UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Zorana Mitrović-Varga

Metaheuristike

master rad

mentor:
prof. dr Đorđe Dugošija

Beograd, 2008.

Sadržaj

1. Uvod

Jedna od glavnih odlika računara je da u kratkom vremenu egzaktno reši složene računske operacije koje su sastavni deo nekog problema. Ipak, uz svu tehnologiju i napredak postignut u računarskoj nauci, postoje problemi koji današnjim metodama nisu rešivi u stvarnom vremenu. Posebnu grupu problema čine NP-teški i NP-potpuni problemi. U NP-teške i NP-potpune probleme spadaju i mnogi kombinatorni problemi, poput problema trgovačkog putnika i problem zadovoljenja Booleove funkcije. Poznati algoritmi za rešavanje NP-teških problema su u najboljem slučaju eksponencijalne složenosti, tj. vreme izvođenja je eksponencijalnog rasta. Za ilustraciju, to znači da se pojedini problemi ne mogu poznatim algoritmima rešiti milijardama godina. Pitanje je kako pristupiti takvim problemima.

Praksa nas uči da često nije potrebno rešiti probleme egzaktno, tj. dovoljno ih je rešiti približno. U tu svrhu koristimo se nekakvim iskustvenim metodama čiji je učinak eksperimentalno potvrđen. Neke od tih metoda su i heuristički algoritmi.

Reč *heuristika* potiče od grčke reči heurisko što znači pronašao sam. Odavde se da naslutiti da su heuristički algoritmi zapravo algoritmi nastali eksperimentisanjem u svrhu dobijanja zadovoljavajućeg rešenja. Bitno svojstvo heurističkih algoritama je da mogu približno (dovoljno dobro) rešiti probleme eksponencijalne i faktorijelne složenosti. Ipak, valja napomenuti da rešavanje problema heurističkim algoritmima ne mora voditi ka zadovoljavajućem rešenju, a za neke probleme, heuristički algoritmi pokazuju relativno loše rezultate. To se pogotovo odnosi na probleme za koje postoje egzaktni algoritmi polinomske složenosti. Isto tako, heuristički algoritmi nisu jednoznačno određeni. Heuristički algoritmi se razlikuju u pojedinim delovima zavisno od situacije u kojoj se koriste. Ti su delovi uglavnom funkcije cilja, koje u najvećem broju algoritama predstavljaju funkciju koja se optimizuje ili funkciju koja je direktan pokazatelj konvergencije ka optimalnom rešenju, tako da njihovo definisanje znatno utiče na efikasnost algoritma.

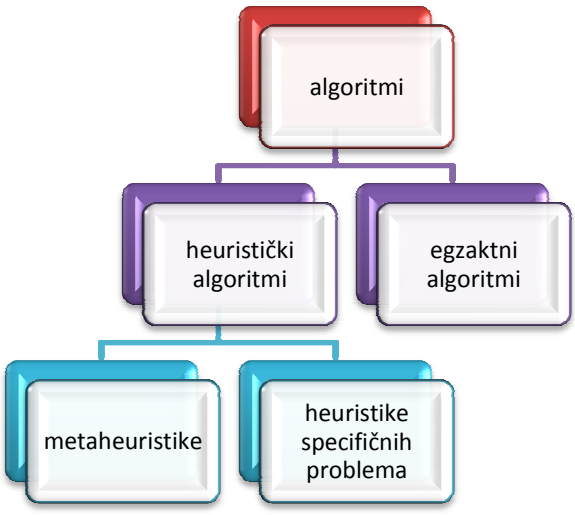
Cilj ovog rada je dati pregled osnovnih heurističkih algoritama, a neke i pobliže opisati. U prvom delu se daje kratak opis heurističkih algoritama i nabrajaju se moguće podele metaheurističkih algoritama. Bira se podela na algoritme inspirisane prirodom i algoritme koji nisu prirodom inspirisani. Pobliže se opisuju: lokalno pretraživanje, tabu pretraživanje, metoda promenljivih okolina, genetski algoritmi, mravlje kolonije kao i pohlepni algoritam i algoritam simuliranog kaljenja. Na kraju je dat kratak opis preostalih heurističkih algoritama.

2. Pregled heurističkih algoritama

Pre nego što se napravi podela heurističkih algoritama, potrebno je saznati šta predstavlja heuristika. *Heuristika* je pravilo temeljeno na našem iskustvu pomoću kojeg mi tražimo rešenje nekog problema. Pomoću heuristike se mogu pronalaziti optimalna rešenja, ali i procenjivati odluke. Heuristički algoritmi se temelje na heuristici i uglavnom se koriste u rešavanju optimizacionih problema za čije rešavanje nisu poznati algoritmi polinomske složenosti. Glavna snaga heurističkih algoritama je da smanjuju prostor pretraživanja koristeći neke iskustvene spoznaje, pa tako znatno ubrzavaju proces pronalaženja rešenja. Ali, to znači i da heuristički algoritmi ne moraju uvek dati optimalno rešenje, a ponekad njihovo izvođenje može trajati duže od algoritama koji koriste egzaktne metode rešavanja. Dva su osnovna uslova koje algoritam mora zadovoljiti ako se želi pronaći globalni optimum:

- *uslov stabilnosti* (stabilizaciju u globalnom optimumu)
- *uslov oslobađanja iz lokalnog optimuma* (*beg* iz lokalnog optimuma).

2.1 Podela heurističkih algoritama



Najopštije posmatrano, metode optimizacije se mogu podeliti na *egzaktne* i *heurističke* (koje se dalje dele na *heuristike specifičnih problema* i *metaheuristike*). Egzaktne metode

su najpoželjnije jer garantuju optimalnost dobijenog rešenja, ali je njihova primenljivost u praktičnim problemima najmanja.
Osnovni problem kod rešavanja zadatka kombinatorne optimizacije je što je broj (dopustivih) rešenja, iako konačan, izuzetno veliki. Stoga je za probleme većih dimenzija nemoguće primenjivati egzaktne metode rešavanja. U cilju rešavanja takvih problema, došlo je do razvoja heurističkih metoda. *Heuristike* predstavljaju konačan skup koraka kojima se dobijaju rešenja problema kombinatorne optimizacije (bez garancije njihove optimalnosti) za relativno kratko vreme. Osnovna prednost heurističkih metoda je njihova brzina, što omogućava dobijanje zadovoljavajućih rešenja za probleme velikih dimenzija kakvi se najčešće javljaju u realnim primenama. Nedostatak je što za dobijeno rešenje, ne samo da ne postoji garancija optimalnosti, već najčešće ni procena kvaliteta dobijenog rešenja. Međutim, za probleme velikih dimenzija osnovni cilj je da se neko rešenje dobije, bez obzira na njegov kvalitet. Kada rešenje već postoji, mogu se primeniti razne tehnike za njegovo poboljšanje, što je u suštini osnovna ideja prilikom nastajanja metaheuristika (o kojima će više reči biti u nastavku ove glave).
Heurističke metode mogu se podeliti u nekoliko grupa [8].
(i) *konstruktivne heuristike*;
(ii) *heuristike iterativnog poboljšavanja*;
(iii) *heuristike matematičkog programiranja*;
(iv) *dekompozicione heuristike*;
(v) *heuristike bazirane na podeli dopustivog skupa*;
(vi) *heuristike bazirane na restrikciji dopustivog skupa*;
(vii) *heuristike bazirane na relaksaciji*.
(i) *Konstruktivne metode*. Ova grupa metoda postepeno gradi (konstruiše) rešenje maksimalno koristeći specifična znanja svakog pojedinačnog zadatka.
(ii) *Iterativno poboljšavanje*. Savremeni naziv za ovu grupu je *metode lokalnog pretraživanja* i biće im više pažnje posvećeno kasnije u tekstu. Polazi se od proizvoljnog početnog rešenja, koje se postepeno poboljšava pretraživanjem njemu "bliskih" rešenja. Proces se ponavlja sve dok u "blizini" tekućeg rešenja postoji bolje rešenje od tekućeg. Naravno, problem odredjivanja bliskih rešenja je ključan.
(iii) *Matematičko programiranje*. Problem se formuliše kao zadatak matematičkog programiranja, pa se onda rešava približnim (a ne egzaktnim) metodama.
(iv) *Dekompozicione heuristike*. Početni problem se razbije (dekomponuje) na više manjih podproblema, čijim se čak ni egzaktnim rešavanjem ne mora garantovati optimalnost rešenja za polazni problem.
(v) *Podela dopustivog skupa*. Dopustivi skup se podeli na više podskupova, pa se delimičnim pretraživanjem rešenja u svakom od njih nadje najbolje. Heurističko rešenje je ono kod kojega je funkcija cilja najbolja.
(vi) *Restrikcija dopustivog skupa*. Restrikcijom se jednostavno eliminišu određeni podskupovi dopustivog skupa i tako smanjuje prostor pretraživanja. To omogućava da se novi zadatak lakše rešava.
(vi) *Relaksacija*. Suprotan pristup od restrikcije je relaksacija, kojom se dopustiv skup proširuje, ali tako da se omogućuje jednostavnije rešavanje novog problema.
Naravno, u rešavanju nekog konkretnog problema, moguće je kombinovati različite tipove heurističkih pristupa. Na primer, primenom konstruktivne heuristike pronađe se

neko rešenje optimizacionog problema koje se zatim koristi kao polazno rešenje za iterativno poboljšavanje.
Kako su se heuristike pokazale kao praktično jedini način rešavanja optimizacionih zadataka, istraživačka populacija u poslednje vreme sve više pažnje posvećuje njihovom razvoju i usavršavanju. To je dovelo i do razvoja univerzalnih heuristika ili tzv. *metaheuristika*. Klasične heuristike su, uglavnom, bile namenjene rešavanju nekih konkretnih, pojedinačnih problema i koristile su poznate osobine datog problema pri njegovom rešavanju. *Metaheuristike*, naprotiv, sastoje se od uopštenih skupova pravila koja se mogu primeniti za rešavanje raznovrsnih problema optimizacije. Metaheuristički pristupi u rešavanju optimizacionih problema zasnovani su na opštim algoritmima optimizacije koji podrazumevaju primenu iterativnih postupaka u cilju popravljanja nekog postojećeg rešenja.
Sa druge strane, razvile su se mnoge metaheurističke metode optimizacije, najčešćce po uzoru na neke poznate procese, prvenstveno u biologiji (genetski algoritmi (GA) i razne specijalne varijante evolutivnih algoritama (EA), neuralne mreže (NN), mravlje kolonije (AC)), u fizici (simulirano kaljenje(SA)), ali i metode inspirisane lokalnim pretraživanjem sa raznim idejama da se izbegne zamka lokalnog minimuma (višestartno lokalno pretraživanje(MLS), metoda promenljivih okolina (VNS), tabu pretraživanje (TS), Greedy Randomized Adaptive Search Procedure (GRASP)) i druge.

Metaheurističke algoritme možemo podeliti po različitim osnovama, a najvažnije podele su [1]:

- *Prirodom-inspirisani algoritmi* i *algoritmi koji nisu prirodom inspirisani*. Primeri prirodom-inspirisanih algoritama su *optimizacija mravljom kolonijom* i *genetski algoritmi*, a primeri onih koji nisu prirodom inspirisani su *tabu pretraživanje* i *lokalno pretraživanje*.

- *Algoritmi koji imaju dinamičku funkciju cilja* i *algoritmi koji imaju statičku funkciju cilja.* Većina algoritama ne menja svoju funkciju cilja (funkcija koja se optimizuje), ali postoje algoritmi poput *vođenog lokalnog traživanja* čija se funkcija cilja menja u svrhu bežanja iz lokalnog optimuma. Takvi algoritmi imaju dinamičku funkciju cilja.

- *Algoritmi koji koriste jednu strukturu okoline* i *algoritmi koji koriste skup struktura okoline.* Algoritmi uglavnom koriste jednu strukturu okoline, ali postoje algoritmi, poput *metode promenljvih okolina*, koja koristi skup struktura okoline.

- *Algoritmi koji imaju mogućnost pamćenja prethodnih rešenja* i *oni koji to nemaju.* Primer, *tabu pretraživanje* ima mogućnost pamćenja prethodno odabranih rešenja, dok *algoritam simuliranog kaljenja* nema. On se koristi drugim metodama da bi odabrao dobro rešenje u sledećoj iteraciji.

- *Konstruktivni, poboljšavajući i hibridni algoritmi.* Konstruktivni algoritmi grade rešenje (primer, *pohlepni algoritam*). Poboljšavajući algoritmi biraju rešenje i usavršavaju ga kroz niz iteracija (primer, *algoritam simuliranog kaljenja*). Hibridni algoritmi su njihova kombinacija (primer, *GRASP*).

•*Algoritmi bazirani na populaciji rešenja* i *algoritmi putanje.* Algoritmi koji se baziraju na populaciji rešenja koriste skup rešenja u proceni trenutnog optimuma (primer, *Stohastic Diffusion Search*), dok algoritmi putanje koriste jedno trenutno rešenje u svakom koraku (primer *Lokalno pretraživanje*).

Metaheuristički algoritmi	***Podela 1***	*Prirodom-inspirisani algoritmi*
		Algoritmi koji nisu prirodom inspirisani
	Podela 2	*Algoritmi koji imaju dinamičku funkciju cilja*
		Algoritmi koji imaju statičku funkciju cilja
	Podela 3	*Algoritmi koji koriste jednu strukturu okoline*
		Algoritmi koji koriste skup struktura okoline
	Podela 4	*Algoritmi koji imaju mogućnost pamćenja prethodnih rešenja*
		Algoritmi koji nemaju mogućnost pamćenja prethodnih rešenja
	Podela 5	*Konstruktivni algoritmi*
		Poboljšavajući algoritmi
		Hibridni algoritmi
	Podela 6	*Algoritmi bazirani na populaciji rešenja*
		Algoritmi putanje

Metaheuristike su postale značajno, a najčešće i jedino, sredstvo u rešavanju realnih problema baziranih na optimizaciji. Osnovni zahtev je dobijanje rešenja bliskih optimalnom u razumnom vremenu. Osim toga, mogu se primeniti za ubrzavanje tačnih metoda tako što će se koristiti za dobijanje dobrog početnog rešenja. Skorašnje primene potvrđuju da su se pokazale veoma uspešnim i kao sastavni delovi sistema za otkrivanje znanja u okviru veštačke inteligencije [1].

U svetlu ovih primena mogu se definisati osobine koje efikasne metaheuristike treba da poseduju kako bi obezbedile značaj i na praktičnom i na teorijskom planu [17]:

Jednostavnost : treba da budu zasnovane na jednostavnim i lako razumljivim pravilima;
preciznost : koraci kojima se opisuje metaheuristička metoda treba da su formulisani preciznim, po mogućnosti matematičkim terminima;
doslednost : svi koraci metode treba da budu u skladu sa pravilima kojima je metaheuristika definisana;

efikasnost : primena metaheuristike na neki konkretan problem treba da obezbedi dobijanje rešenja bliskih optimalnom za većinu realnih primera, naročito za zvanične test primere (benchmarks) raspoložive u toj klasi;
efektivnost : za svaki konkretan problem, metoda mora da obezbedi optimalno ili rešenje blisko optimalnom u razumnom vremenu izvršavanja;
robusnost : metoda treba da daje podjednako dobre rezultate za širok spektar primera iz iste klase, a ne samo za neke odabrane test primere;
jasnoća : treba da bude jasno opisana kako bi se lako razumela i, što je još važnije, lako implementirala i koristila;
univerzalnost : principi kojima je metoda definisana treba da budu opšteg karaktera kako bi se sa lakoćom primenjivala na nove probleme.

3. Postavka problema kombinatorne optimizacije

U ovoj glavi je data opšta formulacija problema optimizacije i defnicije odgovarajućih pojmova, istaknute su različite metode rešavanja optimizacionih problema i naglašen značaj primene metaheuristika.
U opštem slučaju zadatak optimizacije se definiše na sledećci način [8]:

Definicija 1. *Neka je* $f : S \to R$ *realna funkcija definisana na skupu* S *i neka je* $X \subseteq S$ *neki zadati skup. Problem je naći*

$$\min f(x)$$

pod uslovom (ograničenjem)

$$x \in X.$$

Problem maksimizacije $f(x)$ se može svesti na minimizaciju $-f(x)$, tako da se neće posebno razmatrati.
Ako je X konačan ili prebrojiv skup, problem optimizacije je *problem kombinatorne* ili *diskretne optimizacije*, dok je u suprotnom reč o *kontinualnoj optimizaciji*. Elementi skupa S predstavljaju (potencijalna) *rešenja* problema, dok se u skupu X nalaze *dopustiva rešenja* optimizacionog problema. Skupovi S i X nazivaju se, redom, *prostor rešenja* i *prostor dopustivih rešenja*.
Funkcija $f(x)$ predstavlja *funkciju cilja*, a zadatak je naći ono rešenje x za koje $f(x)$ ima najmanju moguću vrednost. Takvo $x^* \epsilon\ X$ za koje važi $f(x^*) \leq f(x)$ za svako $x \epsilon\ X$ naziva se *optimalno rešenje* ili *globalni minimum*. Sva ostala (dopustiva) rešenja nazivaju se još i suboptimalnim.
Optimalno rešenje može biti jedinstveno, ali može postojati i više međusobno različitih elemenata skupa X koji odgovaraju istoj (minimalnoj, optimalnoj) vrednosti funkcije cilja. Rešenje problema optimizacije sastoji se najčešće u pronalaženju jednog optimalnog rešenja, ređe se zahteva pronalaženje svih optimalnih rešenja. Implementacija većine metaheurističkih metoda zasnovana je na pojmu dopustivog rešenja i definiciji

njegove okoline (u cilju primene iterativnog poboljšavanja). Da bi se definisala okolina nekog rešenja mora se uvesti pojam rastojanja između dveju tačaka.

Definicija 2. *Neka je X proizvoljan skup tačaka. Ako se svakom uređenom paru* (x, y) *tačaka iz X pridruži realan broj* $d(x, y)$ *koji ima osobine:*

1. $0 \leq d(x, y) < +\infty$;
2. $d(x, y) = 0 \iff x = y$;
3. $d(x, y) = d(y, x)$;
4. $d(x, y) \leq d(x, z) + d(z, y)$;

kaže se da je skup X snabdeven metrikom *d. Takav skup X, tj. uređeni par* (X, d)*, naziva se* metričkim prostorom*. Vrednost* $d(x, y)$ *predstavlja* rastojanje *između tačaka x i y.*

Definicija 3. *Skup* $N_d(x) \subseteq X$ *svih tačaka na rastojanju d od neke izabrane tačke* $x \in X$ *naziva se d-*okolina *tačke x. Elementi skupa* $N_d(x)$ *nazivaju se d-*susedi *tačke x.*

Opštija definicija može obuhvatiti sve tačke na rastojanju manjem ili jednakom *d*. Ukoliko je vrednost *d* fiksirana, govori se o *okolini* tačke *x* koja sadrži sve njene *susede*.

U optimizacionim problemima uobičajeno se koristi manje formalna definicija okoline i suseda nekog rešenja *x* [3]. Pod okolinom rešenja *x* se podrazumeva $N(x) \subseteq X$ koji je pridružen rešenju *x* po nekom pravilu. To je skup rešenja dobijenih od *x* primenom neke *elementarne transformacije*. Definicija elementarne transformacije zavisi od optimizacione metode koja se koristi kao i od njene konkretne implementacije. Obično se podrazumeva da rešenje *x* ne pripada svojoj okolini, tj. $x \notin N(x)$. U ovom slučaju se takođe mogu posmatrati *d*-okoline nekog rešenja pri čemu je *d* prirodan broj (a ne realan kao u definicijama 2 i 3). $N_d(x)$ je sada skup svih rešenja koja se dobijaju primenom *d* puta (iste) elementarne transformacije na rešenje *x*.

Definicija 4. *Rešenje* $x \in X$ *naziva se* lokalni minimum *nekog optimizacionog problema ukoliko ne postoji* $x' \in N(x) \subseteq X$ *takvo da važi* $f(x') < f(x)$.

Definicija 5. *Rešenje* $x \in X$ *naziva se* globalni minimum *nekog optimizacionog problema ukoliko ne postoji* $x' \in X$ *takvo da važi* $f(x') < f(x)$.

Lokalni minimumi se nazivaju *suboptimalna rešenja*. Obzirom na složenost optimizacionih zadataka, sve heurističke i metaheurističke metode usmerene su ka nalaženju što boljih suboptimalnih rešenja u što kraćem vremenu.

