	10.0299	52

Tablica 3.4: Vrijeme u sekundama potrebno za izvršavanje pojedinih iteracija i za postizanje globalnog optimuma

3.5 Usporedba s evolucijskim algoritmom

Na kraju još želimo pokazati da je naš memetički algoritam uistinu bolji od običnog evolucijskog algoritma.

3.5.1 Evolucijski algoritam za dani problem

Ponovno gledamo isti problem: 500 žaba na području $[-10, 10] \times [-10, 10]$ s funkcijom cilja $f(x, y) = x^2 + y^2$ koju želimo maksimizirati. Za razliku od memetičkog algoritma, ovdje nemamo više memplekse, nego potomke tražimo na razini cjelokupne populacije. Također, nema više niti vektora vjerojatnosti odabira roditelja, odnosno vjerojatnosti odabira su jednake za sve jedinke. Za maksimalni broj iteracija uzeli smo 5000 koraka. U tablici 3.5 dan je pseudokod za naš evolucijski algoritam.

Opišimo sada glavni dio programa. U tablici 3.6 opisan je odabir potomaka pomoću rekombinacije i mutacije. Prvo nasumično odaberemo 150 roditelja od sveukupno 500

START
funkcija_cilja$(x, y) = x^2 + y^2$;
okolina = $[-10, 10] \times [-10, 10]$;
max_br_iteracija = 5000;
veličina_populacije = 500;
br_roditelja = max $\{round(0.3 * veličina\_populacije), 2\}$;
% gdje je $round(x) = \lfloor x + \frac{1}{2} \rfloor$, odnosno tzv. najbliže cijelo
br_potomaka = max $\{round(0.1 * br\_roditelja), 1\}$;
INICIJALIZIRAJ NASUMIČNU POPULACIJU:
ZA $i = 1 : veličina\_populacije$ {
populacija(i).Pozicija = (1, 1) * slučajna_uniformna_distribucija_na_intervalu $[-10, 10]$;
populacija(i).Podobnost = funkcija_cilja(populacija(i).Pozicija);
}
Sortiraj populaciju u uzlaznom poretku s obzirom na podobnost;
GLAVNA PETLJA:
ZA $it = 1 : max\_br\_iteracija$ {
Odaberi nasljednike za sljedeću generaciju; (**pseudokod u tablici 3.6**);
Sortiraj populaciju u uzlaznom poretku s obzirom na podobnost;
}
KRAJ

Tablica 3.5: Pseudokod za evolucijski algoritam

žaba te idemo u potragu za 15 potomaka. Kada odaberemo roditelje i izvučemo ih iz populacije, sortiramo ih uzlazno s obzirom na podobnost. Najprije vršimo rekombinaciju dvaju najboljih roditelja tako da uzmemo 1. koordinatu prvog po redu najboljeg roditelja, i 2. koordinatu drugog po redu najboljeg roditelja. Zatim te dvije koordinate u istom rasporedu kopiramo na potomka i izračunamo mu podobnost. Dakle, ako je $A = (a_1, a_2)$ prvi po redu najbolji roditelj, i $B = (b_1, b_2)$ drugi po redu najbolji roditelj, onda je njihov $potomak = (a_1, b_2)$. Nakon ovog postupka idemo na mutaciju koja je definirana na sljedeći način: prvo nasumično odaberemo jednu od dviju koordinata potomka i pogledamo joj predznak. Ako je veća od nule onda joj pribrajamo slučajan broj između $[0, 0.1]$, inače joj oduzimamo slučajan broj iz istog segmenta. Na taj način dobivamo *mutanta*. Ako je dobiveni mutant unutar zadane okoline $[-10, 10] \times [-10, 10]$ onda mu izračunamo podobnost. Ako mu je podobnost veća od podobnosti potomka, onda zamijenimo potomka mutantom. Inače, potomak ostaje onakav kakav je bio prije mutacije. Na kraju, kada nađemo svih 15 potomaka najboljeg para roditelja, vratimo i roditelje i potomke u početnu populaciju koju zatim sortiramo uzlazno s obzirom na podobnost. Konačno, najlošijih 15 jedinki izbacimo iz populacije.

```
Nasumično odaberi roditelje i izvuci ih iz populacije;
ZA k = 1 : br_potomaka {
   Sortiraj roditelje uzlazno s obzirom na podobnost;
   % prvo radimo rekombinaciju dvaju najboljih roditelja:
   1. koordinata potomka je 1. koordinata prvog po redu najboljeg roditelja;
   2. koordinata potomka je 2. koordinata drugog po redu najboljeg roditelja;
   Izračunaj podobnost potomka;
   % sada idemo na mutaciju:
   Na slučajan način odaberi jednu od dvije koordinate potomka;
   AKO je ta odabrana koordinata veća od nule, ONDA {
      Zbroji tu koordinatu sa slučajnim brojem između [0, 0.1];
   }
   INAČE {
      Oduzmi od te koordinate slučajni broj između [0, 0.1];
   }
   Mutant=potomak nakon mutacije;
   AKO je mutant unutar okoline [−10, 10] × [−10, 10], ONDA {
      Izračunaj podobnost mutanta;
      AKO je podobnost mutanta veća od podobnosti potomka, ONDA {
         Potomak=mutant;
      }
   }
}
Vrati sve roditelje i dobivene potomke u populaciju;
Sortiraj cijelu novu populaciju uzlazno s obzirom na podobnost;
Izbaci najlošijih n jedinki iz populacije;    % gdje je n = br_potomaka
```