Osnovna ideja metaheurističkih metoda je da se polazi od jednog (ili više) mogućeg dopustivog rešenja (dobijenog konstruktivnom heurističkom metodom ili na slučajan način) i zatim se pokušava njegova popravka sve dok se nezadovolji neki izlazni kriterijum. Obzirom da su konstruktivne heuristike ograničene na dobijanje (najčešće jednog) suboptimalnog rešenja i da bi u mnoštvu postojećih heuristika bilo teško izabrati najpogodniju za dati problem (da ne govorimo o skupoći izvršavanja svih) značaj metaheuristika kao alata za efikasno popravljanje postojećih rešenja je evidentan. Prefiks *meta*-označava da je reč o uopštenim heuristikama, koje ne zavise od prirode problema

koji se rešava. Osim toga, ovaj prefiks govori i da se kod svake metode heuristike pojavljuju na više nivoa u procesu izvršavanja. Drugim rečima, jedna heuristika kontroliše izvršavanje neke druge heuristike.
Kao što je već više puta istaknuto, metaheuristike predstavljaju uopšteni skup pravila za rešavanje optimizacionih zadataka. Najčešće su bazirane na nekoj opštoj ideji ili analogiji sa prirodnim procesima (u fizici, biologiji, medicini,...). Te ideje se zatim razvijaju, modifikuju, proširuju, a često i kombinuju (hibridizacija) u cilju povećanja njihove efikasnosti. To ponekad može iskomplikovati primenu metode jer se njihova pravila usložnjavaju, parametri umnožavaju, a samim tim se prikriva, pa i gubi osnovna ideja metode. U ovom radu opisane su poznate metaheurističke metode (višestartno lokalno pretraživanje, metoda promenljivih okolina, tabu pretraživanje, genetski algoritam i mravlje kolonije), a dat je i osvrt na preostale poznate metaheurističke i heurističke metode.

4.Pregled metaheurističkih algoritama koji nisu prirodom inspirisani

U ovom poglavlju će biti razmatrane najpoznatije metaheuristike koje nisu prirodom inspirisane već uglavnom bazirane na lokalnom pretraživanju (MLS, VNS i TS).

4.1.Monte-Carlo metoda

Kao najjednostavnija metaheuristička metoda pominje se metoda Monte-Carlo [8]. Suština ove metode je da se iz prostora dopustivih rešenja X bira slučajna tačka koja predstavlja novo rešenje.Ukoliko je to rešenje bolje od trenutno najboljeg rešenja, usvaja se kao novo najbolje, u protivnom se odbacuje. Ovaj postupak se ponavlja sve dok se ne zadovolji neki, unapred zadati, izlazni kriterijum. Pseudokod metode Monte-Carlo ima sledeći oblik:

1. *Inicializacija*. Izabrati početno rešenje x, $x_{opt} = x$, $f_{opt} = f(x)$.
2. **Ponavljaj**
(a) *Pokušaj*. Izabrati slučajno rešenje x' u prostoru dopustivih rešenja X;
(b) *Provera rešenja*. **ako** $f(x') < f(x_{opt})$
onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$
dok nije zadovoljen kriterijum zaustavljanja.

Uobičajeni izlazni kriterijumi su broj pokušaja (broj slučajnih izbora novog rešenja, ili broj iteracija, kako se najčešće naziva) i broj pokušaja između dve popravke tekućeg najboljeg rešenja.
Metoda Monte-Carlo nastala je kao suprotnost detaljnom pretraživanju prostora dopustivih rešenja koje je praktično neizvodljivo zbog velikog broja takvih rešenja. Ova metoda veoma haotično pretražuje prostor dopustivih rešenja, te je stoga lako objasniti njenu neefikasnost prilikom traženja globalnog minimuma funkcije cilja, naročito u slučajevima problema velikih dimenzija ili problema kod kojih je prostor dopustivih rešenja izuzetno veliki. Stoga su se razvile mnoge metode koje pokušavaju da prevaziđu problem Monte-Carlo metode uvođenjem raznih pravila za sistematizaciju pretraživanja prostora dopustivih rešenja.

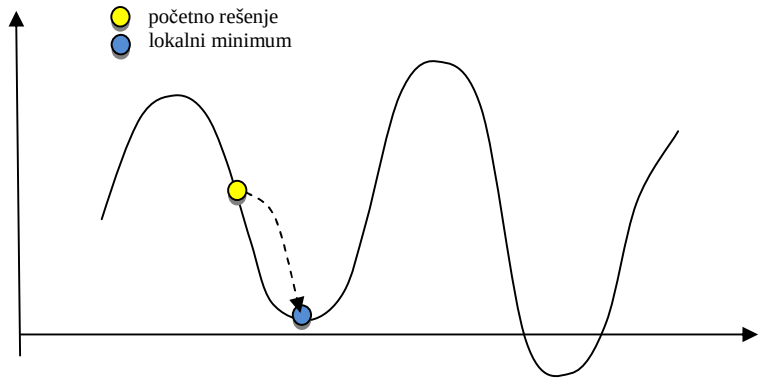
4.2. Lokalno pretraživanje (Local search, LS)

Metoda iterativnog poboljšavanja kod koje je pretraživanje ograničeno na unapred definisanu okolinu rešenja u potrazi za lokalnim minimumom naziva se lokalno pretraživanje.

Lokalno pretraživanje (Local search, LS) podrazumeva da se za svako $x' \in N(x)$ u okolini nekog početnog rešenja x, izračunava vrednost $f(x')$ i ukoliko je $f(x') < f_{min}$, čuva se novo $x_{min} = x'$ i $f_{min} = f(x')$. Kada se"obiđu" sva rešenja u okolini $N(x)$, nastavlja se pretraga u okolini $N(x_{min})$. Ispitivanje svih suseda nekog rešenja x_{min} naziva se pretraživanje okoline (neighborhood exploration, NE). NE predstavlja jednu iteraciju lokalnog pretraživanja. Proces lokalnog pretraživanja odvija se u iteracijama i zaustavlja se kada u okolini $N(x_{min})$ ne postoji bolje rešenje. Ilustracija izvršavanja lokalnog pretraživanja data je na slici 4.2.1. Pseudokod za LS može se prikazati u sledećem obliku.

1. *Inicijalizacija*. Izabrati početno rešenje x (slučajno ili primenom neke konstruktivne heuristike). Postaviti $x_{min} = x$ i $f_{min} = f(x)$.
2. **Ponavljaj**
POPRAVKA = 0;
    $\forall x' \in N(x_{min})$
      ako ($f(x') < f_{min}$) **onda**
     $x_{min} = x'$;
     $f_{min} = f(x')$;
POPRAVKA = 1;
    kraj-ako
dok nije POPRAVKA == 0;

Ponekad je isuviše sporo, pa čak i nemoguće pretraživati celu okolinu $N(x_{min})$. Stoga postoje razni načini (heuristike) kako da se odrede susedi koji "obećavaju" popravljanje vrednosti funkcije cilja, tj. da se na neki način smanji (redukuje) veličina okoline koja se pretražuje.



4.2.1 Grafički prikaz metode lokalnog pretraživanja

Osnovni nedostatak lokalnog pretraživanja je što se zaustavlja pri nailasku prvog lokalnog minimuma koji ne mora biti (i najčešće nije) globalni minimum, a to uveliko zavisi od početnog rešenja.

Primer : Lokalno pretraživanje na problemu SAT

Ilustrujmo algoritam lokalnog pretraživanja na prvom problemu za koji je pokazano da je NP-potpun. To je problem zadovoljenja Booleove formule, a češće se označava sa *SAT* (*SAT*(*x*) znači „x je formula koja je istinita za neke Booleove vrednosti"). Potrebno je otkriti postoji li kombinacija vrednosti promenljivih (istina ili laž) da Booleova formula bude istinita. SAT spada u klasu NP-potpunih problema.

Za bilo koju Bulovu funkciju, izraženu u obliku sume proizvoda (SOP - sum of products) ili proizvoda suma (POS - product of sums), postoji standardna ili kanonička forma. Kod obe alternativne forme svaka promenljiva se javlja bilo u komplementiranom bilo u nekomplementiranom obliku u svakom od članova tipa proizvod ili suma. Član tipa proizvod koji ima osobine naziva se *minterm*, dok član tipa suma koji poseduje ove osobine nazivamo *maksterm*.

Za Bulovu funkciju koja je u potpunosti komponovana od mintermova kažemo da ima kanoničku formu tipa SOP.

Na primer, $f(x,y,z) = \bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + x\bar{y}z + xyz$

predstavlja kanoničku funkciju od tri promenljive, jer svaki član tipa proizvod sadrži sve promenljive u funkciji. Promenljivu *v* i njen komplement−*v* nazivamo *literalima*. Ako je Bulova funkcija u potpunosti komponovana od makstermova, tada kažemo da ima kanoničku formu tipa proizvod suma.
Na primer,

$$f(x,y,z) = (x+y+z)(x+\bar{y}+\bar{z})(\bar{x}+y+\bar{z})(\bar{x}+\bar{y}+z)$$

predstavlja kanoničku funkciju od tri promenljive sa četiri maksterma.

Pogodno je Booleovu funkciju napisati u potpunom konjuktivnom obliku (proizvodu makstermi), te argumentima funkcije pridružiti binarni vektor. Taj vektor će predstavljati vrednost argumenata funkcije (0-laž, 1-istina). Problem ćemo rešiti **GSAT** algoritmom. GSAT algoritam se sastoji od dve petlje, jedna ugnežđena u drugoj. U prvoj petlji se nasumice bira jedan binarni vektor, a sama petlja ima konačno mnogo iteracija. Druga petlja je ugnežđena u prvu i kroz konačno mnogo iteracija proverava da li za binarni vektor Booleova funkcija poprima vrednost istine. Ako poprima, algoritam završava sa izvođenjem, a ako ne, menja se jedan od bitova binarnog vektora (vrednost jednog literala) i to onaj čijom promenom najveći broj makstermi poprima vrednost istine. Pri tome se može i smanjiti broj istinitih makstermi. Algoritam završava pronalaženjem binarnog vektora za koji Booleov izraz poprima vrednost istina ili prolaskom kroz obe petlje.

```
funkcija GSAT (funkcija *Booleova){
   binarni_vektor t;
   indeks p;
   za ( i = 1 do BROJ_POKUŠAJA) {
      t = slučajno_odaberi();
      za (j = 1 do BROJ_PROMENA){
         ako (Booleova(t)==istina) vrati t;
         odredi indeks bita p čijom promenom dobijamo
         najveći broj istinitih makstermi;
         t(p) = 1- t(p);
      }
   }
   vrati nisam našao zadovoljavajući vektor
}
```

Da bi izbegli pojavu zaglavljivanja u lokalnom optimumu, možemo za svaku makstermu uvesti oznaku težine koja će se povećavati unutar ugnežđene petlje dok god maksterma nema vrednost istine za binarni vektor t [37]. Sada će se menjati onaj bit binarnog vektora t čijom je promenom zbir težina neistinitih makstermi najmanji [36, 37].

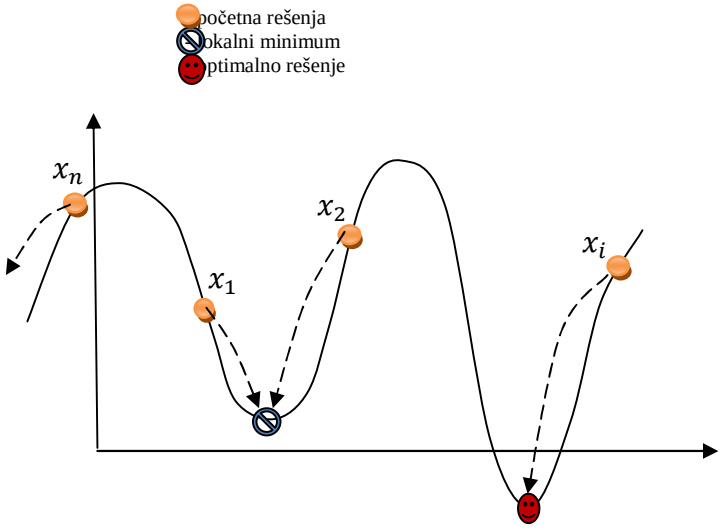
4.3. Višestartno lokalno pretraživanje

Da bi se izbegla zamka lokalnog minimuma najjednostavniji način je da se LS procedura restartuje iz novog početnog rešenja. Ta metoda naziva se *višestartno lokalno pretraživanje* (Multistart Local Search, MLS).
Višestartno lokalno pretraživanje realizuje se ponavljanjem LS procedure svaki put iz novog slučajnog rešenja. Svi dobijeni lokalni minimumi se upoređuju i najbolji među

njima se proglašava za konačno rešenje. Procedura se zaustavlja kada se zadovolji neki izlazni kriterijum (broj ponavljanja, zadato vreme, broj ponavljanja bez poboljšanja i sl.).

Na slici 4.3.1 dat je grafički prikaz ove metode, dok se pseudokod može zapisati u sledećem obliku:

Ponavljaj
1. *Inicijalizacija.* Izabrati početno rešenje x (slučajno).
   Ako je prva iteracija $x_{opt} = x$, $f_{opt} = f(x)$.
2. ***LS***. Primeniti LS proceduru počev od x;
   neka je x' dobijeni lokalni optimum.
3. *Provera rešenja.* **ako** $f(x') < f(x_{opt})$
   onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$.
dok nije zadovoljen kriterijum zaustavljanja.

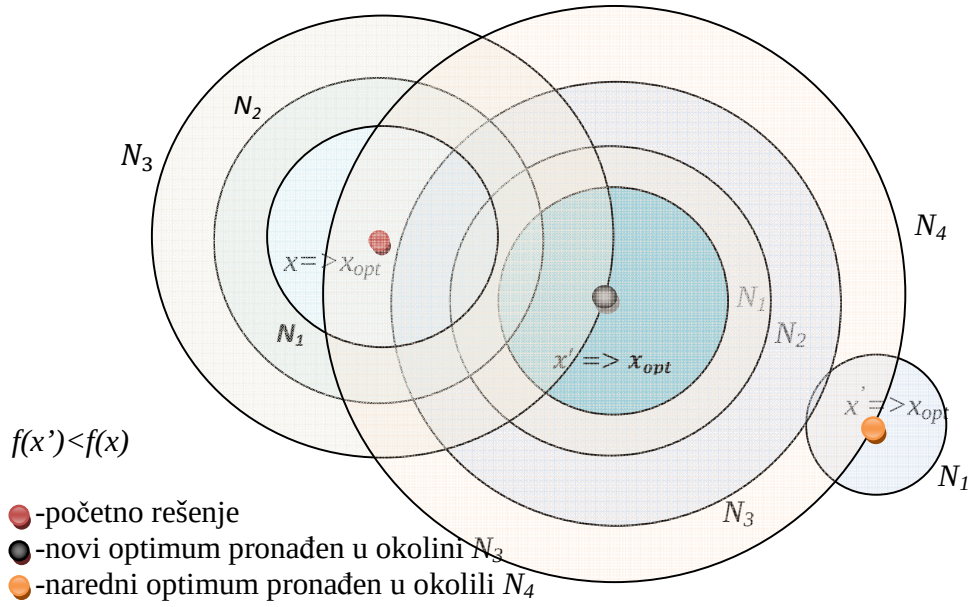


Sl. 4.3.1: Grafički prikaz metode višestartnog lokalnog pretraživanja

MLS predstavlja najjednostavniju metaheurističku metodu baziranu na lokalnom pretraživanju. Na žalost, ova metoda podleže *centralnoj graničnoj teoremi* (central limit theorem) što znači da je pokazano da ovako dobijeni minimumi teže srednjoj vrednosti lokalnih minimuma, a ne globalnom minimumu, tj. optimalnom rešenju.

4.4. Metoda promenljivih okolina

Metodu promenljivih okolina (Variable Neighborhood Search, VNS) predložili su Mladenović i Hansen [19], a prvi put ideja metode izložena je 1995. godine [18]. Osnovna ideja metode je veoma jednostavna: sistematska promena okolina unutar lokalnog pretraživanja. Dakle, neophodno je uvesti više okolina, bilo da se menja metrika u odnosu na koju se definiše okolina, bilo da se povećava rastojanje u odnosu na istu metriku. Metoda je do sada uspešno primenjena na veliki broj problema kombinatorne optimizacije i veštačke inteligencije. Predloženo je nekoliko modifikacija i poboljšanja, uglavnom namenjenih uspešnom rešavanju problema velikih dimenzija [13, 14, 15, 16, 17]. VNS metaheuristika zasnovana je na tri jednostavne činjenice [17]:
(*i*) lokalni minimum u odnosu na jednu okolinu ne mora biti i lokalni minimum u odnosu na neku drugu okolinu;
(*ii*) globalni minimum je lokalni minimum u odnosu na sve okoline;
(*iii*)u većini problema lokalni minimumi u odnosu na razne okoline su međusobno bliski.
Prva činjenica opravdava uvođenje više okolina. Druga činjenica ne garantuje da rešenje koje je lokalni minimum u odnosu na svaku od izabranih okolina predstavlja i globalni minimum, već nešto malo slabije: ako neko rešenje nije lokalni minimum u odnosu na neku okolinu, onda sigurno nije globalni minimum, stoga takođe ima smisla uvođenje više okolina. Kao posledica činjenice 3 (koja je empirijska, a ne teoretska) javlja se potreba detaljnog istraživanja najbliže okoline lokalnog minimuma jer se tu očekuje popravljanje tekućeg najboljeg rešenja.



4.4. Upotreba više okolina u lokalnom pretraživanju

Slika 4.4. ilustruje upotrebu više okolina u rešavanju optimizacionih problema. Mnoge metaheurističke metode bazirane na lokalnom pretraživanju koriste jednu, najviše dve okoline.
Upotreba više okolina postavlja dodatna pitanja na koja treba odgovoriti u okviru konkretne implementacije VNS algoritma [16]:

- *izbor okolina koje će se koristiti*: zavisi od toga koje se metrike mogu definisati i primeniti u datom slučaju;

- *uređenje (redosled) okolina*: definisano je usvojenom metrikom i treba da bude takvo da u odnosu na tu metriku rastojanje među rešenjima raste;
- *veličine okolina*: odnosi se na restrikcije, tj. smanjenja okolina uočavanjem i razmatranjem samo onih suseda koji potencijalno mogu da obezbede poboljšanje;

- *strategije pretraživanja*: koje mogu biti do prvog poboljšanja (FI) ili naći najbolje u okolini (BI);

- *spust i penjanje*: odnos između koraka koji obezbeđuju poboljšanje i ostalih (koji omogućavaju izbegavanje zamke lokalnog minimuma);

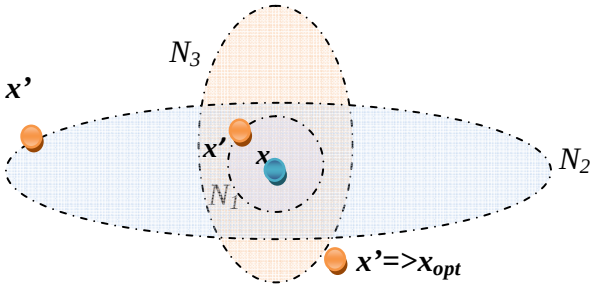
- *smer pretraživanja*: određuje redosled kojim će se pretraživati okoline(kod VNS metode on je definisan parametrima k_{min}, k_{max} i k_{step} o kojima će više reči biti kasnije);

- *intenzifikacija i diversifikacija pretraživanja*: načini na koje će se pretraživanje koncentrisati u okolinama u kojima se očekuje poboljšanje, ili usmeriti pretraga ka novim, neistraženim regionima prostora rešenja čime se povećava šansa nalaženja globalnog minimuma.

Pri konkretnoj implementaciji VNS metode polazi se od tri ranije navedene činjenice ((*i*)*;* (*ii*) i (*iii*)). Ove tri činjenice mogu se iskoristiti na tri različita načina: deterministički, stohastički ili kombinovano. Time se dobijaju tri prve verzije metode promenljivih okolina.

4.4.1 Metoda promenljivog spusta

Kada se koristi deterministička varijanta, dobija se metoda koju nazivamo *metoda promenljivog spusta* (Variable Neighborhood Descent, VND). Ona se sastoji u tome da se izabere k_{max} okolina, N_k , k = 1, 2,....,k_{max}; odredi početno rešenje x i startuje LS procedura u odnosu na svaku od okolina, po redosledu indeksa k, a počev od izabranog rešenja x. Ukolikoje k_{max} = 1, reč je o običnom lokalnom pretraživanju.



Sl. 4.4.1: Upotreba više okolina u metodi promenljivog spusta (VND)

Grafički prikaz upotrebe više okolina u okviru VND metode dat je na slici 4.4.1. Grafičkom ilustracijom je istaknuto da okoline ne moraju biti "ugnežđene", tj. da ne moraju biti indukovane iz iste metrike (ili, ukoliko je reč o definiciji okoline u odnosu na neke zadate transformacije rešenja, okoline se mogu definisati primenom različitih transformacija).
Konkretno, VND znači da se pretražuje okolina $N_1(x)$ dok god u njoj postoji mogućnost popravljanja trenutno najboljeg rešenja. Kada se odredi lokalni minimum x' u odnosu na okolinu N_1, ažurira se tekuće najbolje rešenje(postavi $x_{opt} = x'$), startuje se LS procedura u odnosu na okolinu $N_2(x_{opt})$. Čim dođe do popravke trenutno najboljeg rešenja x_{opt}, pretraživanje se vraća u okolinu N_1 novog najboljeg rešenja. Ukoliko se x_{opt} ne popravi, procedura lokalnog pretraživanja se usmerava na narednu neispitanu okolinu $N_k(x_{opt})$. VND procedura se završava ako je nemoguće popraviti trenutno najbolje rešenje x_{opt} ni u jednoj okolini, tj. ako je x_{opt} lokalni minimum u odnosu na sve okoline $N_1,N_2,...,N_{kmax}$.
Pseudokod VND metode sa osnovnim koracima izgleda ovako:

1. *Inicializacija.* Izabrati početno rešenje x, $x_{opt} = x$, $f_{opt} = f(x)$.
2. **Ponavljaj**
   (a) k=1
   (b) **Ponavljaj**
   i. *LS.* Primeniti LS proceduru počev od x u okolini N_k;
   neka je x' dobijeni lokalni optimum.
   ii. *Provera rešenja.* **ako** $f(x') < f(x_{opt})$
   onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$, $k = 1$
   inače $k = k + 1$
   dok je $k <= k_{max}$
dok ima poboljšanja.