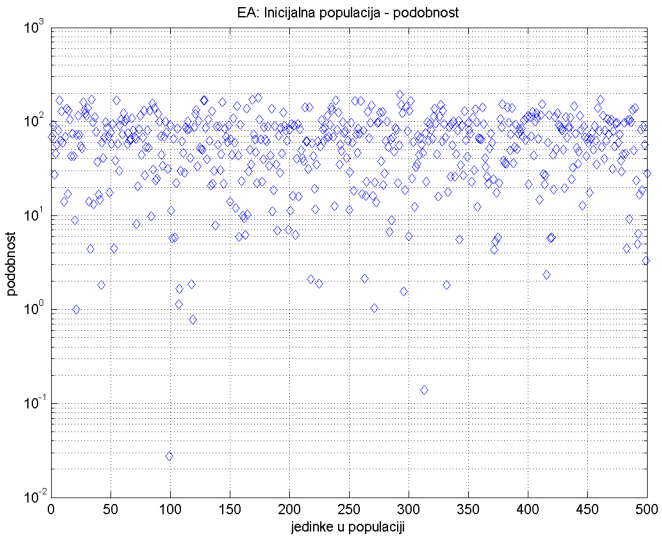
Tablica 3.6: Pseudokod za odabir potomaka rekombinacijom i mutacijom

3.5.2 Dobiveni rezultati

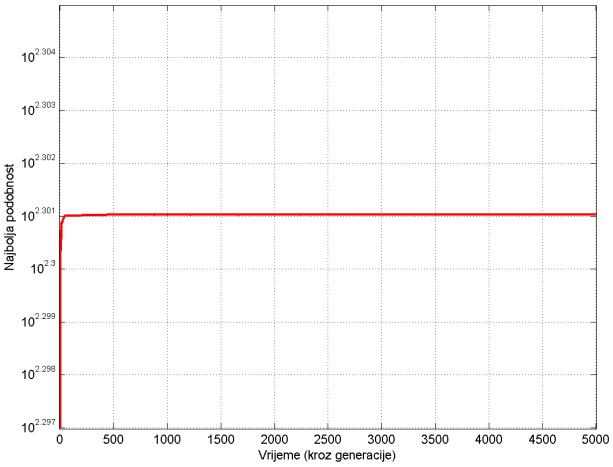
Napokon dolazimo do rezultata dobivenih pokretanjem upravo opisanog evolucijskog algoritma. Napomenimo da smo i u ovom slučaju također mjerili vrijeme pomoću MATLAB-ovih funkcija *tic* i *toc*.