4.4.2 Redukovana metoda promenljivih okolina

U slučaju stohastičke definicije metode koraci su utoliko jednostavniji što ne postoji proces lokalnog pretraživanja. Dakle metoda se sastoji u sistematskoj promeni okolina i izboru jednog slučajnog rešenja u svakoj od okolina. Koraci odlučivanja bazirani su na tom jednom slučajnom rešenju. Metoda koja se na ovaj način dobija naziva se *redukovana metoda promenljivih okolina* (Reduced Variable Neighborhood Search, RVNS).
RVNS je izuzetno korisna kod primera velikih dimenzija jer se izbegava složena i dugotrajna LS procedura. Ova metoda veoma liči na Monte-Carlo metodu, mada je donekle sistematičnija. Dok Monte-Carlo bira slučajno rešenje u celom prostoru pretraživanja, RVNS se u svakom koraku ograničavana na neku, strogo definisanu okolinu. Ipak, obzirom na stepen slučajnosti, najbolji rezultati se postižu kombinacijom ove metode sa nekom drugom varijantom. Na primer, RVNS se koristi za dobijanje početnog rešenja, a zatim se primenjuje neka varijanta koja sistematično pretražuje okoline tako dobijenog početnog rešenja [9].
Pseudokod RVNS metode može se zapisati u sledećem obliku:

1. *Inicializacija.* Izabrati početno rešenje x, $x_{opt}=x$, $f_{opt}=f(x)$.
2. **Ponavljaj**
   (a) $k=1$
   (b) **Ponavljaj**
   i. *Razmrdavanje*. Izabrati slučajno rešenje x' u okolini $N_k(x)$;
   ii. *Provera rešenja*. **ako** $f(x') < f(x_{opt})$
   onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$, $k = 1$
   inače $k = k + 1$
   dok nije $k = k_{max}$

dok nije zadovoljen kriterijum zaustavljanja.

Uobičajeni kriterijum zaustavljanja u ovom slučaju je maksimalni broj iteracija između dva poboljšanja. Naravno, mogu se koristiti i svi ostali navedeni kriterijumi, zavisno od konkretne primene i zahteva koje treba zadovoljiti prilikom rešavanja svakog pojedinačnog optimizacionog zadatka.

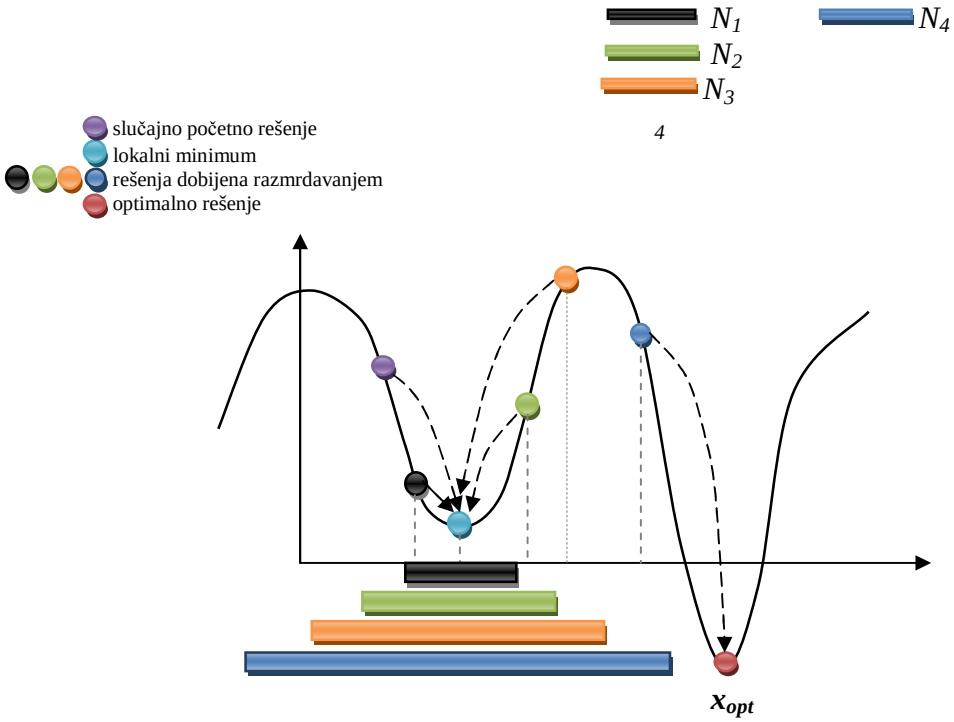
4.4.3 Metoda promenljivih okolina

Kombinacijom prethodna dva principa, tj. sistematskom (determinističkom) promenom okolina, slučajnim izborom početnog rešenja u tekućoj okolini i primenom LS procedure počev od tog, slučajnog rešenja, dobija se *osnovna metoda promenljivih okolina* ((Basic) Variable Neighborhood Search, BVNS). Ovo je najrasprostranjenija varijanta metode promenljivih okolina jer obezbeđuje više preduslova za dobijanje kvalitetnijih konačnih rešenja.
Osnovni koraci VNS metode sadržani su u petlji u okviru koje se: menja indeks okoline k, određuje slučajno rešenje iz k-okoline, izvršava procedura lokalnog pretraživanja i proverava kvalitet dobijenog lokalnog minimuma. Ovi koraci se ponavljaju sve dok ne bude zadovoljen neki kriterijum zaustavljanja. Početno rešenje u okolini k generiše se na slučajan način kako bi se obezbedilo pretraživanje različitih regiona prilikom sledećeg razmatranja okoline k. Okoline se mogu razlikovati po osnovu rastojanja (broja transformacija) ili po osnovu metrike (vrste transformacija). Važno je napomenuti da okoline za razmrdavanje (izbor slučajnog rešenja) i lokalno pretraživanje *ne moraju biti istog tipa*.
Opisani koraci mogu se ilustrovati pseudokodom na sledeći način:

Inicializacija. Izabrati početno rešenje $x \in X$;
definisati kriterijum zaustavljanja; STOP = 0.

Ponavljaj
1. $k = 1$;
2. **Ponavljaj**
   (a) *Razmrdavanje.* Generisati slučajno rešenje x' u k-toj okolini od x, ($x' \in N_k(x)$);
   (b) *LS.* Primeniti neku proceduru lokalnog pretraživanja počev od x' ; označiti sa x'' dobijeni lokalni minimum;
   (c) *Provera rešenja.* Ako je lokalni minimum bolji od trenutnog minimuma, preći u to rešenje ($x = x''$); nastaviti od novog početnog rešenja u okolini N_1 ($k = 1$); inače preći u sledeću okolinu, tj. $k = k + 1$.
   (d) *Provera završetka.* Ako je zadovoljen kriterijum zaustavljanja, STOP = 1.
   dok nije $k == k_{max}$ ili STOP == 1;
dok nije STOP == 1;

Slika 4.4.3 sadrži mogući grafički prikaz primene VNS metode na minimizaciju funkcije na pravoj [8].



4.4.3: Ilustracija izvršavanja VNS metode za funkciju jedne promenljive

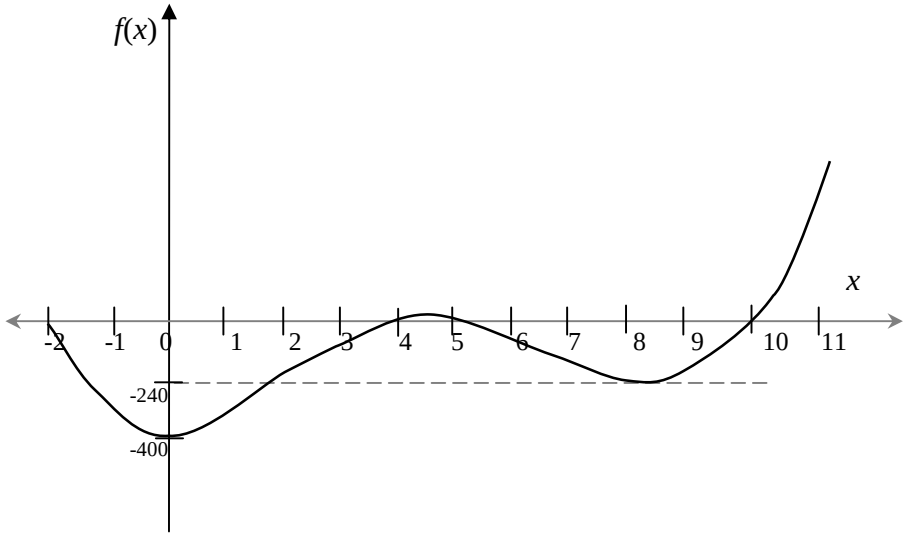
Metoda VNS u toku svog izvršavanja forsira "spust", uvek se prelazi u bolje rešenje i to u prvo na koje se naiđe, tj. metoda je *First Improvement* (FI) karaktera. Osnovni parametar VNS metode je k_{max} (maksimalan broj okolina) [15, 19]. Kriterijum zaustavljanja može biti maksimalno dozvoljeno vreme izvršavanja, maksimalni broj iteracija, tj. koliko puta se dostigne k_{max}, ili broj iteracija između dve popravke globalno najboljeg rešenja. Osim već pomenute kombinacije VNS metode sa RVNS, moguće je kombinovati i VNS sa VND tako što bi se LS procedura zamenila sa VND. Pri tome okoline u kojima se bira početno rešenje i okoline u okviru kojih se izvršava VND procedura mogu biti definisane na različite načine. Za ilustraciju primene VNS metode može poslužiti sledeći primer.

Primer. Naći minimum funkcije $f(x)=x^4-17x^3+72x^2+20x-400$ na intervalu [-2,11], pri čemu je x celobrojno. Grafik funkcije dat je na slici, a relevantne vrednosti funkcije prikazane su tabelarno. Evidentno je da ova funkcija ima lokalni i globalni minimum, te stoga postoji opasnost da se procedura lokalnog pretraživanja zaustavi u lokalnom minimumu. Dakle, neophodno je primeniti neku od metaheurističkih metoda za rešavanje ovog zadatka.
Da bi se primenila VNS metoda, najprirodnije je rešenje definisati kao argument funkcije (x), a niz okolina $N_k(x) = \{x-k, x+k\}$, tj. par tačaka na rastojanju k od datog rešenja x.

Procedura lokalnog pretraživanja izvršava se u $N_1(x)$, dok se za razmrdavanje koriste sve raspoložive okoline. Neka se početno rešenje bira slučajno i neka je ono $x_0 = 6$. Vrednost funkcije u toj tački je $f(x_0)=f(6)=-64$. Procedura inicijalnog lokalnog pretraživanja zaustavlja se u lokalnom minimumu $x_{min}=8$. Naime, polazeći od x_0 i njegove okoline $N_1=\{5,7\}$, u prvoj iteraciji LS procedura će izabrati tačku $x_1= 7$ jer je tu vrednost funkcije manja. Zatim će se, u drugoj iteraciji, LS pomeriti u tačku x_1 i u odnosu na njenu okolinu $N_1=\{6,8\}$ izabrati novo, poboljšano rešenje $x_2 = 8$, i $f(8)=-240$. U narednoj iteraciji, LS procedura se zaustavlja jer u okolini $N_1=\{7,9\}$ nema poboljšanja.

Izvršavanje VNS metode nastavlja se razmrdavanjem u okolini $N_1(x_{min})$ pri čemu se na slučajan način bira jedna od tačaka iz te okoline. Bilo koji izbor vraća LS proceduru u lokalni minimum $x_{min} = 8$. Stoga se povećava indeks okoline za razmrdavanje i vrši slučajan izbor tačke iz okoline $N_2(x_{min})=\{6,10\}$. Kao i u prethodnom koraku, procedura lokalnog pretraživanja iz bilo koje od ovih dveju tačaka zaustaviće se u već određenom lokalnom minimumu. Okolina $N_3(x_{min})=\{5,11\}$, takođe navodi LS proceduru, koja od nje započinje svoje izvršavanje, opet u x_{min}. Na redu je, dakle, okolina $N_4(x_{min})=\{4\}$, koja u ovom primeru sadrži samo jednu tačku. Polazeći od ove tačke, lokalno pretraživanje ne uspeva da pronađe globalni minimum ni u narednoj okolini $N_5(x_{min})=\{3\}$. Tek u okolini $N_6(x_{min})=\{2\}$ se dobija prvo poboljšanje i vrednost funkcije pada na $f(1)=-324$. Sada LS procedura započinje pretraživanje u N_1 okolini novog lokalnog minimuma, tačke $x_{min}=1$, a to je $N_1(x_{min)} =\{0,2\}$.U tački $x = 0$ procedura LS pronalazi globalni minimum koji iznosi $f(0)=-400$. Naravno, nije uvek ovako lako utvrditi da je to optimalno rešenje i stoga se, u složenijim primerima, izvršavanje VNS metode nastavlja sve dok se ne zadovolji neki kriterijum zaustavljanja.

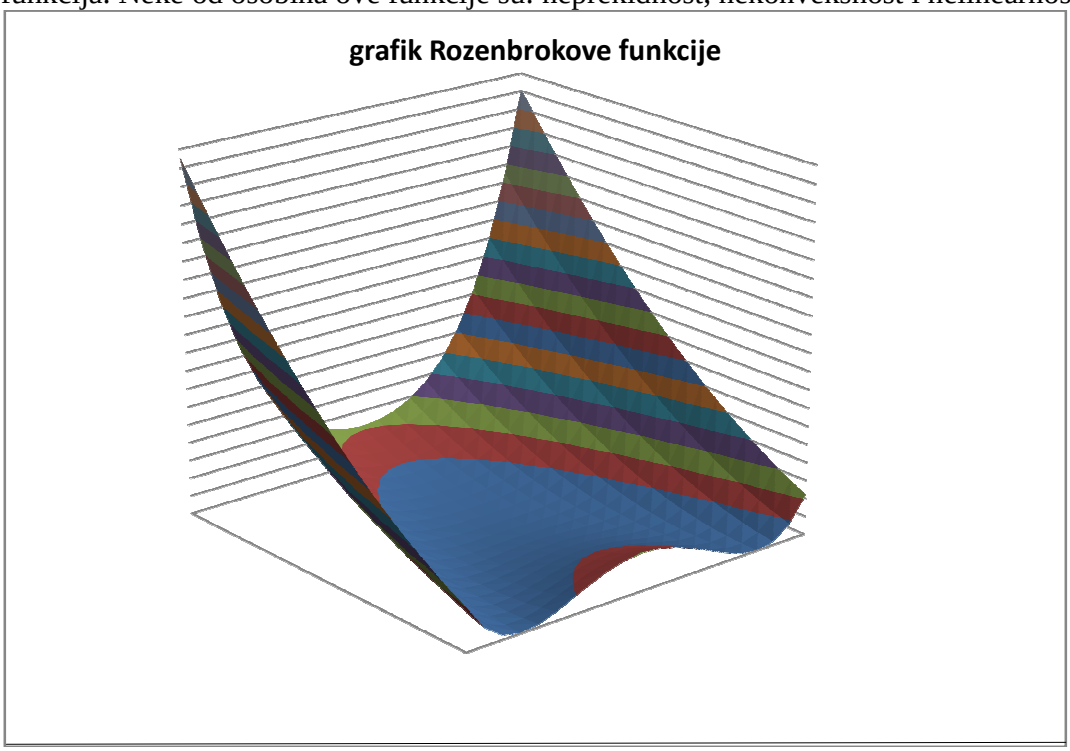
x	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**
$f(x)$	0	-330	-400	-324	-192	-70	0
x	**5**	**6**	**7**	**8**	**9**	**10**	**11**
$f(x)$	0	-64	-108	-240	-220	0	546



Koliko je bitno da se dobro definiše okolina nekog rešenja u VNS metodi, biće razmotreno na funkciji dve promenljive poznatijoj kao Rozenbrokova funkcija čiji je grafik prikazan na slici. Analitički oblik Rozenbrokove funkcije i domen izgledaju ovako:

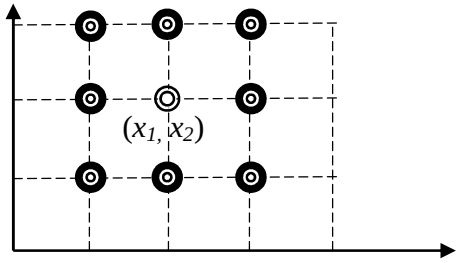
$$f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \quad \text{pri čemu je } x_i \quad [-2.048, 2.048]$$

Ova funkcija spada u grupu *De Jongovih test funkcija* koje se koriste za testiranje genetskih algoritama i omogućavaju poređenje algoritama na skupu raznovrsnih test-funkcija. Neke od osobina ove funkcije su: neprekidnost, nekonveksnost i nelinearnost.



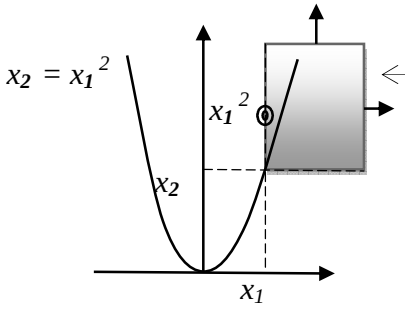
Traži se minimum Rozenbrokove funkcije. To i nije tako veliki problem ako bi se ograničio na celobrojna rešenja, jer je samim tim olakšano definisanje okolina za pretraživanje. Domen čine tačke sa realnim koordinatama, sa tačnošću od tri decimale, što ukupno iznosi 2^{22} dopustivih rešenja. U ovom primeru će biti prikazan algoritam VNS metode primenjen na restrikovanom domenu koji pokriva samo 0.01% prostora dopustivih rešenja. Zato rešavanje problema počinje postavljanjem ekvidistantne mreže na domenu funkcije sa korakom $k = 0{,}204$ da bi se od leve do desne krajnje tačke domena, po obe promenljive, uzela u obzir po 21 tačka, tačnije ukupno 441 tačka na tako suženom domenu. Preciznije, ako je x_1 koordinata neke tačke x, sledeća tačka može imati za vrednost koordinate x_1 jednu od vrednosti: $x_1' = x_1 + k*n$ ili $x_1' = x_1 + k*n$ n pripada intervalu [1,20] i x_1' pripada [-2.048,2.048], gde će n uzimati one vrednosti koje omogućavaju novim tačkama da ostanu unutar prostora rešenja. U ovom primeru, zbog ovakvog izbora koraka k, najveća vrednost po obe koordinare biće 2.032.

Okolina za lokalno pretraživanje će biti ujedno i N_1 okolina za razmrdavanje. Ona se sastoji iz 8 tačaka koje okružuju bilo koje početno rešenje, iz gore definisane ekvidistantne mreže: $N_1(x_1, x_2)$={ $(x_1 + k, x_2 + k)$, $(x_1 , x_2 + k)$, $(x_1 - k, x_2 + k)$, $(x_1 - k, x_2)$, $(x_1 - k, x_2 - k)$, $(x_1 , x_2 - k)$, $(x_1 + k , x_2 - k)$, $(x_1 + k , x_2)$. Grafički to izgleda ovako:



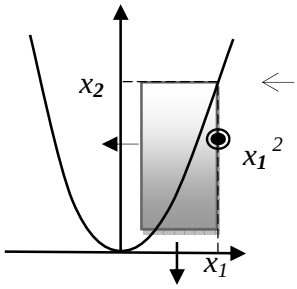
Iz analitičkog oblika Rozenbrokove funkcije, primenom pomenute osobine *intenzifikacija i diversifikacija pretraživanja*, može se zaključiti da globalni minimum treba očekivati u delu domena gde je x_1^2 približno jednak x_2, tj. da se njene minimalne vrednosti kreću po oblasti domena u kojoj važi $x_1^2 \sim x_2$ ili u najboljem slučaju $x_2 = x_1^2$
Na osnovu ovog zapažanja, preostale okoline za razmrdavanje mogu se takođe suziti na sledeći način:

- *ako je u nekoj tački (x_1, x_2), gde je LS procedura stala, $x_1^2 > x_2$, onda će se vraćanje pretraživanja u deo prostora gde se očekuje poboljšanje rešenja postići pretraživanjem samo dela prostora ˝gore desno˝ od te tačke. To znači da će koordinate tačaka koje će se naći u okolinama za razmrdavanje biti oblika:*
  $x_1 + n*k$ *i* $x_2 + n*k$ *. Primer je prikazan na slici.*



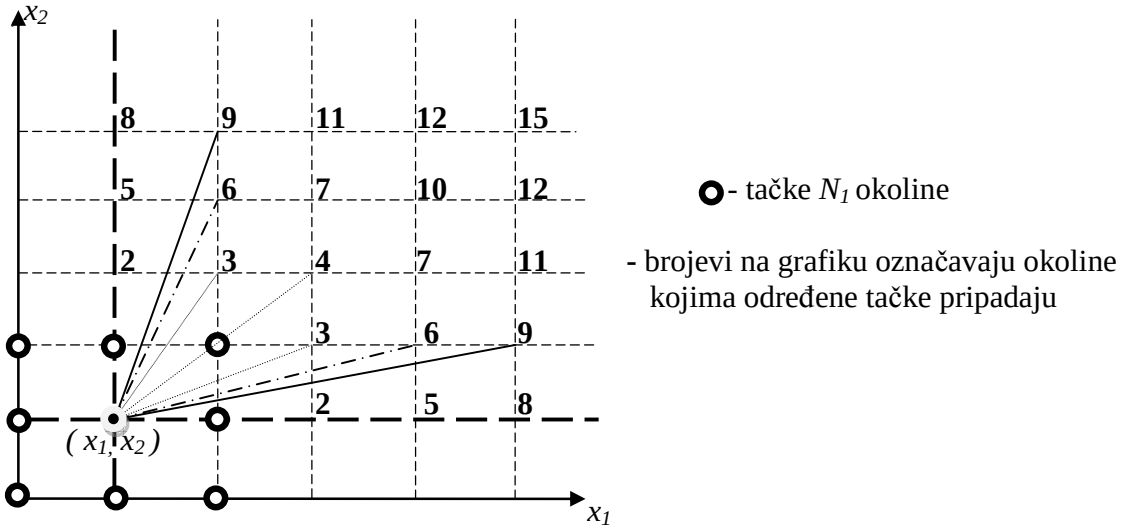
⇐deo prostora ˝gore desno˝ u kome se nalaze sve okoline za razmrdavanje

- *ako je u tački (x_1, x_2), $x_1^2 < x_2$, onda će se sve okoline za razmrdavanje ograničiti na deo prostora ˝dole levo˝ od te tačke. Koordinate tako nastalih tačaka biće* $x_1 - n*k$ *i* $x_2 - n*k$.



⇐deo prostora ˝dole levo˝ u kome se nalaze sve okoline za razmrdavanje

Sada se mogu definisati okoline indeksa većeg od 1. U ovom primeru, u nekim okolinama će se u zavisnosti od njihovog indeksa naći različit broj tačaka. U nekima će biti samo jedna tačka a u nekima četiri. Okoline se formiraju tako što se udaljava od rešenja dobijenog lokalnim pretraživanjem: u okolini N_1 se nalaze tačke koje su na rastojanju k i $\sqrt{k}$ od trenutno najboljeg rešenja. U okolini N_2 se nalaze tačke koje su na sledećem većem rastojanju od trenutnog rešenja a to je $2k$, u okolini N_3 tačke koje su na rastojanju $\sqrt{5}k$, u okolini N_4 tačke koje su na rastojanju $\sqrt{8}k$, u N_5 one koje su na rastojanju $3k$ i tako redom. Grafički prikaz govori nešto više o tome.



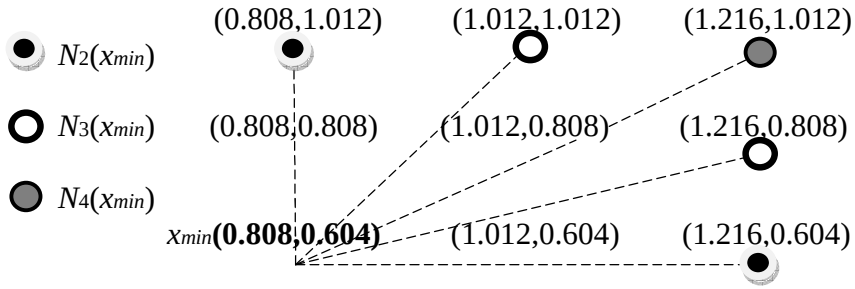
Grafik okolina tačke (x_1, x_2) u slučaju $x_1^2 > x_2$

To znači da u slučaju kada je $x_1^2 > x_2$, okolini N_2 pripadaju tačke **(x_1 +2*k*, x_2)** i **(x_1 , x_2 +2*k*).** Okolini N_3 pripadaju tačke **(x_1 +2*k*, x_2 +*k*)** i **(x_1 +*k*, x_2 +2*k*),** okolini N_4 samo tačka **(x_1 +2*k*, x_2 +2*k*)** i tako dalje**.** Slično se određuju tačke koje pripadaju okolinama u slučaju kada je $x_1^2 < x_2$.

Neka se početno rešenje ovog problema bira slučajno i neka je to (-1.232,-0.416). Vrednost Rozenbrokove funkcije u ovoj tački iznosi f(-1.232,-0.416) = 378.949. Procedura lokalnog pretraživanja polazi od te tačke i nakon 10 iteracija se zaustavlja u lokalnom minimumu x_{min}=(0.808,0.604) koji u toj tački iznosi 0.276. Razmrdavanjem u N_1 okolini te tačke, gde se slučajnim izborom bira jedna od 8 tačaka iz te okoline, samo tri od tih osam tačaka, dovodi LS proceduru do globalnog minimuma (1.012,1.012) koji iznosi 0.015. Kako je verovatnoća izbora te tačke u okolini N_1 samo 37,5%, predpostaviće se da izbor nije pao na te tačke i da se razmrdavanjem u toj okolini ne nalazi bolje rešenje. Pokušaće se pronaći okolina iz koje se sigurnim izborom bilo koje tačke iz te okoline dolazi do globalnog minimuma.

Sada na red dolazi izbor okolina za pretraživanje, na osnovu uslova: $x_1^2 > x_2$ ili $x_1^2 < x_2$. Kako je u nađenom lokalnom minimumu ispunjen prvi uslov, $x_1^2 > x_2$, pretraživaće se okoline koje se nalaze ˝gore desno˝od pronađenog lokalnog minimuma. Izvršavanje VNS metode nastavlja se razmrdavanjem u okolini $N_2(x_{min})$ pri čemu se na slučajan način bira

jedna od tačaka iz te okoline. U toj okolini se u ovom slučaju nalaze tačke (1.216,0.604) i (0.808,1.012). I u ovoj okolini slučajnim izborom jedne od dve tačke, tačnije tačke (0.808,1.012), lokalno pretraživanje pronalazi već pomenuti globalni minimum. Verovatnoća izbora ove tačke prilikom razmrdavanja u okolini $N_2(x_{min})$ je znatno veća nego u predhodnom slučaju i iznosi 50%. Ponovo će se predpostaviti da izbor nije pao na tu tačku i da nema poboljšanja, i nastaviti razmrdavanje u okolini N_3. Izborom jedne od tačaka iz N_3 okoline a to su (1.012,1.012) i (1.216,0.808), LS procedura takođe pronalazi globalni minimum, sa verovatnoćom 50%. Nastavlja se razmrdavanje u sledećoj okolini N_4. U toj okolini se nalazi samo jedna tačka i to (1.216,1.012) koja LS proceduru sigurno vodi do globalnog minimuma $\boldsymbol{x}$ =(**1.012,1.012)** i on iznosi $\boldsymbol{f}$(**1.012,1.012)=0.015**. To je dosta dobro rešenje, obzirom da je pretraživano samo 0.01% prostora mogućih rešenja kao i da je globalni minimum Rozenbrokove funkcije u tački (**1,1**) i iznosi f(**1,1**)= **0**.



Iz primera se vidi da se do globalnog minimuma moglo doći i iz prve tri okoline, a iz četvrte sa verovatnoćom od 100%. Takođe treba napomenuti da se i bez restrikcija okolina za razmrdavanje na ˝gore desno˝ i ˝dole levo˝ dolazi do globalnog minimuma ali se ne može sa sigurnošću na ovom primeru odrediti u kojoj iteraciji će se to desiti.

Postoje brojne modifikacije i proširenja originalno definisane VNS metode[74]. One su uglavnom namenjene za rešavanje složenih problema i problema velikih dimenzija. Nekoliko jednostavnih modifikacija su:

1) prihvatanje lošijih rešenja sa nekom zadatom verovatnoćom, čime metoda postaje i penjanje i spust;

2) razmatranje svih k_{max} okolina pre nego što se donese odluka u koje rešenje preći, tj. implementacija *Best Improvement* (BI) verzije metode;

3) umesto jednog početnog rešenja u okolini k, generiše se b slučajnih rešenja (b je novi parametar) i bira se najbolje među njima za početno rešenje LS procedure;

4) mogu se uvesti i parametri k_{min} i k_{step} sa značenjem da se ne polazi iz okoline k = 1, već k_{min} i da se indeks okoline ne uvećava za 1 nego za k_{step}. Ukoliko je priroda problema takva da postoji veliki broj rešenja sa istom vrednošću funkcije cilja, tj. postoje tzv. *platoi*, može se kao parametar uvesti i $p_{plateaux}$ {verovatnoća prelaska u novo rešenje koje ima istu, trenutno minimalnu vrednost funkcije cilja [14, 15, 16].

Neke malo složenije modifikacije osnovnih verzija VNS metode bile bi:

(*a*) *Metoda promenljivih okolina sa dekompozicijom* (Variable Neighborhood Decomposition Search, VNDS);

(*b*) *Ukošeno pretraživanje promenljivom okolinom* (Skewed VNS, SVNS) i druge o čemu se više detalja može pronaći u [17].

4.4.4. Metoda promenljivih okolina sa dekompozicijom (VNDS)

U praksi, prilikom rešavanja problema velikih dimenzija, često se raspolaže sa ograničenim brojem podataka(alata) za rešavanje tog problema, i oni uglavnom nisu dovoljni za njegovo rešavanje u realnom vremenu. Čak šta više, kada se heuristike primene na probleme jako velikih dimenzija, njihove slabosti postanu očigledne. Zbog toga se usavršavanje ovih metoda smatra veoma poželjnim [3].
Metoda promenljivih okolina sa dekompozicijom (VNDS) je jedna od modifikacija osnovne VNS metode, koja se od nje razlikuje samo u koraku 2(b) pseudokoda :

- umesto primene lokalnog pretraživanja u čitavom prostoru potencijalnih rešenja S(počevši od $x' \in N_k(x)$), u VNDS metodi rešava se u svakoj iteraciji podproblem u nekom subprostoru $V_k \subseteq N_k(x)$ gde je $x' \in V_k$. Kada se u tom koraku za lokalno pretraživanje takođe koristi VNS, dobijamo dvostruku VNS šemu (two-level VNS scheme). Drugim rečima, rešenja traženog problema mogu se rastaviti na komponente. Tada, u svakom koraku VNS-a možemo svakom rešenju pridružiti tačno k promenjivih komponenti. Struktura susedstva N_k izmenjenog algoritma je jasno definisana, a prilikom lokalnom pretraživanja menjamo samo promenjive komponente.

Na slici 4.4.4. je prikazan pseudokod VNDS metode.

Inicializacija. Izabrati početno rešenje $x \in X$;
definisati kriterijum zaustavljanja; STOP = 0.

Ponavljaj
1. $k = 1$;
2. **Ponavljaj**
   (a) *Razmrdavanje.* Generisati slučajno rešenje x' u k-toj okolini od x, ($x' \in N_k(x)$) ;
   drugačije rečeno, neka y bude k promenljivih komponenti sadržanih u x' ali ne i u x ($y = x' \setminus x$)
   (b) *LS.* Primeniti neku proceduru lokalnog pretraživanja
   počev od y ; označiti sa y' najbolje dobijeno rešenje a sa x'' odgovarajuće rešenje iz celog prostora S ($x'' = (x' \setminus y) \cup y'$)
   (c) *Provera rešenja.* Ako je lokalni minimum bolji od
   trenutnog minimuma, preći u to rešenje ($x = x''$);
   nastaviti od novog početnog rešenja u okolini N_1 ($k = 1$);
   inače preći u sledeću okolinu, tj. $k = k + 1$.
   (d) *Provera završetka.* Ako je zadovoljen kriterijum
   zaustavljanja, STOP = 1.
   dok nije $k == k_{max}$ ili STOP == 1;
dok nije STOP == 1;

4.4.5. Ukošeno pretraživanje promenljivom okolinom (SVNS)

Ova metoda izmešta pretraživanje prostora u novu okolinu daleko od trenutno najboljeg rešenja. Naime, kada se pretraživanjem velikog regiona konačno pronađe najbolje rešenje, neophodno je da se pretraživanje prostora nastavi prilično daleko da bi se dobilo bolje rešenje od trenutno najboljeg poboljšanog rešenja. Rešenja dobijena slučajnim izborom iz dalekih okolina mogu se znatno razlikovati od trenutno najboljeg rešenja, pa bi se samim tim VNS metoda mogla pretvoriti u višestartno lokalno pretraživanje(MLS), koje nije tako efikasno. Dakle, neka vrsta procene za udaljenost od trenutno najboljeg rešenja se mora napraviti, pa je tako nastala Skewed VNS metoda [3].

SVNS koristi funkciju $r(x,x'')$ da izmeri rastojanje između trenutno najboljeg rešenja i pronađenog lokalnog minimuma. Rastojanje koje se koristi za definisanje okoline N_k, može takođe biti iskorišćeno kao rastojanje kojim će biti definisana funkcija r. Parametar a mora biti odabran tako da omogući pretraživanje okolina daleko od x kada je $f(x'')$ veće od $f(x)$ ali ne previše daleko (jer u tom slučaju x može ispasti iz jedne od tih okolina i izgubiti se kao rešenje). Parametar a se bira eksperimentalno npr. da bi se izbegla mala pomeranja od x do bliskih rešenja(kada je $r(x,x'')$ malo) može se izabrati velika vrednost za parametar a.

Sl. 4.4.5. pseudokod Skewed VNS metode

Inicializacija. Izabrati početno rešenje $x \in X$; izračunati njegovu vrednost $f(x)$
$x = x_{opt}$, $f(x)=f_{opt}$; definisati kriterijum zaustavljanja; STOP = 0.

Ponavljaj
1. $k = 1$;
2. **Ponavljaj**
   (a) *Razmrdavanje.* Generisati slučajno rešenje x'
   u k-toj okolini od x, ($x' \in N_k(x)$);
   (b) *LS.* Primeniti neku proceduru lokalnog pretraživanja
   počev od x'; označiti sa x'' dobijeni lokalni minimum;
   (c) *Provera rešenja.* **Ako** je $f(x'') < f_{opt}$ onda $f(x'')= f_{opt}$ i $x''= x_{opt}$;
   (d) **Ako** je $f(x'') - ar(x,x'') < f(x)$ onda $x''=x$ i nastaviti od novog početnog
   rešenja u okolini N_1 ($k = 1$);
   inače preći u sledeću okolinu, tj. $k = k + 1$.
   (e) *Provera završetka.* Ako je zadovoljen kriterijum
   zaustavljanja, STOP = 1.
dok nije $k == k_{max}$ ili STOP == 1;
dok nije STOP == 1;

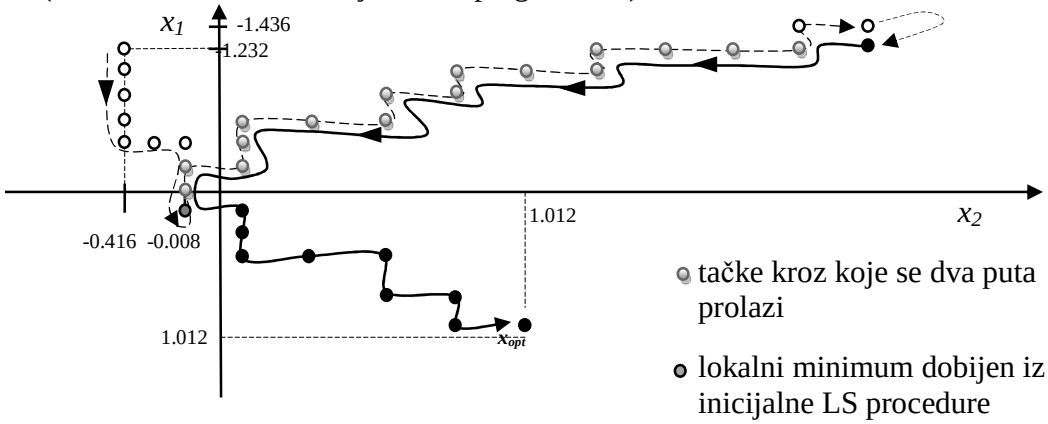
4.5. Tabu pretraživanje

Tabu pretraživanje (Tabu Search, TS) metaheuristika opisana je detaljno u knjizi Glovera i Lagune [11]. Metodu su nezavisno razvili Glover [10] i Hansen [12] koji je formulisao heuristički princip najbržeg spusta/najsporijeg uspona (steepest descent/mildest ascent).
Kao i MLS i VNS, i TS metoda zasniva proces pretraživanja na okolini tekućeg minimalnog rešenja. Kada se u toku pretraživanja naiđe na lokalni minimum, potrebno je definisati kriterijum izlaska iz njega. Taj kriterijum je kod TS metode zasnovan na upotrebi memorija. Za razliku od ostalih navedenih metoda koje čuvaju samo trenutno najbolje rešenje i odgovarajuću (trenutno minimalnu) vrednost funkcije cilja, TS metoda pamti kretanje preko rešenja posećenih u nekoliko prethodnih iteracija. Te informacije se koriste za izbor narednog rešenja kao i za modifikaciju definicije okoline u smislu izbacivanja nekih "zabranjenih" rešenja. Obzirom na to, jasno je da okolina $N(x)$ rešenja x varira od iteracije do iteracije, te se stoga TS ubraja u procedure koje se nazivaju *tehnike dinamičkog pretraživanja okoline*. Preciznije, opis izvršavanja osnovne verzije TS metode sastoji se u sledećem: ova metoda sistematski prati proces minimizacije zasnovan na LS proceduri sve dok se ne dostigne lokalni minimum. Zatim se taj lokalni minimum isključuje iz okoline za pretraživanje i traži minimalno rešenje u tako modifikovanoj okolini. Isključena rešenja se pamte u listi nazvanoj *tabu lista* (*TL*) koja predstavlja zabranjena rešenja u toku nekoliko narednih iteracija (definisanih dužinom *TL*). Po isteku zabrane, ova rešenja vraćaju se u okolinu, a neka druga postaju zabranjena. Na osnovu dosada izloženog može se opisati pseudokod osnovne verzije TS metode.

Inicijalizacija. Izabrati početno rešenje $x \in X$;
postaviti $x_{opt} = x$, $TL = \emptyset$; izabrati kriterijum zaustavljanja.
Ponavljaj
1. *LS.* Naći $x' \in N(x) \setminus TL$ za koje f ima minimalnu vrednost.
2. *Provera rešenja.* Ako je $f(x') < f(x_{opt})$ postaviti $x_{opt} = x'$
Postaviti $x = x'$.
3. *Ažuriranje TL .* $TL = TL \cup \{x'\}$; **ako** $|TL| > NTABU$
onda $TL = TL \setminus \{x^t\}$
pri čemu je x^t rešenje koje je najduže u tabu listi *TL* (FIFO princip).
dok nije zadovoljen kriterijum zaustavljanja.