Prvo prilažemo graf podobnosti inicijalne populacije na slici 3.11. Analogno kao i kod memetičkog algoritma, očekivano smo dobili razne vrijednosti jer smo populaciju odabrali na slučajan način.

Slika 3.12 pokazuje krivulju rasta najbolje podobnosti. Ponovno uočavamo *anytime* svojstvo kao i kod memetičkih algoritama, odnosno rapidni rast podobnosti u prvih par



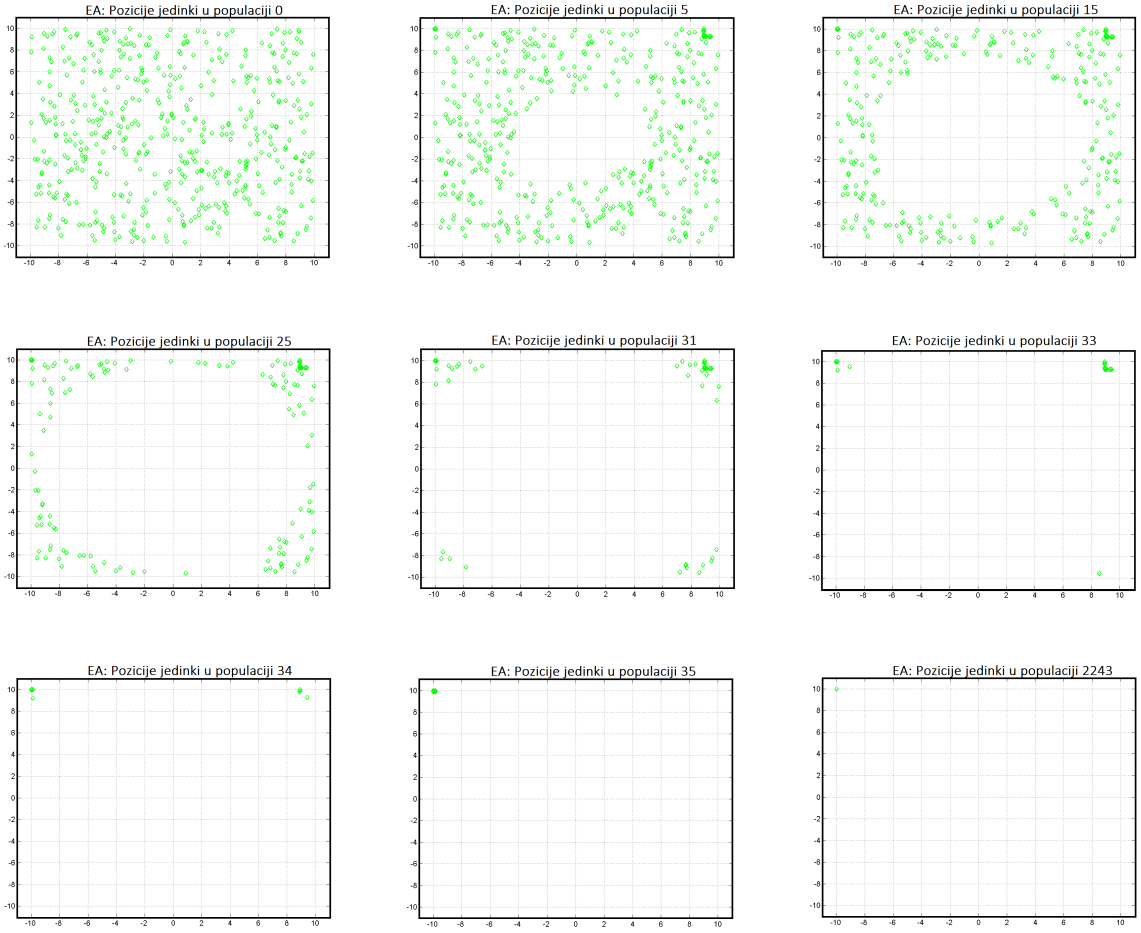
Slika 3.11: EA: Podobnost inicijalne populacije



Slika 3.12: EA: Rast najbolje podobnosti kroz vrijeme

iteracija te zatim stagnacija. Globalni optimum dostižemo tek u 2243.-oj iteraciji.

Sljedeće što trebamo uočiti je kretanje žaba kroz generacije na slici 3.13. Naime, slično kao kod memetičkog algoritma, žabe već u prvih par iteracija gravitiraju ka 4 lokalna optimuma. Nakon samo 35 iteracija okupljaju se oko globalnog optimuma, ali ga ne uspijevaju

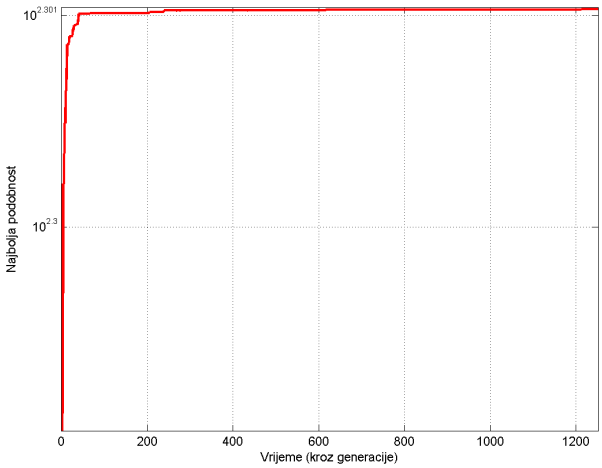


Slika 3.13: EA: Kretanje žaba s obzirom na funkciju cilja $f(x, y) = x^2 + y^2$

dostići. Iako već u prvih 300 iteracija dolazimo jako blizu globalnom optimumu (slika 3.14[2]), za postizanje tog optimuma trebalo nam je još skoro 2000 iteracija. O tome smo govorili u odjeljku 2.1 gdje smo navodili razloge za hibridizaciju evolucijskih algoritama. Naime, evolucijski algoritmi su dobri u brzom pronalaženju povoljnih područja za traženje rješenja (istraživanje), ali su manje dobri u završnoj fazi gdje bi trebali profiniti rješenje (iskorištavanje). Upravo ta značajka je ovdje najvidljivija.

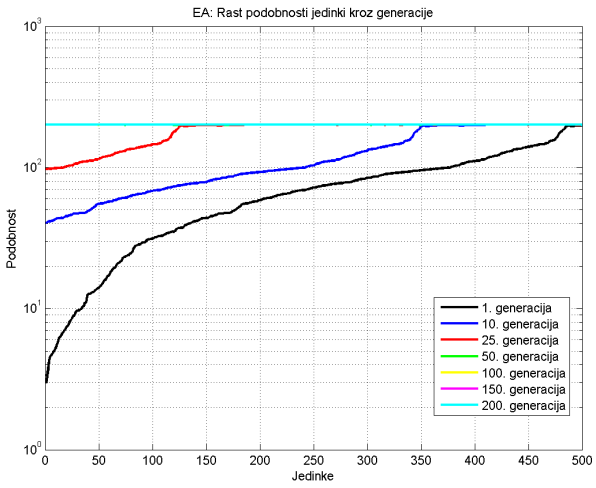
Slika 3.15 prikazuje krivulje podobnosti kroz više generacija. Na ovom grafu čak vi-

[2]Slika 3.14 je zapravo uvećani graf sa slike 3.12. Na njemu bolje vidimo rast podobnosti u prvih 1200 iteracija.



Slika 3.14: EA: Uvećani graf rasta najbolje podobnosti kroz vrijeme

dimo i bolju situaciju s obzirom na primjere memetičkih algoritama od 2 i 5 mempleksa. No, ovaj graf služi više kao ilustracija kako se podobnost mijenja i nema veze s tim u kojem smo trenutku dostigli globalni optimum niti koliko nam je vremena trebalo za to.