Ako se koristi tabu lista promenljive dužine, kao prvi korak u pseudokodu TS metode dodaje se:
1. *Ažurirati* $|TL|$ *(dužinu tabu liste TL).* Generisati slučajan broj iz intervala(1,*NTABU*) koji predstavlja dužinu *TL* za tekuću iteraciju.
Kriterijumi zaustavljanja su uobičajeni, već pominjani: maksimalan broj iteracija, maksimalan broj iteracija između dve popravke najboljeg rešenja ili maksimalno dozvoljeno vreme izvršavanja metode.
Suština TS metode je u tome da se nakon dostignutog lokalnog minimuma posećuju rešenja koja ne vode poboljšanju vrednosti funkcije cilja sa idejom da se napusti okolina

tog lokalnog minimuma i omogući potraga za novim lokalnim minimumima od kojih bi neki mogao biti i globalni. Da bi se sprečilo ponovno posećivanje nekog lokalnog minimuma, tzv. cikliranje, koristi se tabulista (*TL*) u kojoj se čuvaju zabranjena rešenja.
Primer minimizacije Rozenbrokove funkcije koji je već razmatran, poslužiće i kao primer za tabu pretraživanje. Primenom TS metode, počev od inicijalnog rešenja (0.400,1.420), minimizacija Rozenbrokove funkcije bi se odvijala na sledeći način. Kako TS koristi samo okolinu N_1 , u ovoj metodi ćemo koristiti malo drugačije definisanu tu okolinu u odnosu na njenu definiciju u VNS metodi. Neka N_1 okolina tačke (x_1, x_2) sadrži 4 tačke:$(x_1, x_2+k), (x_1, x_2-k), (x_1+k, x_2), (x_1-k, x_2),$ i to na istoj ekvidistantnoj mreži i sa istim korakom *k* kao i u VNS metodi. Počevši od inicijalnog rešenja (0.400,1.420), LS procedura se zaustavlja u x_{min}=(1.216,1.420). To rešenje ubacuje se u tabu listu, i u njegovoj okolini određuje se sledeće najbolje, a to je x_1=(1.216,1.624). Novo rešenje je lošije od tekućeg minimuma, ali se ipak ono bira zato što nije zabranjeno(nije u tabu listi). Naravno, rešenje x_1 se takođe ubacuje u tabu listu i u njegovoj okolini se određuje sledeće najbolje a to je x_2 =(1.216,1.828). Sada iz tabu liste, koja je za ovaj primer dužine dva, izlazi x_{min} a u listu zabranjenih rešenja ulazi x_2. Tako se sada u tabu listi nalaze rešenja x_1 i x_2 .Ovaj sistem punjenja i pražnjenja tabu liste, gde prvo rešenje koje ulazi u listu prvo i izlazi iz nje kad se lista popuni, naziva se FIFO(eng. *first-in first-out*). LS procedura nastavlja pretraživanje u N_1 okolini tačke x_2. Od dozvoljenih rešenja iz te okoline najbolje je x_3 =(1.420,1.828). Sada to rešenje ulazi u tabu listu, iz liste izlazi x_1, i nastavlja se LS procedura u okolini tačke x_3 .U toj okolini, u tački x_4 =(1.42,2.032) dostiže se novi lokalni minimum koji je bolji od trenutnog minimuma. Ubacivanjem tačke x_4 u tabu listu, postupak se nastavlja. Kako je uvek tačka koja ima veću vrednost po prvoj ili drugoj koordinati od tekućeg rešenja zabranjena, to omogućava da se pretraga izvuče iz zamke lokalnog minimuma. Posle 7 iteracija dostiže se optimalno rešenje u tački (1.012,1.012).
Treba napomenuti da je još efikasniji primer bega iz lokalnog minimuma primenom TS metode, primer Rozenbrokove funkcije ali sa inicijalnim rešenjem koje je korišćeno u VNS metodi. To je tačka (-1.232,-0.416) od koje se započeta LS procedura zaustavlja u lokalnom minimumu (0.196,-0.008) i njegova vrednost je 0.862. TS metoda koja započinje od ovog inicijalnog rešenja, njeno dalje kretanje kroz rešenja koja *ne vode* poboljšanju vrednosti funkcije cilja kao i vraćanje u globalni minimum, prikazani su na slici (ose koordinata su zamenjene radi preglednosti).



Zabranjena rešenja se mogu pamtiti pomoću nekih atributa, kako bi se smanjila memorija potrebna za njihovo smeštanje. Stoga je uobičajeno reći da se u *TL* pamte *zabranjeni pokreti*. Uobičajeno je da se koristi i po nekoliko tabu lista i u tom slučaju, pokret ima "tabu status" ako se nalazi u svim listama, u suprotnom, pokret je dozvoljen. Osnovna uloga tabu liste je izbegavanje zamke lokalnog minimuma i sprečavanje cikliranja u okolini nekog rešenja.
Obzirom na činjenicu da se u tabu listi čuvaju atributi rešenja (zbog uštede u prostoru), moguće je da to dovede do znatne restrikcije u prostoru pretraživanja, tj. do zabrane svih pokreta koji imaju iste atribute kao i onaj koji se zaista zabranjuje, a samim tim do sprečavanja da se poseti mnogo više rešenja, a ne samo ona dobijena u poslednjih I*TL*I iteracija. Sprečavanje cikliranja, takođe, nije u potpunosti zagarantovano i zavisi od dužine tabu liste. To je parametar koji treba pažljivo birati (ukoliko je prevelik, diversifikacija pretraživanja je veća i može dovesi do tzv. *slučajnog kretanja* (*random walk*); ukoliko je dužina *TL* isuviše mala, veća je mogućnost cikliranja). Korisnik treba da bude svestan tih činjenica i da ih na najpovoljniji način uključi u implementaciju metode [11].
Da bi se prevazišli ovi nedostaci uvodi se mogućnost ukidanja tabu statusa nekom pokretu. Najjednostavniji način za to je korišćenje tabu liste promenljive dužine o čemu je već bilo reči. Sistematičniji način je definicija tzv. praga zadovoljivosti (aspiration level) kojim se opisuju "dobra" rešenja. Najjednostavniji primer je da se skida tabu status pokretu koji vodi u novo najbolje rešenje (u odnosu na funkciju cilja).
Još jedna specifičnost TS metode je upotreba različitih vrsta memorije. Postoje tzv.*kratkoročne memorije*, zatim *srednjoročne* i *dugoročne* memorije. Kratkoročne memorije se koriste prilikom formiranja tabu lista i služe za izbegavanje cikliranja pretraživanja u okolini nekog "dobrog" rešenja. Srednjoročne memorije imaju zadatak da obezbede intenzifikaciju pretraživanja u nekom regionu u kome se nalazi trenutno najbolje rešenje, tzv. proces intenzifikacije, dok je uloga dugoročnih memorija diversifikacija, tj. prenošenje procesa pretraživanja u još neistražene regione prostora dopustivih rešenja. Da bi se to obezbedilo, pamte se neki definisani "atributi" posećenih rešenja. U procesu intenzifikacije najčešće se biraju rešenja koja imaju veći broj promenljivih ili "atributa" zajedničkih sa trenutno najboljim rešenjem, dok se u procesu diversifikacije, favorizuju rešenja kod kojih su ti atributi različiti.
Način definicije okoline, izbora atributa koji se prate i pamte, kao i ostalih parametara TS metode (dužina tabu liste, izbor kriterijuma zaustavljanja, definicija praga zadovoljivosti), zavise od konkretnog problema koji se rešava kao i od same implementacije metode za neki dati problem. Opšta pravila metode dovoljno su fleksibilna da omogućavaju efikasne implementacije za najraznovrsnije probleme. Ukoliko se u implementaciju uključi i znanje o strukturi samog problema, moguće je dobiti inteligentno pretraživanje (pretraživanje sa elementima automatskog rezonovanja).

Postoji izuzetno veliki broj radova koji se bave primenom TS metode na rešavanje raznih problema kombinatorne i globalne optimizacije, ali veoma malo njih se bavi teorijskim razmatranjima njene konvergencije.

4.6. Pohlepni algoritam

Jedan od najjednostavnijih heurističkih algoritama je *pohlepni algoritam* (*Greedy Algorithm*). Traženje optimalnog rešenja može biti eksponencijalne složenosti, a pohlepni algoritam smanjujući prostor pretraživanja, smanjuje i samu složenost. Naime, pohlepni pristup nas u svakom trenutku usmerava na trenutno najbolje rešenje, ne uzimajući u obzir da nas to ne mora voditi ka globalnom optimumu. Zbog toga, pohlepni algoritam ne mora uvek naći optimalno rešenje, ali je znatno brži od drugih algoritama [36].

Primer: Trgovac koji koristi pohlepni algoritam

Tipičan primer na kojem se ilustruje pohlepni algoritam (zasnovan na pohlepnom pristupu) je *povraćaj ostatka (novca) prilikom kupovine*. Kupac je platio neki proizvod s novčanicom većom nego što je proizvod koštao i sada trgovac treba da vrati određenu sumu novca S. Neka trgovac pri tome ima kod sebe puno novčanica N1, N2, N3 i N4 (N1>N2>N3>N4=1) i neka može odjednom staviti u ruke kupcu samo jednu novčanicu. Trgovcu se žuri (jer je red u trgovini jako veliki) i u interesu mu je da što pre vrati ostatak kupcu. Trgovac koristi pohlepni algoritam, i radi sledeće:

1. stavlja najveću novčanicu koja ne prelazi sumu S
2. od sume S oduzima vrednost te novčanice i suma S poprima vrednost dobijene razlike
3. ako je S jednako nula trgovac je vratio ostatak, inače se vraća na korak 1.

```
Izračunaj ostatak S koji treba vratiti;
Dok (N1 <= S) {
        Stavi novčanicu N1 u ruke kupcu;
        S = S - N1;
}
Dok (N2 <= S) {
        Stavi novčanicu N2 u ruke kupcu;
        S = S - N2;
}
Dok (N3 <= S) {
        Stavi novčanicu N3 u ruke kupcu;
        S = S - N3;
}
Dok (N4 <= S) {
        Stavi novčanicu N4 u ruke kupcu;
        S = S - N4;
}
Obavi ostale poslove;
```

Slika: Pseudokod trgovca

Ako trgovac treba da vrati 62 dinara, a poseduje novčanice od 50, 10, 5 i 1 dinara, trgovac će vratiti ostatak sa jednom novčanicom od 50, jednom od 10 i dve novčanice od 1 dinara. Ovakav pristup ne mora nužno voditi do najboljeg rešenja. Ako trgovac, na primer, treba da vrati 15 dinara, a ima novčanice od 11, 5 i 1 dinara, on će pohlepnim algoritmom vratiti ostatak jednom novčanicom od 11 i četiri novčanice od 1 dinara, iako bi mu bilo brže da vrati ostatak sa tri novčanice od 5 dinara.

Premda pohlepni algoritam ne mora uvek da vodi ka globalnom optimumu, on može biti vrlo koristan u razmatranju NP-teških problema. Naime, pohlepni pristup vodi do rešenja koje sigurno predstavlja donju granicu globalnog maksimuma, ukoliko se traži maksimum, odnosno gornju granicu globalnog minimuma, ukoliko se traži minimum. Kako se pohlepni algoritam brzo izvodi i daje granicu optimalnom rešenju, on nam omogućava da drugim algoritmima smanjimo prostor pretrage rešenja. Dodatna prednost pohlepne strategije je da, čak i kada ne pronađe optimalno rešenje, često vodi na novu strategiju razvoja algoritma koja rezultuje efikasnijim rešenjem ili tehnikom koja brzo pronalazi dobru aproksimaciju rešenja (heuristika) [38]. Heuristika koju koristi pohlepni algoritam je jednostavna: pronađi najbolje trenutno rešenje i idi za njim. Treba napomenuti da postoje mnogi algoritmi koji su zasnovani na pohlepnom pristupu, pa kada govorimo o pohlepnom algoritmu, ustvari govorimo o grupi algoritama.

Postoje problemi kod kojih je dokazano da pohlepni algoritam daje optimalno rešenje. Jedan od tih problema je i problem rasporeda zadataka. Pitanje je kako rasporediti zadatke, a da prosečno vreme završavanja zadatka bude minimalno. Neka su poslovi a_1, a_2, a_3 i a_4, a njihova vremena izvođenja redom dta_1=10s, dta_2=11s, dta_3=5s i dta_4=15s. Ako poređamo poslove redosledom a_1, a_2, a_3 i a_4 onda će vreme završavanja poslova (od početnog trenutka) biti redom t_1=10s, t_2=21s, t_3=26s i t_4=41s. Prosečno vreme završavanja (od početnog trenutka) jednako je tp=24.5s. Tada se promeni raspored tako da prvi poslovi budu oni s manjim vremenima trajanja, tj. da se poslovi obavljaju redom a_3, a_1, a_2 i a_4. To je u skladu s pohlepnim pristupom. Vremena završavanja će iznositi redom t_3=5s, t_1=15s, t_2= 26s i t_4=41, a prosečno vreme izvršavanja će biti tp=21.75s. Očigledno se može dokazati da će prosečno vreme završavanja biti najmanje ako najpre obavljamo najkraće poslove (u skladu s pohlepnim algoritmom). Ipak, valja pripaziti kada ovom problemu pristupamo pohlepnim algoritmom jer nemaju svi poslovi jednak prioritet [37, 38].

5.Pregled metaheurističkih algoritama koji su prirodom inspirisani

U ovom poglavlju će biti opisani algoritmi koji su nastali oponašanjem različitih procesa u prirodi. Najviše pažnje biće posvećeno *genetskim algoritmima,* a biće opisani i *mravlji algoritmi* i *algoritam simuliranog kaljenja*.

5.1. Genetski algoritmi (GA)

5.1.1. Kako su nastali genetski algoritmi

Neće se puno pogrešiti ako se kaže da su genetski algoritmi izum prirode. Tačnije, genetski algoritmi su nastali kao simulacija nekih procesa zapaženih u prirodnoj evoluciji.
Biolozi su bili zaintrigirani mehanizmom evolucije još od kada je teorija evolucije prihvaćena kao opšti uzrok bioloških promena. Mnogi su zapanjeni saznanjem da se život na našoj planeti mogao razviti do sadašnjeg nivoa složenosti u relativno kratkom vremenu, sobzirom da kao dokaz toga imamo brojne fosilne ostatke.
Evolucija je neprekidan proces prilagođavanja živih bića na svoju okolinu tj. na uslove u kojima žive. U prirodi vlada nemilosrdna borba za opstanak u kojoj pobeđuju najbolji a loši umiru. Da bi neka vrsta tokom evolucije opstala, mora se prilagoditi uslovima i okolini u kojoj živi, jer se uslovi i okolina menjaju. Svaka sledeća generacija neke vrste mora pamtiti dobra svojstva predhodne generacije, pronalaziti i menjati ta svojstva tako da ostanu dobra u neprekidno novim uslovima.
Mehanizam po kome funkcioniše evolucija nije u potpunosti razjašnjen, ali su neke od njegovih karakteristika poznate.
Evolucija se odigrava na hromozomima, koji su organski uređaji za kodiranje strukture živih bića. Kako specifičnosti hromozomskog kodiranja i dekodiranja nisu poznate, sledeće karakteristike o teoriji evolucije široko su prihvaćene:
a) proces evolucije operiše na hromozomima a ne na živim bićima koja su kodirana hromozomima
b) proces prirodne selekcije prouzrokuje da se češće reprodukuju hromozomi koji kodiraju uspešne strukture nego drugi hromozomi
c) reprodukcija je tačka na kojoj evolucija počinje svoje delovanje:
- rekombinacija (ukrštanje) može stvoriti različite hromozome kod dece, kombinovanjem materijala iz hromozoma njihovih roditelja,
- mutacija može rezultirati da hromozomi kod dece budu različiti od hromozoma njihovih bioloških roditelja,
d) evolucija nema memoriju.
U ranim sedamdesetim godinama prošlog veka, gore navedene karakteristike prirodne evolucije, zainteresovale su naučnika Johna Hollanda (1975). On je verovao da će uspeti da napravi tehniku za rešavanje teških problema ako na odgovarajući način sjedini ove karakteristike u računarski algoritam. Tako je počeo istraživanje na algoritmu koji upravlja nizovima binarnih cifara (0,1), u kome nizovi predstavljaju hromozome. Koristeći jednostavno kodiranje i mehanizam reprodukcije, Hollandov algoritam je rešio neke ekstremno teške probleme. Kao i priroda, algoritam je bio običan upravljač nad hromozomima. Primenom nekih verzija ovog algoritma danas, postižu se bolja rešenja na širokom spektru problema koje ne možemo rešiti drugim tehnikama.
Kada je Holland počeo da izučava ove algoritme, oni nisu imali ime. Obzirom na njihovo poreklo iz nauke o genetici nazvani su **genetski algoritmi(GA).** Posle velikog broja istraživanja sprovedenih u ovoj oblasti, genetski algoritmi su razvijeni.

Sada su *GA stohastička metoda globalnog pretraživanja koja imitira prirodnu biološku evoluciju.* Primenom principa preživljavanja, koji se sastoji u tome da se od najpogodnijih jedinki proizvode bolja i bolja rešenja, dolazimo do bitne osobine GA: Genetski algoritmi operišu na populaciji potencijalnih rešenja. Aproksimacija novih rešenja u svakoj generaciji dobija se procesom *selekcije* jedinki prema njihovom *nivou prilagođenosti (fitness)* u domenu problema kao i stvaranjem novih jedinki korišćenjem operatora "pozajmljenih" iz genetike kao što su *ukrštanje* i *mutacija*. Ovi procesi rezultiraju na takav način da se jedinke dobijene njima bolje uklapaju u okolinu nego one od kojih su stvorene, baš kao i u prirodnom prilagođavanju.
Ukratko opisano, osnovne ideje metode su da se izabere inicijalna populacija i da se zatim kroz niz generacija evoluiranja te populacije vrši popravka trenutno najboljeg rešenja. Popravka, tj. transformacija polaznih rešenja i generisanje potomaka se vrši primenom tzv. *operatora*, od kojih su osnovni *selekcija*, *ukrštanje* i *mutacija*.

5.1.2.Osnovna struktura GA

Postoji više načina da se predhodno pomenute karakteristike prirodne evolucije povežu sa genetskim algoritmima. Za početak treba uvažiti dva mehanizma koja povezuju GA sa problemom koji se rešava. Jedan od njih je način kodiranja potencijalnih rešenja na problem hromozoma, tj. njihovo kodiranje u hromozome. Drugi je funkcija ocene(fitness), koja predstavlja meru valjanosti nekog hromozoma.
Način kodiranja rešenja igra važnu ulogu u GA. Tehnika kodiranja može varirati od problema do problema kao i od GA do GA. U ranim GA za kodiranje su korišćeni nizovi bitova, binarno kodiranje. Kasnije su naučnici razvijali i mnoge druge tehnike kodiranja.
Najverovatnije je da ni jedna tehnika kodiranja ne radi najbolje za sve probleme, pa je potrebna velika veština u izboru dobre tehnike kada se problem proučava. Zato, kada se bira tehnika za kodiranje u nekom stvarnom problemu, treba obratiti pažnju na niz faktora, o kojima će biti reč kasnije.
Funkcija ocene (*fitness function*), je veza između GA i problema koji se rešava. Funkcija ocene uzima kao ulazni podatak hromozom a kao rezultat vraća broj ili listu brojeva koji predstavljaju performance tog hromozoma. Ova funkcija igra istu ulogu u GA kao što okolina to radi u prirodnoj evoluciji. Interakcija jedne jedinke sa okolinom daje meru njene prilagođenosti, a interakcija hromozoma sa funkcijom ocene određuje meru pogodnosti, tj. koliko je taj hromozom pogodan za dalju reprodukciju.
Ako se uzme da su date početne komponente: problem, tehnika za kodiranje rešenja i funkcija koja daje informaciju o tome koliko je neko rešenje dobro. Tada se može primeniti GA da sprovede simuliranu evoluciju na nekoj populaciji rešenja. Osnovna struktura GA izgleda ovako:
a) Inicijalizovati populaciju hromozoma
b) Oceniti "kvalitet" svakog hromozoma u populaciji
c) Stvoriti nove hromozome od hromozoma iz postojeće populacije

d) Ukloniti neke članove populacije da bi se napravilo mesta za nove hromozome
e) Ubaciti nove hromozome u populaciju
f) Zaustaviti proceduru i prikazati najbolji hromozom ako je vreme isteklo, u suprotnom ići na korak b).
Treba napomenuti da kriterijum zaustavljanja kod GA može biti maksimalan broj generacija, broj generacija bez popravke trenutno najboljeg rešenja, prevelika sličnost jedinki ili maksimalno dozvoljeno vreme izvršavanja koje je i najrasprostranjenije jer omogućava poređenje sa drugim metodama.
Na osnovu formulisane osnovne strukture, pseudokod GA ima sledeci oblik (u pseudo PASKAL-u):

```
Procedura GA
begin
    t = 0;
   Generiši inicijalnu populaciju P(t)=P(0);
     "Oceni" P(t);
   While (dok vreme ne istekne ) do
      begin
         t = t +1;
         Selektuj P(t) od  P(t-1);
           Reprodukuj parove u P(t)
                 begin
                    Ukrštanje;
                    Mutacija;
                    Reinsertovanje;
                 end
         "Oceni" P(t);
       end
end
```

U ovom pseudokodu populacija hromozoma u trenutku t je predstavljena pomoću vremenski-zavisne promenljive P(t) sa inicijalnom populacijom P(0).
Ako u ovoj proceduri sve prođe dobro, početna populacija će biti zamenjena sve boljim i boljim hromozomima. Najbolja jedinka u finalnoj populaciji može biti najbolje rešenje problema.
Genetski algoritmi se bitno razlikuju od mnogih tradicionalnih optimizacionih metoda i metoda pretraživanja. Najznačajnije razlike su [5, 7]:

- GA pretražuje populaciju čvorova u prostoru potencijalnih rešenja, a ne jedan čvor, što mu daje osobinu robustnog algoritma;
- Tokom procesa rešavanja, GA ne koristi dodatne informacije o prirodi problema;
- GA koriste probabilistička (verovatnosna) pravila, a ne deterministička
- GA ne koristi same parametre,već vrši kodiranje parametara.

Važno je primetiti da GA proizvodi mnogo potencijalnih rešenja za zadati problem, a izbor konačnog rešenja se ostavlja dizajneru. Izbor takođe zavisi od prirode i veličine problema, kao i od toga da je pored kvaliteta jako bitna i brzina dobijanja rešenja.