Slika 3.15: EA: Podobnost populacije u raznim generacijama

Na kraju, pogledajmo vremensko trajanje izvršavanja iteracija. U tablici 3.7 vidimo da je i najbolje i najlošije vrijeme izvršavanja jedne iteracije puno bolje nego kod sva tri

primjera memetičkih algoritama. Međutim, dostizanje globalnog optimuma je i vremenski i što se tiče broja iteracija daleko nakon memetičkog algoritma u svakom pogledu. Dakle, iako se same iteracije izvršavaju neusporedivo brže nego u slučaju memetičkog algoritma, na kraju ipak zbog prevelikog broja iteracija potrebnih za postizanje globalnog optimuma gubimo bitku i s najlošijim memetičkim algoritmom.

Vrijeme (sekunde)	Najbolje vrijeme	Najlošije vrijeme	Globalni optimum	u iteraciji
5 m. × 100 žaba	0.0977	0.1872	15.862254	272
2 m. × 250 žaba	0.047699	0.428075	23.8683	355
100 m. × 5 žaba	0.114372	0.512488	10.0299	52
EA[3]	0.01129	0.049253	29.013905	2243

Tablica 3.7: Vrijeme u sekundama potrebno za izvršavanje pojedinih iteracija i za postizanje globalnog optimuma

3.5.3 Zaključak

Iako se trošak po iteraciji povećao u memetičkim algoritmima (što se tiče vremena izvršavanja) u odnosu na EA, broj iteracija se drastično smanjio, pa je i ukupno vrijeme izvršavanja kraće. Kod memetičkih algoritama je to općenito stvar 'trgovine': negdje nešto dobijemo (manji broj iteracija), nešto izgubimo (povećava nam se vrijeme izvršavanja pojedine iteracije), ali ukupan rezultat je bolji od polaznog evolucijskog algoritma. Stoga, s obzirom na sve promatrane parametre, memetički algoritam zaista jest bolji od običnog evolucijskog algoritma.

[3]EA- Evolucijski algoritam

Bibliografija

[1] J. M. Baldwin, *A new factor in evolution*, The American Naturalist **30** (1896.), br. 354, str. 441–451.

[2] E. K. Burke i J. P. Newall, *A multi-stage evolutionary algorithm for the timetable problem*, IEEE Transactions on Evolutionary Computation, IEEE, 1999., str. 63–74.

[3] Richard Dawkins, *The Selfish Gene*, Oxford University Press, 1976.

[4] ———, *Sebični gen*, Izvori, 2007., prijevod: Petar Kružić.

[5] F. Glover, *Tabu search: 1.*, ORSA Journal on Computing **1(3)** (1989.), str. 190–206.

[6] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[7] N. Krasnogor, *Studies in the Theory and Design Space of Memetic Algorithms*, Disertacija, University of the West of England, 2002.

[8] N. Krasnogor, B. P. Blackburne, E. K. Burke i J. D. Hirst, *Multimeme algorithms for protein structure prediction*, Proceedings of the 7th Conference on Parallel Problem Solving from Nature (J. J. Merelo Guervos, P. Adamidis, H. G. Beyer, J. L. Fernandez-Villacanas i H. P. Schwefel, ur.), Lecture Notes in Computer Science, br. 2439, Springer, 2002., str. 769–778.

[9] N. Krasnogor i J. E. Smith, *Emergence of profitable search strategies based on a simple inheritance mechanism*, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (L. Spector, E. Goodman, A. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon i E. Burke, ur.), Morgan Kaufmann, San Francisco, 2001., str. 432–439.

[10] W.B. Langdon i Riccardo Poli, *Foundations of Genetic Programming*, Springer, 2002.

[11] Hrvatski leksikon, *Definicija determinizma (pridjev : deterministički)*, `https://www.hrleksikon.info/definicija/determinizam.html`.

[12] Robert Manger i Miljenko Marušić, *Strukture podataka i algoritmi, skripta*, Zagreb, rujan,2003.

[13] MathWorks, *Objašnjenje MATLAB-ove funkcije 'randsample'*, https://www.mathworks.com/help/stats/randsample.html.

[14] P. Merz i B. Freisleben, *Fitness landscapes and memetic algorithm design*, New Ideas in Optimization (D. Corne, M. Dorigo i F. Glover, ur.), McGraw Hill, London, 1999.

[15] R. Meuth, M. H. Lim, Y. S. Ong i D. C. Wunsch, *A proposition on memes and meta-memes in computing for higher-order learning*, Memetic Computing, Springer, 2009., str. 85–100.

[16] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, 1996.

[17] Pablo Moscato, *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms*, 1989., http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.9474&rep=rep1&type=pdf.

[18] S. Mostapha Kalami Heris (Member of Yarpiz Team), *Implementation of Shuffled Frog Leaping Algorithm (SFLA)*, Yarpiz (www.yarpiz.com), 2015., http://yarpiz.com/71/ypea109-shuffled-frog-leaping-algorithm.

[19] Y. S. Ong, M. H. Lim, N. Zhu i K. W. Wong, *Classification of adaptive memetic algorithms: A comparative study*, IEEE Transactions on Systems Man and Cybernetics Part B **36(1)** (2006.), str. 141–152.

[20] Dario Pavić, *Uzorci i uzorkovanje*, https://www.hrstud.unizg.hr/_download/repository/Uzorci_i_uzorkovanje.pptx.

[21] Kemijski rječnik, *Definicija konformacije*, https://glossary.periodni.com/glosar.php?page=3&hr=T-oblik+molekule.

[22] A. E. Eiben & J. E. Smith, *Introduction to Evolutionary Computing*, Springer, 2015.

[23] J. E. Smith, M. Bartley i T. C. Fogarty, *Microprocessor design verification by two-phase evolution of variable length tests*, Proceedings of the 1997 IEEE Conference on Evolutionary Computation, IEEE Press, 1997., str. 453–458.

[24] P. Surry i N. Radcliffe, *Innoculation to initialise evolutionary search*, Evolutionary Computing: Proceedings of the 1996 AISB Workshop (T. C. Fogarty, ur.), Springer, 1996., str. 269–285.

[25] P. D. Turney, *How to shift bias: lessons from the Baldwin effect*, Evolutionary Computation **4(3)** (1996.), str. 271–295.

[26] Wikipedia, *CPU (Procesor)*, https://hr.wikipedia.org/wiki/Procesor_(ra%C4%8Dunarstvo).

[27] _____, *Definicija anytime algoritma*, https://en.wikipedia.org/wiki/Anytime_algorithm.

[28] _____, *Definicija mempleksa*, https://en.wikipedia.org/wiki/Memeplex.

[29] _____, *Definicija NP-potpunih problema*, https://hr.wikipedia.org/wiki/Ra%C4%8Dunska_teorija_slo%C5%BEenosti.

[30] _____, *Definicija 'ponovnog izuma kotača'*, https://en.wikipedia.org/wiki/Reinventing_the_wheel.

[31] _____, *Definicija stohastičkog algoritma*, https://hr.wikipedia.org/wiki/Algoritam.

[32] D. H. Wolpert i W. G. Macready, *Transactions on Evolutionary Computation*, IEEE, 1997.

[33] Leksikografski zavod Miroslav Krleža, *Definicija stohastičkog procesa*, 2017, http://www.enciklopedija.hr/natuknica.aspx?id=58201.

[34] _____, *Jean-Baptiste de Lamarck*, http://www.enciklopedija.hr/natuknica.aspx?id=35232.

Sažetak

Na početku ovog rada opisali smo evolucijske algoritme i njihove značajke. Evolucijski algoritmi najčešće se koriste za rješavanje problema optimizacije, a baziraju se na principima Darwinove teorije o prirodnoj selekciji. Temeljni način evolucijskog rješavanja problema oslanja se na metodu pokušaja i pogreške. Ideja je jedinke što bolje prilagoditi nekoj danoj okolini s obzirom na određene karakteristike. Intuitivno, želimo izvesti analogiju između evolucije u stvarnom svijetu i evolucijskog programiranja na način da okolina predstavlja zadani problem koji treba riješiti, jedinke predstavljaju rješenja, a podobnost jedinki (eng. fitness) predstavlja koliko je to rješenje dobro. Do boljeg rješenja u svakoj generaciji dolazimo pomoću operatora varijacije: mutacije i rekombinacije. Njih primijenjujemo na roditelje u svrhu dobivanja podobnijih potomaka, odnosno boljih rješenja.