5.1.3. Reprezentacija i inicijalizacija populacije

GA operišu na određenom broju potencijalnih rešenja koja se nazivaju *populacija*. Populacija se sastoji od hromozoma, koji su ustvari kodirani parametri nekog problema. Najčešće se populacija sastoji od 21 hromozoma ili više, mada postoje GA takozvani "micro GA" koji koriste populacije manje od 10 jedinki [39].
Postoje različiti tipovi reprezentacije članova neke populacije. U daljem tekstu će biti predstavljene neke tehnike kodiranja koje se koriste na različitim problemima.

Kodiranje (reprezentacija)

Binarno kodiranje

Jedna od bitnih odlika GA je da se operatori ne primenjuju u okviru samog prostora rešenja, već se najpre vrši kodiranje rešenja nizovima simbola nekog konačnog alfabeta.
U daljem razmatranju će se uvek pod *rešenjem* podrazumevati njegova reprezentacija pomoću izabranog koda. Takva reprezentacija naziva se *jedinka* populacije ili *hromozom*. Svaki od simbola u kodu se naziva *gen*.
Najjednostavnije je binarno kodiranje (0-1 simbolima). U binarnoj reprezentaciji svaki hromozom je predstavljen kao niz binarnih cifara:

1 0 1 1 0 1 1 0

Dakle, u pitanju je niz bitova {0,1} u obliku vektora fiksirane dužine. Kod različitih problema selekcije koji se pojavljuju u oblastima modelovanja i optimizacije, dužina vektora odgovara broju stavki između kojih vršimo selekciju. Konkretan takav problem bi bio problem ranca u kome je rešenje predstavljeno vektorom sadržanom od 0 i 1 gde 1 znači da odredjeni predmet indeksiran tom pozicijom ulazi u rešenje.
Nedostatak binarnog kodiranja je taj što Hammingova razdaljina, tj. broj bitova u kojima se dva broja razlikuju može biti velika, u najgorem slučaju između dva susedna broja ona može biti jednaka dužini binarnog zapisa. Dakle, dve tačke koje su blizu u problemu, nisu blizu i u reprezentativnom prostoru. Na primer, između brojeva:

$255_{(10)} = 011111111_{(2)}$ i $256_{(10)} = 100000000_{(2)}$

Hammingova razdaljina iznosi 9. Drugim rečima, ako je GA u nekom koraku pronašao jedno dobro rešenje za b=011111111, a optimum se postiže za b=100000000 treba promeniti svih devet bitova.
Kako bi se ispravio ovaj nedostatak, za kodiranje se koristi i Gray-ev kod.

Kodiranje Gray-evim kodom

Susedni brojevi kodirani Gray-evim kodom, razlikuju se samo u jednom bitu. Algoritam transformacije binarnog broja b= $b_m b_{m-1}...b_1$ u Gray-ev kod g = $g_m g_{m-1}...g_1$ i obrnuto, iz Gray-evog koda u binarni glasi:

$$g_m = b_m \qquad g_k = b_k \oplus b_{k+1} \text{ , k = 1,2….m-1} \qquad b_k = \sum_{k=j}^{m} g_k \pmod 2 \text{ , j =1,2…m.}$$

Tako da na primer u binarnom kodu broj 7 i 8 su predstavjeni kao :

$7_{10} = 0111_2$ i $8_{10} = 1000_2$

Dok su u Gray-evom kodu zapisani kao:

$7_{10} = 0100_g$ i $8_{10} = 1100_g$.

U Gray-evom kodu se ova dva broja razlikuju u samo jednom bitu. Koliko je ta osobina dobra toliko moze biti i loša, jer je ovaj kod ujedno i *ciklični kod* koji se može formirati dodavanjem 0 i 1 na dvobitnu formu koda da bi se dobio tro-bitni kod :
Kada dvobitni Gray-ev kod (00,01,11,10) zapišemo u oba smera

00, 01 , 11 , 10 , 10 , 11 , 01 , 00

I dodamo prvoj polovini nule a drugoj jedinice:

<u>000</u>, 001 , 011 , 010 , 110 , 111 , 101 , <u>100</u>

Dobijamo tro-bitni kod kod kojeg se takođe prvi i poslednji broj u nizu razlikuju za samo jedan bit. To nije dobra osobina kod operatora jednobitne mutacije koji može izazvati velike promene i udaljiti se od potencijalno dobrog rešenja.

Kodiranje realnim brojevima

Kodiranje realnim brojevima daje puno prednosti u odnosu na binarno kodiranje naročito u slučajevima optimizacije numeričkih funkcija. U tom slučaju je bolje koristiti oblik broja sa pokretnom tačkom, tj. normalizovani eksponencijalni oblik koji se sastoji od mantise *m*, osnove *b*, i eksponenta *e*:

$$X = m \cdot b^e$$

Za takav prikaz u GA nije potrebno ugrađivati poseban mehanizam kodiranja kakav mora biti prisutan pri binarnom kodiranju, jer hromozom i jeste realan broj samo je zapisan kao broj sa pokretnom tačkom prema standardu IEEEE 854.

Primer kodiranja realnim brojevima kod optimizacije numeričkih funkcija je očigledniji za primenu od primera problema trgovačkog putnika (*traveling sallesman problem*,TSP), gde se u nekim slučajevima takođe može koristiti kodiranje realnim brojevima [3]. U tom slučaju se koristi vektor dužine n, gde n odgovara broju gradova. Komponente vektora su poređane tako da njihov redosled određuje redosled posećivanja gradova. Na primer, vektor:

v = (2.34, -1.09, 1.91, 0.87, -0.12, 0.99, 2.13, 1.23, 0.55)

odgovara ruti

2 – 5 – 9 – 4 – 6 – 8 – 3 – 7 – 1.

Kako je najmanji broj -1.09 na drugoj poziciji vektora v, a drugi najmanji broj je -0.12 na petoj poziciji vektora v, redosled počinje sa 2-5 itd. Ove realne vrednosti mogu predstavljati rastojanje između gradova, dok negativni predznaci mogu da se iskoriste kada između dva grada postoje dva različita jednosmerna puta različite dužine, itd.
Kod kodiranja realnim brojevima, nova rešenja se mogu dobiti raznovrsnim metodama koje se primenjuju na vektore sa realnim vrednostima. Stoga treba napomenuti da su za svaku vrstu kodiranja, kao što su binarno i kodiranje realnim brojevima, posebno definisani genetski operatori.

Permutacije

Za permutacije se može reći da su jedan od načina celobrojnog kodiranja. Za primer je uzeta permutacija primenjena na problem rasporeda. Redosled obavljanja nekih poslova kojih ima j, možemo zapisati u obliku permutacije:

$$[1, 2, 3 ..... j].$$

Redosled u permutaciji odgovara redosledu obavljanja poslova označenih brojevima od 1 do j. Ako je potrebno neki od tih poslova obaviti više puta, jednostavno se taj broj uključi više puta u permutaciju.
Permutacije se dosta koriste kod već pomenutog TSP problema.

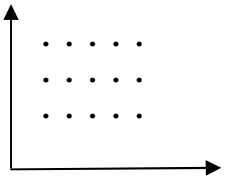
Inicijalizacija

Kada se odredi način kodiranja, sledeći korak u GA je kreirati inicijalnu populaciju. Obično se uzima da je veličina populacije konstantna, odnosno da se ona ne menja kroz nove iteracije. Inicijalizacija populacije se najčešće postiže slučajnim generisanjem određenog broja jedinki iz prostora mogućih rešenja. Za primer, binarna populacija od N

hromozoma dužine M bita, može se generisati sa NxM slučajnih brojeva jednako dodeljenih iz skupa $\{0,1\}$.

Početna populacija može biti uniformna, tj. sve jedinke mogu biti iste. Inicijalizacija se takođe može obaviti pomoću konstruktivnih heuristika. Kod konkretnih problema treba iskoristiti informacije koje mogu pomoći za inicijalizaciju populacije sa nagoveštajem o bliskom rešenju.

Ako se ne zna puno o problemu, postoje i drugi načini da se inicijalizuje populacija, s obzirom da slučajnim generisanjem može doći do nagomilavanja u jednoj oblasti prostora mogućih rešenja. Da bi se postigla raznolikost, moguća alternativa je da se kao početna populacija uzme mreža tačaka kao na slici:



Druga mogućnost je da svaka jedinka koja je odabrana za inicijalnu populaciju bude bar na minimalnoj udaljenosti od ostalih jedinki [3].

Iako ove alternative mogu oduzeti puno vremena, one omogućavaju da se jedinke iz različitih oblasti prostora rešenja uključe u inicijalnu populaciju.

5.1.4. Funkcija prilagođenosti (Fitness function)

Funkcija prilagođenosti se još u literaturi naziva *funkcija sposobnosti*, *funkcija ocene* ili *eval funkcija*. U slučaju problema optimizacije, naziv *objective function* (funkcija cilja) se najčešće koristi u originalnom problemu, a *evaluation(fitness) function* (funkcija ocene) je ustvari prosta transformacija date funkcije cilja [5].

U ranijem tekstu je pomenuto da fitness funkcija predstavlja meru prilagođenosti hromozoma, tj. koliko je svaki hromozom pogodan za dalju reprodukciju.

Za zadati problem optimizacije, najveću poteškoću predstavlja definisanje fitness funkcije koja je ključ za process selekcije. Iz tog razloga postoji više načina definisanja ove funkcije. U najednostavnijoj interpretaciji, fitness funkcija je ekvivalent funkciji koju treba optimizovati:

$$\text{fitness}(x) = f(x).$$

Ovaj rezon možemo primeniti za slučaj da se traži maksimum neke funkcije $f(x)$. Kod slučaja minimizacije morali bi izvršiti neke korekcije. Kako funkcija $f(x)$ ne sme imati negativne vrednosti, jer bi fitness(x) bila negativna (verovatnoća izbora jedinke je u većini slučajeva proporcionalna sa fitnessom tako da negativna vrednost fitnessa onda nema smisla), tada u slučaju problema minimizacije možemo fitness računati kao:

$$\text{fitness}(x) = f(x) + \text{M}.$$

M se može podesiti tako da je fitness uvek nenegativan broj. Ako bi za primer uzeli da je $f(x) = \sin(x)$ a M =1000, fitness funkcija za sve hromozome će biti približno jednaka i iznosiće ≈ 1000. To znači da će prilikom selekcije svi hromozomi imati prilično jednaku verovatnoću pojavljivanja u sledećoj populaciji, tj. svi su podjednako dobri i loši, što je sve zajedno – loše!
Ovakva jednostavna rešenja najčešće neće dati dobre rezultate. Stoga se svuda sem u najprostijim GA pribegava drugim oblicima računanja fitness funkcije.
Jednu od reprezentacija fitness funkcije naučnik De Jong nazvao je *relativni fitness* (De Jong,1975) i definisao je kao [5]:

$$\mathrm{F}(c) = f(\phi(c)),$$

Gde je ϕ-funkcija iz zadatog problema, $f(.)$ konvertuje vrednost funkcije cilja u nenegativan broj, a F(.) je rezultujući fitness.
U najvećem broju slučajeva, vrednost fitness funkcije je proporcionalna broju potomaka koji će jedna jedinka reprodukovati u narednu generaciju [20]. U tom slučaju, fitness svake jedinke, $\mathrm{F}(c_i)$, se izračunava kao:

$$\mathrm{F}(c_i) = \frac{\phi(c_i)}{\sum_{i=1}^{N} \phi(c_i)},$$

gde je N- veličina populacije, a $\phi(c_i)$ vrednost funkcije cilja jedinke *i*. Iako ovakva definicija fitnessa obezbeđuje da svaka jedinka ima mogućnost reprodukcije prema vrednosti svog fitnessa, i dalje ostaje problem sa negativnim vrednostima funkcije cilja.

Linearno skaliranje se takođe koristi za definisanje fitness funkcije:

$$F(c) = \alpha \cdot \phi(c) + \beta,$$

Gde je α pozitivan skalar ako je u pitanju problem maksimizacije a negativan ako je u pitanju problem minimizacije, a β obezbeđuje da rezultujuće fitness F( ) vrednosti budu nenegativne. Korišćenjem linearnog skaliranja, očekivani broj potomaka je približno proporcionalan performansama jedinke. Zahvaljujući tome, najprilagođenije jedinke u ranim generacijama mogu dominirati reprodukcijom što može dovesti do brze konvergencije ka lokalnim optimumima.
Da bi performance nekog GA bile bolje, pri izboru fitness funkcije treba težiti ka ispunjenju sledećih uslova:

- Da jedinke sa sličnim genetskim kodom imaju slične fitness vrednosti,
- Da fitness funkcija bude glatka,
- Da fitness funkcija nema mnogo lokalnih ekstrema,
- Da fitness funkcija nema suviše izolovan globalni maksimum…

5.1.5. Selekcija

Podela

Genetski algoritmi koriste mehanizam selekcije za izbor jedinki koje će učestvovati u reprodukciji. Selekcijom se omogućava prenošenje boljeg genetskog materijala iz generacije u generaciju. Zajedničko svojstvo svih vrsta selekcija je veća verovatnoća izbora boljih jedinki za reprodukciju. Postupci selekcije se međusobno razlikuju po načinu izbora boljih jedinki. Prema načinu prenošenja genetskog materijala boljih jedinki u sledeću iteraciju postupci selekcije se dele na [1]:

- *generacijske selekcije* - selekcijom se direktno biraju bolje jedinke čiji će se genetski materijal preneti u sledeću iteraciju i
- *eliminacijske selekcije* - biraju se loše jedinke za eliminaciju, a bolje jedinke prežive postupak selekcije.

Generacijskom selekcijom se biraju bolje jedinke koje će učestvovati u reprodukciji. Između jedinki iz populacije iz prethodnog koraka biraju se bolje jedinke i kopiraju u novu populaciju. Ta nova populacija, koja učestvuje u reprodukciji, naziva se *međupopulacijom* [25]. Na taj način se priprema nova generacija jedinki za postupak reprodukcije. To je ujedno prvi nedostatak generacijskih selekcija, jer se u jednoj iteraciji odjednom nalaze dve populacije jedinki. Broj jedinki koje prežive selekciju je manji od veličine populacije. Najčešce se ta razlika u broju preživelih jedinki i veličine populacije popunjava duplikatima preživelih jedinki [29]. Pojava duplikata u sledećoj iteraciji je drugi nedostatak generacijskih selekcija, jer duplikati nikako ne doprinose kvalitetu dobijenog rešenja, već samo usporavaju proces evolucije [27, 6]. Generacijska selekcija postavlja granice među generacijama. Svaka jedinka egzistira tačno samo jednu generaciju. Izuzetak su jedino najbolje jedinke koje žive duže od jedne generacije i to samo ako se primeni elitizam.

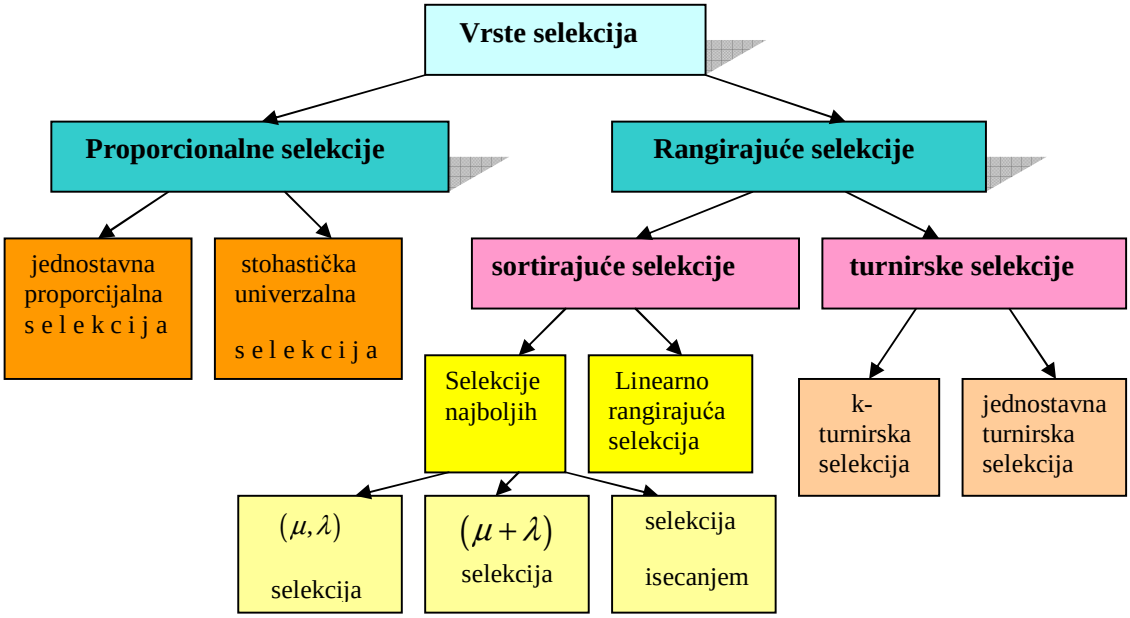
Kod eliminacijskih selekcija bolje jedinke preživljavaju mnoge iteracije pa stroge granice između generacija nema, kao što je to slučaj kod generacijskih selekcija. Eliminacijska selekcija briše M jedinki. Parametar M naziva se mortalitetom ili generacijskim jazom. Izbrisane jedinke se zamenjuju jedinkama koje se dobijaju reprodukcijom. Eliminacijskom selekcijom se otklanjaju oba nedostatka generacijske selekcije: nema dve populacije u istoj iteraciji i u postupku selekcije se ne stvaraju duplikati. Na prvi pogled je uvođenje novog parametra M nedostatak eliminacijske selekcije, jer se svakim dodatnim parametrom značajno produžava postupak podešavanja algoritma. Međutim, uvođenjem novog parametra, nestaje parametar verovatnoće ukrštanja, jer ukrštanje treba obaviti tačno onoliko puta koliko je potrebno da se dopuni populacija do početne veličine N. Što je veća verovatnoća ukrštanja, treba biti veći mortalitet i obrnuto. Na primer, ako se koristi uniformno ukrštanje, koje proizvodi jednu novu jedinku, tada tačno M puta treba ponoviti ukrštanje kako bi se dopunila populacija, odnosno zamenile eliminisane jedinke [1].

Selekcijom se osigurava prenošenje boljeg genetskog materijala s većom verovatnoćom u sledeću iteraciju. Zavisno od metode izbora boljih jedinki kod generacijskih selekcija, odnosno loših jedinki kod eliminacijskih selekcija, postupci selekcije se dele na **proporcionalne** i **rangirajuće** [26]. Važno je napomenuti da je zajedničko obeležje svim

selekcijama veća verovatnoća preživljavanja bolje jedinke od bilo koje druge lošije jedinke. Selekcije se razlikuju prema načinu određivanja vrednosti verovatnoće selekcije određene jedinke.

Proporcionalne selekcije biraju jedinke s verovatnoćom koja je proporcionalna prilagođenosti jedinke, odnosno verovatnoća selekcije određene jedinke zavisi od koeficijenta prilagođenosti jedinke i prosečne prilagođenosti populacije. Broj jedinki s određenim svojstvom u sledećoj iteraciji je proporcionalan koeficijentu prosečne prilagođenosti tih jedinki i prosečne prilagođenosti populacije.

Rangirajuće selekcije biraju jedinke s verovatnoćom koja zavisi od položaja jedinke u poretku jedinki sortiranih po prilagođenosti. Rangirajuće selekcije se dele na *sortirajuće* i *turnirske selekcije*. Sortirajuće selekcije su: linearno rangirajuća selekcija i selekcija najboljih tj. (μ, λ) selekcija i selekcija isecanjem. Turnirske selekcije se dele prema broju jedinki koje učestvuju u turniru. Na slici je prikazana podela selekcija na proporcionalne i rangirajuće, kao i na njihove podvrste [1].



Operator selekcije;

Kriterijumi za upoređivanje operatora selekcije

U proučavanju operatora selekcije autori polaze od različitih predhodno definisanih veličina. Najčešće korišćene veličine koje karakterišu operator selekcije su: **trajanje preuzimanja**, koje su uveli naučnici Goldberg i Deb, zatim **stopa reprodukcije**, **intenzitet selekcije**, **selekciona razlika**, **gubitak raznovrsnosti** i **selekcioni pritisak**.

Treba napomenuti da su selekcioni pritisak i intenzitet selekcije sinonimni termini, koji se često koriste u različitim kontekstima i za različite osobine selekcionog metoda [7].