Memetički algoritmi zapravo su evolucijski algoritmi kombinirani s drugim tehnikama ili pak nadograđeni nekim metodama ili strukturama podataka. Ono što najčešće želimo implementirati jesu podaci o samom problemu koji želimo riješiti. Važan dio memetičkog programiranja je lokalno pretraživanje. U najopćenitijim crtama to je iterativni proces koji ispituje skup točaka oko trenutnog rješenja, i ukoliko nađe bolje rješenje u tom skupu, onda ga postavi za novo trenutno rješenje. Također, operatori koji se koriste u evolucijskom programiranju mogu se nadograditi stečenim znanjima ili pak možemo odmah u inicijalizaciju umetnuti informacije koje bi nam pomogle pri rješavanju problema. Time automatski dobivamo bolji i efikasniji algoritam.

Na kraju smo teoriju demonstrirali praktičnim primjerom u MATLAB-u i to algoritmom skakutanja žaba (eng. Shuffled Frog Leaping Algorithm- SFLA). Gledali smo populaciju od 500 žaba na području $[-10, 10] \times [-10, 10]$ s funkcijom podobnosti $f(x, y) = x^2 + y^2$, gdje je (x, y) pozicija svake žabe. Primjer nam je dao očekivane rezultate s obzirom na teorijsku podlogu koju smo obradili u prva dva poglavlja.

Pokazalo se da je upotreba memetičkih algoritama izrazito korisna u praksi i čini istraživačko područje koje posjeduje veliki potencijal za daljnja istraživanja.

Summary

At the beginning of this graduate thesis we have described evolutionary algorithms and their features. They are most commonly used for solving optimization problems and they form a class of algorithms that are based on the Darwinian principles of natural selection. The fundamental way of evolutionary computing relates to a particular style of problem solving– that of trial and error. The idea is to adapt individuals to better suit the given environment. Intuitively, we want to link the evolution in 'real world' to the evolutionary computing and we do that in a way that environment represents a given problem, individuals represent solutions, and fitness of every individual represents a measure of how good solution solves a given problem. In every generation we make new (better) solutions by using variation operators: mutation and recombination. They are applied to the so-called parents, producing one or more children (new solutions).

Memetic algorithms are actually evolutionary algorithms combined with other techniques or have other methods or data structures incorporated within them. In most cases we want to include information about the problem we are solving into the evolutionary algorithm. The important phase of memetic algorithm is local search. Briefly described, local search is an iterative process of examining the set of points in the neighbourhood of the current solution, and replacing it with a better neighbour if one exists. Variation operators that are used in evolutionary algorithms can also be upgraded by incorporating problem-specific knowledge, or we can just incorporate that information in initialization phase as well. By making these changes we instantly get a better and more efficient algorithm.

In the end, we demonstrated the theory with a practical example in MATLAB. The example we have described is called Shuffled Frog Leaping Algorithm (SFLA). We observed the population of 500 frogs in the $[-10, 10] \times [-10, 10]$ environment with fitness function $f(x, y) = x^2 + y^2$, in which (x, y) represents position of each frog. The example yielded the expected results considering the theory we have explained in the first two chapters.

It turned out that memetic algorithms are very successful in practice and they form a rapidly growing research area with great potential.

Životopis

Marija Ilijaš rođena je 16.02.1992. u Zagrebu. Živi u Sesvetama gdje je stekla osnovnoškolsko i srednjoškolsko obrazovanje. Nakon završene opće gimnazije s odličnim uspjehom, 2010. godine upisuje preddiplomski studij Matematike na PMF-u u Zagrebu. Stjecanjem akademske titule sveučilišne prvostupnice matematike, 2015. godine upisuje diplomski studij Primijenjene matematike na istom fakultetu.