Trajanje preuzimanja je prosečan broj generacija nakon kojih se populacija sastoji od N duplikata najbolje jedinke, ukoliko se u svakoj iteraciji upotrebi samo operator selekcije bez ukrštanja i mutacije. Drugim rečima, meri se broj iteracija nakon kojih se eliminišu sve jedinke osim najbolje. Trajanje preuzimanja poprima najmanju vrednost kada je verovatnoća selekcije svih jedinki jednaka i iznosi p=1/N (slučajno pretraživanje).
Osim od vrste selekcije i njenih kontrolnih parametara, trajanje preuzimanja za proporcionalne selekcije zavisi i od funkcije cilja.
Selekcijom preživljavaju bolje jedinke i očekuje se da prosečna vrednost preživelih jedinki bude veća od prosečne vrednosti cele populacije.
Selekciona razlika s je razlika prosečne vrednosti prilagođenosti preživelih jedinki $\overline{f_p}$ i prosečne vrednosti prilagođenosti svih jedinki $\overline{f}$ u svakoj iteraciji t:

$$s(t) = \overline{f_p}(t) - \overline{f}(t). \qquad (5.1.1)$$

Prosečna vrednost prilagođenosti svih jedinki koje čine populaciju računa se prema izrazu:

$$\overline{f}(t) = \frac{1}{N}\sum_{i=1}^{N} f_i(t)\ . \qquad (5.1.2)$$

Dalje, neka je $\sigma(t)$ standardna devijacija svih prilagođenosti u populaciji u generaciji *t:*

$$\sigma(f) = \sqrt{\frac{\sum_{i=1}^{N}\left[f_i(t) - \overline{f}(t)\right]^2}{N}}\ . \qquad (5.1.3)$$

Uz pretpostavku da prilagođenost populacije f_i ima normalnu raspodelu $N[\overline{f}(t), \sigma(t)]$ u generaciji *t*, selekciona razlika je proporcionalna standardnoj devijaciji populacije:

$$s(t) = s_I \cdot \sigma(t), \qquad (5.1.4)$$

pri čemu se količnik $s_I = \frac{s(t)}{\sigma(t)}$ naziva *intenzitet selekcije* [23].
Naravno, prilagođenost nema normalnu raspodelu za sve funkcije cilja i u svakom trenutku t. Međutim, za većinu funkcija cilja pred kraj procesa evolucije (t>>) raspodela vrednosti funkcije prilagođenosti populacije može se dobro aproksimirati normalnom raspodelom [23].
Selekciona razlika zavisi od standardne devijacije populacije koja se menja iz generacije u generaciju, a intenzitet selekcije je konstantna vrednost koja zavisi samo od vrste selekcije i njenih parametara.

Gubitak raznovrsnosti datog selekcionog metoda, je procenat jedinki koje nisu izabrane za reprodukciju. Gubitak raznovrsnosti je broj između 0 i 1 i treba težiti tome da bude što manji kako bi se izbegla prerana konvergencija.
Stopa reprodukcije je količnik broja jedinki s određenom vrednošcu prilagođenosti posle i pre selekcije [25]. Neka je Q(f) broj jedinki čija je vrednost funkcije prilagođenosti jednaka f i to pre, a Q*(f) posle selekcije. Stopa reprodukcije r za zadatu vrednost prilagođenosti f računa se prema izrazu:

$$r(f)=\begin{cases}\dfrac{Q^*(f)}{Q(f)}: & Q(f)>0\\ 0 \quad : & Q(f)=0\end{cases} \qquad (5.1.5)$$

Svi postupci selekcije favorizuju bolje jedinke pa je njihova stopa reprodukcije veća od jedan. Odnosno, sve selekcije kažnjavaju loše jedinke pa je stopa reprodukcije za loše jedinke manja od jedan.
Neki od zahteva nad postupkom selekcije prema Bäcku [22] su:

- promena kontrolnih parametara selekcije mora biti jednostavna, a posledice te promene moraju biti predvidive;
- poželjno je da selekcija ima samo jedan kontrolni parametar

Nažalost većina postupaka selekcije ne zadovoljava ni prvi zahtev [22].

Verovatnoća selekcije

Neka je Ω prostor elementarnih događaja: $\Omega=\{E_1,E_2,...E_i,..E_N\}$, gde je E_i elementarni događaj selekcija i-te jedinke. Neka je postupkom selekcije odabrana jedinka s oznakom x. Neka je X diskretna slučajna promenljiva koja poprima vrednosti $x\in R$, gde je R={1,2,3,...,i,...,N} diskretan skup od N elemenata. Oznaka $p(i)=P\{x=i\}$ označava verovatnoću selekcije i-te jedinke.
Suma svih verovatnoća mora biti jednaka jedan:

$$\sum_{i=1}^{N} p(i)=1$$

Neka je F funkcija raspodele slučajne promenljive X : $F(x)=P\{X\le x\}$, gde je

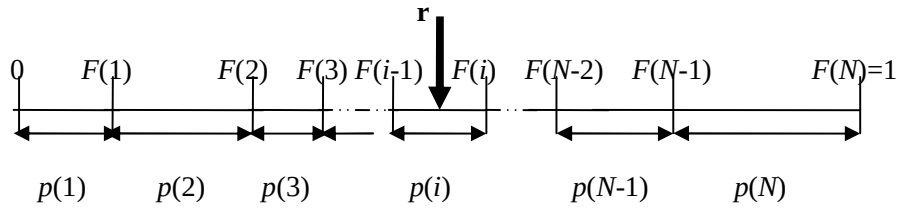
$$F(x)=\sum_{i=1}^{x} p(i).$$

Kod rangirajućih selekcija spomenuta oznaka jedinke zavisi od njenog ranga. Najbolja jedinka ima oznaku 1, druga jedinka po prilagođenosti ima oznaku 2 i tako sve do najslabije jedinke koja ima oznaku N. Isto tako, jedinke mogu nositi oznake obrnutim redosledom: tako da najbolja ima oznaku N, a najgora oznaku 1. Kod proporcionalnih selekcija oznaka ne odgovara rangu jedinke, već predstavlja samo redni broj jedinke u populaciji (prva jedinka ne mora biti najbolja, kao što ni zadnja ne mora biti najgora).

Proporcionalne selekcije

Jednostavna proporcionalna selekcija (rulet selekcija)

Holland je 1975. godine predstavio jednostavni genetski algoritam s jednostavnom proporcionalnom selekcijom, jednostavnom mutacijom i ukrštanjem sa jednom tačkom prekida. Osim toga, Holland je još tada postavio temelje teorijske analize rada genetskih algoritama opisavši jednostavni genetski algoritam hipotezom gradivnih blokova i donekle predvidevši ponašanje algoritma uz pomoć teoreme o shemama.
Verovatnost preživljavanja jedinke pri proporcionalnoj selekciji proporcionalna je njenoj prilagođenosti. Često se pri opisu proporcionalnih selekcija koristi analogija sa točkom ruleta. Analogija nije sasvim potpuna, jer svi brojevi na ivici točka ruleta zauzimaju jednake kružne isečke, dok je zamišljena veličina kružnog isecka jedinke proporcionalna njenoj prilagođenosti, tj. verovatnoći njene selekcije.
Proporcionalna selekcija u svakom koraku generiše slučajni broj r u intervalu [0,1]. Ako je r u intervalu [$F(i\text{-}1)$, $F(i)$], gdje je $F(x)$ već opisana funkcija raspodele, tada je selektovana i-ta jedinka. Taj se postupak ponavlja sve dok se ne odabere N jedinki, s tim da se jedna ista jedinka može odabrati proizvoljan broj puta.

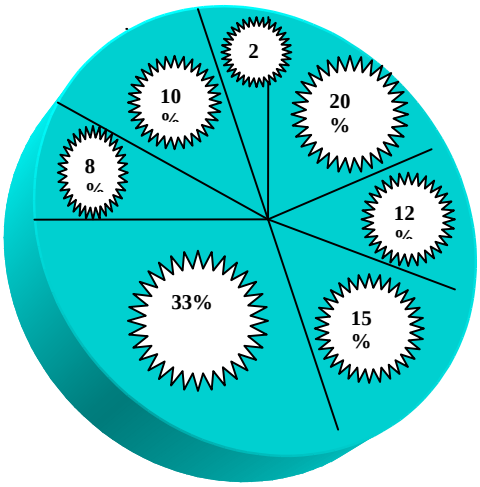


Selekcija *i*-te jedinke upotrebom jednostavne proporcionalne selekcije

Verovatnoća selekcije jedinke, tj. veličina njenog kružnog isecka, određuje se na temelju prilagođenosti jedinke f_i i ukupne prilagođenosti populacije $\sum f_i$. Verovatnoća selekcije i-te jedinke $p(i)$ koja će učestvovati u reprodukciji jednaka je količniku prilagođenosti jedinke i ukupne prilagođenosti populacije [22]:

$$p(i) = \frac{f_i}{\sum_{n=1}^{N} f_n}. \qquad (5.1.6)$$

Na primer, neka se populacija sastoji od 7 jedinki čije su vrednosti prilagođenosti: 10, 15, 2, 8, 12, 20 i 33. Prilagođenost populacije je suma prilagođenosti svih jedinki i iznosi 100. Verovatnoće selekcije su 10/100=0.1 (10%), 15/100=0.15 (15%), itd.

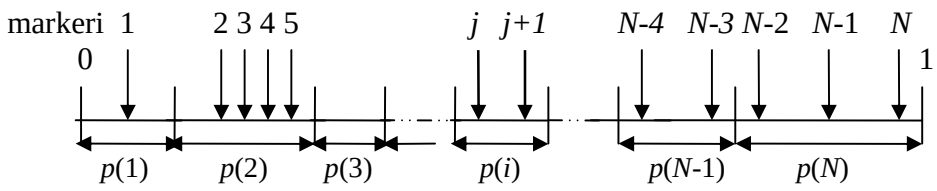


Primer verovatnoće selekcije predhodne populacije na točku ruleta

Može se primetiti da predhodno primenjeni mehanizam nema smisla ako funkcija prilagođenosti ima negativne vrednosti, jer bi u tom slučaju verovatnoća selekcije mogla postati negativna. Da bi se otklonio ovaj nedostatak, pre izračunavanja verovatnoće selekcije mora se obaviti sortiranje ili skaliranje funkcije prilagođenosti [6].
Loše odabrana funkcija prilagođenosti može postupak selekcije pretvoriti u slučajno biranje. Na primer, neka je $f(x)=1000+\sin(x)$, gde je $x \in [-2\pi, 2\pi]$. Ako se populacija sastoji od N=100 jedinki, verovatnoća selekcije bilo koje jedinke će biti približno 0.01. Tačnije, verovatnoća selekcije najbolje moguće jedinke za navedeni primer iznosi 0.01001, a najlošije 0.00999. Posledica toga je da će najbolja jedinka u populaciji preživeti selekciju sa samo 0.002% većom verovatnoćom od najlošije jedinke.
Jednostavna proporcionalna selekcija je vremenski najzahtevnija, nije pogodna za paralelizaciju, postavlja ograničenja nad funkcijom prilagođenosti ili se moraju obaviti neke vremenski zahtevne predradnje.

Stohastička univerzalna selekcija

Stohastička univerzalna selekcija (eng. stochastic universal sampling) se razlikuje od jednostavne proporcionalne selekcije prema načinu izbora jedinki. Verovatnoća selekcije je ista i izračunava se takođe prema izrazu (5.1.6) [26]. Jednostavna proporcionalna selekcija se obavlja u N koraka: N puta se generiše slučajan broj prema kojem se određuje koja jedinka je selektovana. Stohastička univerzalna selekcija bira sve jedinke u jednom koraku: generiše se N ekvidistantnih markera tako da se interval [0,1] podeli na N+1 jednakih odsečaka. Broj markera koji pokazuju na pojedinu jedinku određuje koliko će se njenih kopija preneti u sledeću populaciju. Na slici je prikazan primer u kojem se prenosi: jedna kopije jedinke J_1, četiri kopije jedinke J_2, ni jedna kopija jedinke J_3, dve kopije jedinke Ji, itd.



Selekcija svih N jedinke u jednom koraku upotrebom stohastičke selekcije

Nedostatak proporcionalnih selekcija je taj što funkcija prilagođenosti ne sme imati negativne vrednosti (jer bi verovatnoća selekcije bila negativna), kao ni samo približno jednake vrednosti (GA se pretvara u postupak slučajnog pretraživanja).

Rangirajuće selekcije

Podela i svojstva rangirajucih selekcija

Kod rangirajućih selekcija verovatnoća selekcije ne zavisi direktno od prilagođenosti, već od položaja jedinke u redosledu jedinki sortiranih po prilagođenosti..
Jedinke se mogu sortirati ili po rastućim ili opadajućim vrednostima funkcije prilagođenosti, tj. ili po rastućim ili opadajućim vrednostima funkcije cilja. U prvom slučaju je: $f_1 \le f_2 \le f_3 ... \le f_i \le ... \le f_N$, a u drugom: $f_1 \ge f_2 \ge f_3 ... \ge f_i \ge ... \ge f_N$.
Pri sortiranju s rastućom prilagođenošću, najgora jedinka ima rang 1, a najbolja N.
Za rangirajuću selekciju nije važna vrednost prilagođenosti, već njen odnos prema vrednostima prilagođenosti ostalih jedinki. Isto tako nije važno da li je funkcija prilagođenosti negativna ili ima samo približno jednake vrednosti. *Kod rangirajućih selekcija funkcija prilagođenosti nema nikakvih ograničenja i jednaka je funkciji cilja.*
Rešenje x_1 je bolje od rešenja x_2, ako je $f(x_1) > f(x_2)$ i ukoliko se traži maksimum funkcije *f*. Dalje, ako je $f(x_1) = f(x_2)$, bilo koje od dva rešenja se može proglasiti boljim i preneti ga u novu generaciju. Nad funkcijom prilagođenosti se ne postavljaju nikakva ograničenja (ne treba biti neprekidna i sl.). Dovoljno je da se za svaki par potencijalnih rješenja x_1 i x_2, koji su elementi prostora rešenja (skupa Ω), može odrediti da li je vrednost funkcija cilja $f(x_1)$ veća, jednaka ili manja od $f(x_2)$.
U skup rangirajućih selekcija spadaju *sortirajuće* i *turnirske* selekcije. Sortirajuće selekcije obavljaju sortiranje jedinki prema njihovoj vrednosti funkcije cilja. Postupak troši značajni deo od ukupnog procesorskog vremena, jer se ponavlja u svakoj iteraciji.

S obzirom da je sortiranje vremenski zahtevan posao u odnosu na trajanje same selekcije, sortirajuće selekcije se ređe koriste.
S druge strane, za turnirske selekcije važan je samo odnos između nekoliko slučajno odabranih jedinki i nije potrebno obavljati sortiranje cele populacije. Prema broju slučajno odabranih jedinki iz skupa svih jedinki (cele populacije), turnirske selekcije se dijele na 2-turnirske, 3-turnirske, 4-turnirske itd.

Linearno rangirajuća selekcija

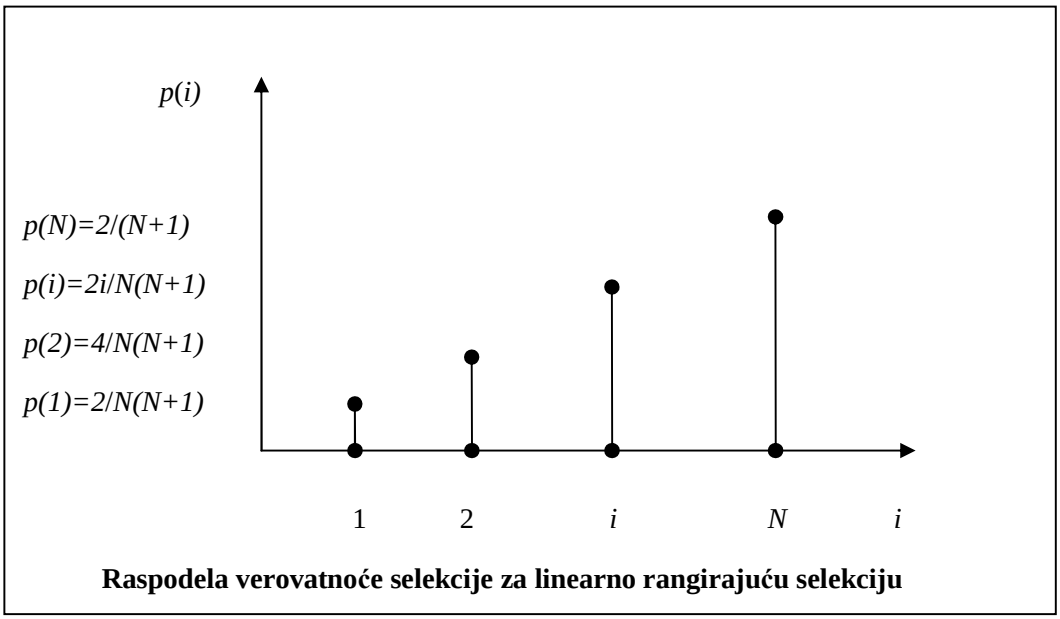
Kod linearno rangirajuće selekcije verovatnoća selekcije je proporcionalna rangu, odnosno poziciji jedinke u poretku jedinki sortiranih po prilagođenosti. Verovatnoća selekcije za linearno rangirajuću selekciju izračunava se prema izrazu:

$$p(i) = \frac{i}{\sum_{i=1}^{N} i} = \frac{2i}{N(N+1)}, \qquad (5.1.7)$$

s tim da najbolja jedinka ima rang N, a najgora 1.
Suma verovatnoća selekcija svih jedinki mora biti jednaka 1:

$$\sum_{i=1}^{N} p(i) = \sum_{i=1}^{N} \frac{2i}{N(N+1)} = \frac{2}{N(N+1)} \sum_{i=1}^{N} i = \frac{2}{N(N+1)} \cdot \frac{N(N+1)}{2} = 1.$$

Slika prikazuje raspodelu verovatnoće linearno rangirajuće selekcije. Tačke koje predstavljaju verovatnoću selekcije nalaze se na pravcu (odatle i naziv linearna selekcija). Pravac ima nagib 2/[*N*(*N*+1)]. Svojstvo selekcije je da *i*-ta jedinka ima *i* puta veću verovatnoću preživljavanja od najgore jedinke ranga 1.



Raspodela verovatnoće selekcije za linearno rangirajuću selekciju

Verovatnoća selekcije najbolje ili najlošije jedinke može biti unapred zadata. Izraz (5.1.7) ne uključuje mogućnost slobodnog izbora verovatnoće selekcije neke jedinke, jer prema tom izrazu verovatnoća selekcije zavisi samo od ranga jedinke i veličine populacije. Nagib pravca na kojem leže verovatnoće selekcija zavisi i od verovatnoće selekcije jedinke s oznakom 1 (ta jedinka je najlošija). Raspodela verovatnoće linearno rangirajuće selekcije ispunjava sledeća dva uslova:

- verovatnoće selekcije moraju ležati na pravcu, tj. $p(i)=a+bi$,
- suma verovatnoća selekcije svih jedinki mora biti jednaka 1: $\sum p(i)=1$.

Uzevši u obzir ta dva uslova mogu se izračunati vrednosti a i b . Konačno, opšti izraz za izračunavanje verovatnoće selekcije, koja zavisi od ranga jedinke i, veličine populacije N i verovatnoće selekcije jedinke s oznakom 1, glasi:

$$p(i)=\frac{N(N+1)\cdot p(1)-2}{N\cdot(N-1)}+\frac{2-2N\cdot p(1)}{N(N-1)}\cdot i \qquad (5.1.8)$$

Uvrsti li se za vrednost verovatnoće selekcije jedinke ranga 1 (to je najgora jedinka) $p(1) =2/[N(N+1)]$ u izraz (5.1.8) dobija se izraz (5.1.7) [1].
Verovatnoće selekcije za ovaj metod selekcije se takođe mogu računati podešavanjem samo jednog parametra; *stepen reprodukcije najgore prilagođene jedinke,* η^-, koji se nalazi u intervalu $[0,1]$. Svakoj jedinki i se tada dodeli verovatnoća izbora po sledećoj formuli:

$$p(i)=\frac{1}{N}\left(\eta^- +\left(\eta^+ -\eta^-\right)\cdot\frac{i-1}{N-1}\right). \qquad (5.1.9)$$

Veličina η^+ predstavlja stepen reprodukcije najbolje prilagođene jedinke u populaciji i on se može izraziti preko η^- ,tj. važi : $\eta^+ = 2-\eta^-$ [7].
Parametar η^- služi za podešavanje nagiba pravca u formuli (5.1.9).

Selekcije najboljih

Selekcije najboljih biraju unapred zadati broj najboljih jedinki. Moguće su tri vrste selekcije: $(\mu+\lambda)$ selekcija, (μ,λ) selekcija i selekcija isecanjem. Navedene podvrste selekcije najboljih se međusobno razlikuju prema skupu jedinki iz kojeg se biraju najbolje jedinke. Najbolje jedinke se mogu birati iz skupa samo roditelja, roditelja i dece i samo dece. Prvo se biraju slučajnim postupkom roditelji, a zatim se njihova deca (zajedno ili bez roditelja) sortiraju i preživljava samo određeni broj najboljih jedinki. μ je veličina populacije roditelja, a λ je veličina populacije dece [21, 24].

Kod $(\mu+\lambda)$ selekcije slučajno se bira μ roditelja iz populacije. Njihovim se ukrštanjem stvara λ dece. Iz skupa roditelja i dece se zatim najboljih μ jedinki selektuje za sledeću generaciju. Postupak se ponavlja sve dok se ne popuni nova generacija s N novih jedinki, odnosno N/μ puta ili se μ jedinki višekratno kopira kako bi se popunila populacija.

Pri (μ,λ) selekciji takođe se slučajno bira iz populacije μ roditelja i njihovim ukrštanjem se stvara λ djece. Međutim, dece je više od roditelja $(\lambda \geq \mu)$ i μ najbolje dece se selektuje za sledeću generaciju [22, 23]. Verovatnoća selekcije i-te jedinke za ukrštanje iznosi:

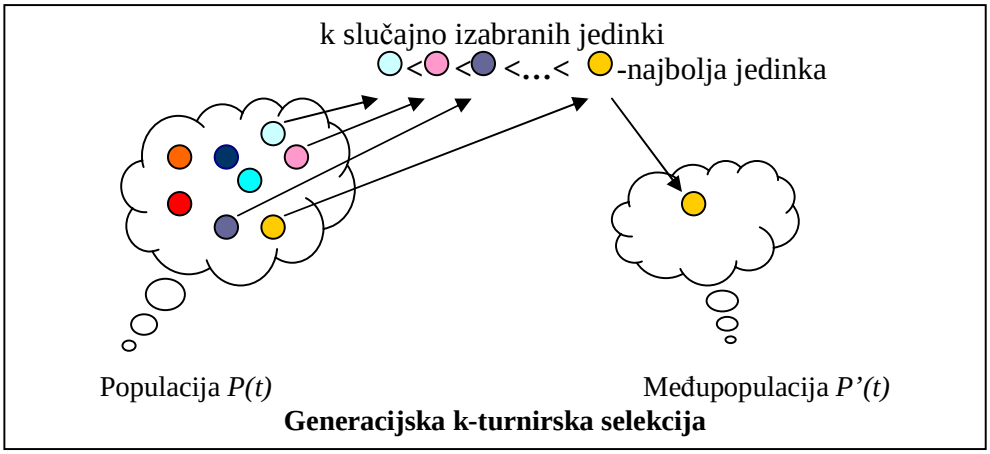
$$p(i)=\begin{cases}1/\mu & : \quad 1\leq i\leq \mu \\ 0 & : \quad \mu < i \leq N\end{cases}$$

Obe opisane selekcije kombinuju rekombinaciju sa slučajnim izborom jedinki, tj. u istom koraku se obavljaju i selekcija i ukrštanje [26]. Roditelji se ne menjaju kod postupka ukrštanja, nego se samo čitaju njihove vrednosti. Međutim, treba obaviti sortiranje i roditelja i dece kod $(\mu+\lambda)$ selekcije ili samo dece kod (μ,λ) selekcije.

Selekcija isecanjem bira n najboljih jedinki i kopira ih N/n puta [26, 29]. Verovatnoća selekcije i-te jedinke u bazen za parenje iznosi: $p(i)=1$, za $i=1,2,...,n$, dok je za ostale jedinke ta verovatnoća jednaka nuli.

k-turnirske selekcije

k-turnirska selekcija (k=2,3,4,...,N) s jednakom verovatnoćom bira k jedinki između kojih selektuje najbolju ili najlošiju jedinku [23, 25, 26, 28, 31]. Generacijska turnirska selekcija N puta slučajno bira k jedinki, između njih selektuje najbolju jedinku i kopira je u međupopulaciju.



Generacijska k-turnirska selekcija

Eliminacijska turnirska selekcija M puta bira *k* jedinki i eliminiše najlošiju među njima. Parametar *k* naziva se veličinom turnira. Prema veličini turnira, turnirske selekcije se dele na binarnu turnirsku selekciju (*k* =2), 3-turnirsku selekciju, 4-turnirsku selekciju, itd.
Za turnirsku selekciju nije potrebno sortirati jedinke, iako verovatnoća selekcije zavisi od ranga jedinke, kao i kod sortirajucih selekcija. U postupku turnirske selekcije važan je samo međusobni odnos između *k* slučajno odabranih jedinki. Prema vrednosti funkcije prilagođenosti određuje se za svaki par jedinki koja je od njih bolja, odnosno lošija.
Neka najbolja jedinka ima oznaku 1, a najgora N. Najbolja jedinka se ne može nikako eliminisati s bilo kojom eliminacijskom turnirskom selekcijom, jer će *k*-1 ostalih slučajno odabranih jedinki biti sigurno lošije od najbolje jedinke. Čuvanje najbolje jedinke ili više najboljih jedinki naziva se *elitizmom*. Elitizam je kod eliminacijskih turnirskih selekcija inherentno ugrađen, tj. verovatnoća eliminacije najbolje jedinke jednaka je nuli: p(1)=0.

Nezavisno jedni od drugih, Baeck [23] , Miller i Goldberg [28] su odredili intenzitet selekcije za *k*-turnirsku selekciju. Vrednosti intenziteta selekcije za *k*-turnirsku generacijsku selekciju, gdje je *k* =1,2,3,4,5, prikazane su u tablici [23, 28]:

k	s_I
1	0
2	0.56419
3	0.84628
4	1.02938
5	1.16296

Intenzitet selekcije za k-turnirsku generacijsku selekciju

Neka se, na primer, koristi samo 3-turnirska generacijska selekcija (bez ostalih genetskih operatora). Uz pretpostavku da u generaciji *t* vrednosti funkcija prilagođenosti imaju normalnu raspodelu N[$\overline{f}(t), \sigma(t)$] može se predvideti prosečna prilagođenost populacije u sledećoj generaciji (to je prosečna prilagođenost preživelih, odnosno selektovanih jedinki). $\overline{f_p}(t+1)$ prema izrazima (5.2.1), (5.2.4) i navedenoj tablici je:

$$\overline{f_p}(t+1) = \overline{f}(t) + s_I \cdot \sigma(t) = \overline{f}(t) + 0.84628 \cdot \sigma(t).$$

Jednostavna turnirska selekcija

Jednostavna turnirska selekcija je specifičan oblik eliminacijske binarne turnirske selekcije prilagođene paralelnom izvođenju [1]. U jednom koraku slučajnim postupkom se biraju dva para jedinki. U svakom paru se lošija jedinka selektuje za eliminaciju. Ukrštanjem preživelih jedinki generiše se dvoje dece, koja se potom mutiraju i evaluiraju(određuje se njihova prilagođenost). Taj par novostvorenih jedinki nadomešćuje eliminisane jedinke. Na taj način genetski algoritam s jednostavnom turnirskom selekcijom u istom koraku obavlja i selekciju i reprodukciju.

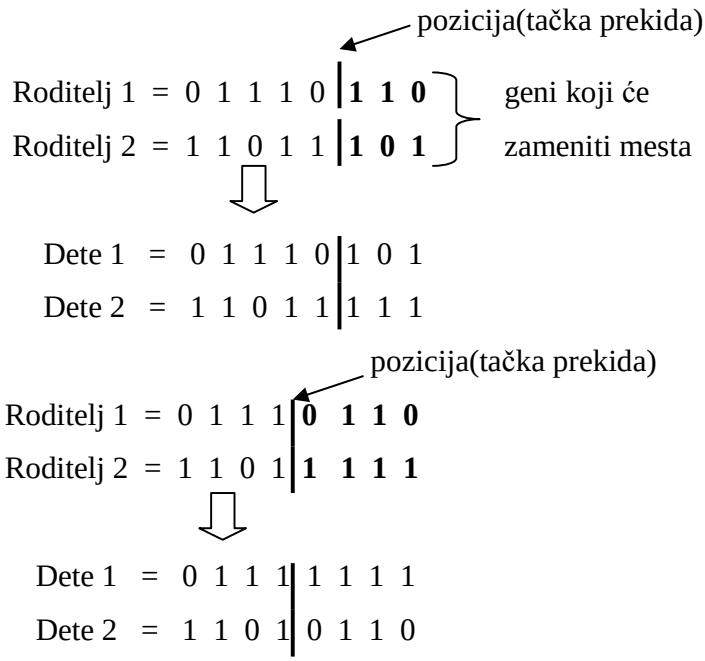
Izbor operatora selekcije svodi se na izbor selekcije koja je najpogodnija za paralelno izvođenje. Poželjno je da selekcija, osim što se na jednostavan način može paralelizovati, ima mogućnost podešavanja intenziteta selekcije. Kako je to već opisano, postupci selekcije se dele na generacijske i eliminacijske. Generacijske selekcije nisu pogodne za paralelizaciju, jer se reprodukcija mora obavljati odvojeno od selekcije. Reprodukcija može početi tek kada generacijska selekcija formira novu populaciju. Generacijska selekcija troši više memorijskog prostora jer algoritam obavlja posao nad dvema populacijama. Osim toga, generacijskom selekcijom se generišu duplikati. Duplikate je poželjno nakon reprodukcije eliminisati, što dodatno usporava rad genetskog algoritma. Zbog toga je eliminacijska selekcija pogodnija za paralelno izvođenje.
Postupci selekcije se dele i prema tome kako funkcija cilja utiče na verovatnoću selekcije. Ako funkcija cilja direktno utiče na verovatnoću selekcije, tako da je verovatnoća selekcije proporcionalna vrednosti funkcije cilja, radi se o proporcionalnim selekcijama. Između brojnih nedostataka proporcionalnih selekcija, koji su već navedeni, može se zaključiti da selekciju pogodnu za paralelno izvođenje treba tražiti među rangirajućim selekcijama. Mnogi autori [4, 26, 30, 31] se slažu da su eliminacijske turnirske selekcije najpogodnije za paralelno izvođenje.

5.1.6. Ukrštanje (crossover)

Operator ukrštanja (crossover operator) je osnovni operator za generisanje novih hromozoma u genetskim algoritmima. Operatorom ukrštanja proizvode se nove jedinke koje nasleđuju genetski materijal od oba roditelja. To i jeste najvažnija osobina ukrštanja jer ako su roditelji dobri (prošli su proces selekcije), tada će najverovatnije i dete biti dobro, ako ne i bolje od svojih roditelja.
Dakle, ukrštanje je binarni operator u kome učestvuju dve jedinke koje nazivamo *roditeljima*. Ukrštanjem nastaju jedna ili dve nove jedinke koje nazivamo *deca*. Postoji više vrsta operatora ukršanja od kojih će u ovom radu biti obrađene najfrenkventnije: ukrštanje sa jednom tačkom prekida, ukrštanje sa više tačaka prekida, uniformno ukrštanje i mešajuće ukrštanje.

Ukrštanje sa jednom tačkom prekida
(single-point crossover)

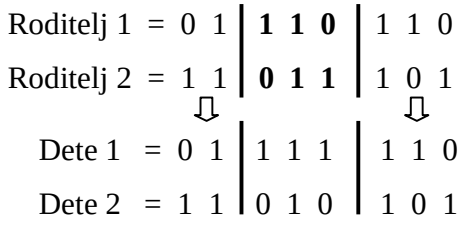
Ovaj operator se još naziva i jednopozicioni, i predstavlja najprostiji oblik ukrštanja. Operatorom sa jednom tačkom prekida se deca dobijaju tako što geni kod oba roditelja zamene mesta od nasumice izabrane pozicije u hromozomima. Dva primera ovog operatora mogu se videti na slici, na primeru hromozoma predstavljenih u binarnom kodu.



Sa slike se vidi da je dete 1 dobijeno prepisivanjem prvog dela hromozoma roditelja 1 do vertikalne linije i zamenom drugog dela hromozoma sa genima koji odgovaraju roditelju 2**.** Bitna osobina operatora ukrštanja sa jednom tačkom prekida je da se mogu proizvesti deca koja su potpuno drugačija od svojih roditelja. Druga bitna osobina jednopozicionih operatora je da on neće uneti izmene u genima na čijim pozicijama oba roditelja imaju iste vrednosti. To se najbolje vidi na drugom primeru, gde deca na pozicijama 2 i 3 od vertikalne linije imaju istu vrednost kao i njihovi roditelji iako je operator primenjen.
Operator ukrštanja sa jednom tačkom prekida je povoljniji za korišćenje na većim populacijama, jer kada populacija postane homogena, prostor pretraživanja se smanjuje [5].

Ukrštanje sa više tačaka prekida
(multi-point crossover)

Kod ovog operatora, koji se još naziva i višepozocioni, broj tačaka i pozicija prekida se bira slučajno za svako pojedinčno ukrštanje. Kod, na primer, dvopozicionog operatora ukrštanja, biraju se dve pozicije kod oba roditelja, tako da samo deo između njih bude zamenjen, dok početni i krajnji deo hromozoma ostaju neizmenjeni. To ukrštanje izgleda ovako:



Osnovna ideja višepozicionih operatora je da delovi hromozoma koji mu daju dobre performanse možda nisu sadržani baš u susednim podnizovima tog hromozoma. Takođe, višepozocioni operatori omogućavaju robustnije pretraživanje prostora rešenja, čime se sprečava prerana konvergencija u ranim fazama algoritma [5].

Uniformno ukrštanje
(Uniform crossover)

Uniformno ukrštanje je ukrštanje sa *b*-1 prekidnih tačaka (*b* – broj bitova). Verovatnoća da dete nasledi svojstvo jednog roditelja je 0.5 tj. jednaka je verovatnoći nasleđivanja svojstva od oba roditelja. Operator uniformnog ukrštanja funkcioniše na sledeći način: formira se maska ukrštanja, koja je iste dužine kao i hromozom. Bitovi u masci se postavljaju nasumice, i oni određuju da li će odgovarajući gen biti nasleđen od prvog ili drugog roditelja. 1 znači preuzimanje gena od prvog roditelja, a 0 preuzimanje odgovarajućeg gena od drugog roditelja [5].

Roditelj 1 = 0 1 0 1 0 1 1 0

Roditelj 2 = 1 1 1 0 1 0 0 1

Maska = 1 1 0 0 1 0 0 1

Dete 1 nasleđuje gen od prvog roditelja

Dete 1 = 0 1 1 0 0 0 0 0

Dete 2 = 1 1 0 1 1 1 1 1

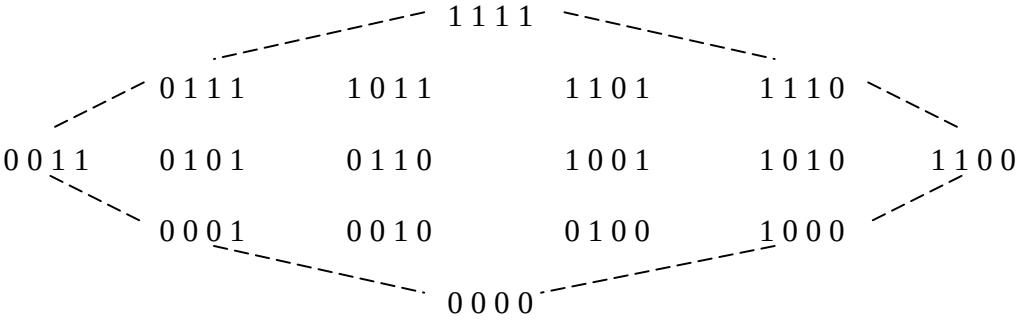
U slučaju drugog potomka, deteta 2, koristi se inverzija prvobitne maske ili zamena roditelja 1 i 2.
Spears i DeYong su pokazali da se uniformno ukrštanje može parametrizovati primenom verovatnoće zamene gena [1]. Na primer,ako je p=0.3 tada je verovatnoća da će jedan gen biti nasleđen od prvog roditelja 30%, a od drugog 70%. Ako se verovatnoća zamene gena razlikuje za pojedine gene, tada se takvo uniformno ukrštanje naziva *p-uniformno ukrštanje*. Tada se zadaje maska koja posebno definiše koja je verovatnoća nasleđivanja za svaki gen.

gen	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**....**	**b-3**	**b-2**	**b-1**	**b**
p	0.1	0.5	0.9	0.1	1	1	0.5	0.1	0	....	0.7	0.7	0.6	0.1

Primer maske koja određuje verovatnoću biranja gena

Tablica pokazuje da se geni 5 i 6 uzimaju od prvog roditelja. Gen sa brojem 9 se kopira od drugog roditelja a gen 1 će najverovatnije (90% verovatnoće) biti od drugog roditelja.

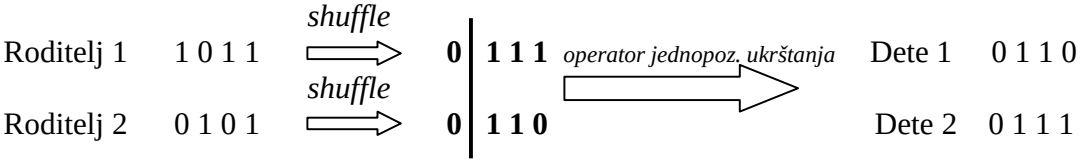
Uniformno ukrštanje pretražuje veliki prostor rešenja zbog toga se ono češće koristi na malim populacijama [5]. Prostor koji pretražuje je jednopozicioni; dvopozicioni i uniformni operator može se videti na primerima hromozoma datog u binarnom kodu. Neka su roditelji hromozomi dižine 4 bita, 0000 i 1111. Jednopozicionim operatorom ukrštanja se od roditelja 0000 i 1111 mogu dobiti sledeći potomci: 1000, 0111, 1100, 0011, 1110, 0001. Dvopozicionim operatorom ukrštanja se od istih roditelja mogu dobiti: 1011, 1101, 0110, 1001, 0010, 0100. Uniformnim operatorom ukrštanja se mogu dobiti svi gore navedeni potomci i još dva: 0101 i 1010.



Na slici je prikazan prostor rešenja koji se dobija pomoću sva tri operatora; isprekidanim linijama su povezana rešenja dobijena jednopozicionim ukrštanjem

Mešajuće ukrštanje
(shuffle crossover)

Mešajuće ukrštanje obavlja se u tri koraka. U prvom koraku izmešaju se bitovi svakom roditelju. Drugi korak je klasično ukrštanje sa jednom ili više tačaka prekida. I konačno, u trećem koraku izmešani bitovi kod roditelja vrate se na stara mesta (roditelji ostaju ne promenjeni) [1].



Na svaki par jedinki, sa zadatom verovatnoćom p_c primenjuje se operator ukrštanja. Primena operatora sa verovatnoćom znači da na neke jedinke operator neće biti primenjen i one će dobiti šansu da ne promenjene pređu u sledeću generaciju. Za svaki par jedinki odabere se slučajan broj r iz intervala [0,1] i ako je on manji od verovatnoće ukrštanja p_c , vrši se ukrštanje. Verovatnoća p_c (koja najčešće iznosi 0.9 za populaciju od

30 jedinki, a 0.6 za 100 jedinki) predstavlja okvirni procenat jedinki koje će proizvesti potomke i time dati mogućnost da se u tekućoj generaciji pronađe novo najbolje rešenje.

Osnovni cilj operatora ukrštanja treba da bude očuvanje dobrih osobina trenutno najkvalitetnijih rešenja. Postoje i druge definicije operatora ukrštanja, a konkretan izbor najčešće zavisi od samog problema koji treba rešiti kao i od izbora kodiranja za rešenje.

Jedan konkretan primer operatora ukrštanja je *delimično mapirano ukrštanje* PMX (eng. partially–mapped crossover) koji se koristi kod problema trgovačkog putnika (TSP) [3]. PMX operator vrši ukrštanje na isti način kao dvopozicioni operator, pri čemu se poštuju pravila mapa nastalih tim postupkom ukrštanja.
Neka imamo sledeći redosled obilaska gradova:

1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9

i predstavimo ga u obliku permutacije sa istim redosledom gradova kao i gore

(1 2 3 4 5 6 7 8 9).

Uzmimo da je to roditelj 1, a **(4 5 2 1 8 7 6 9 3)** da je roditelj 2. Ukrštanjem pomoću dvopozicionog operatora dobijamo sledeće:

Roditelj 1= (1 2 3 | 4 5 6 7 | 8 9)
Roditelj 2= (4 5 2 | 1 8 7 6 | 9 3)

Dete 1 = (x x x | 1 8 7 6 | x x)
Dete 2 = (x x x | 4 5 6 7 | x x)

4 ⟷ 1
5 ⟷ 8
6 ⟷ 7
7 ⟷ 6

} mapiranje

Sadržaj između vertikalnih linija je zamenjen, a prvi i poslednji segment kod dece će zavisiti od tako nastalog mapiranja. Kako gradovi 2,3 i 9 nisu obuhvaćeni ovim mapama, njih možemo direktno prepisati sa roditelja na potomke kako se to inače i radi kod dvopozicionog operatora.

Dete 1 = (x 2 3 | 1 8 7 6 | x 9)
Dete 2 = (x x 2 | 4 5 6 7 | 9 3)

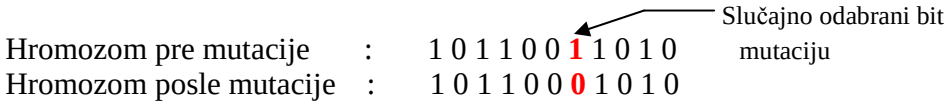
Konačno, prvi x kod prvog deteta (koji bi trebao da bude 1) zbog mapiranja 1 ⟷ 4 će biti zamenjen sa 4. Slično drugi x kod prvog deteta će biti zamenjen sa 5 zbog mapiranja 8 ⟷ 5 . Iz istog razloga će kod drugog deteta prvi x biti zamenjen 1 a drugi sa 8. Konačan izgled potomaka je sledeći:

Dete 1 = (4 2 3 | 1 8 7 6 | 5 9)

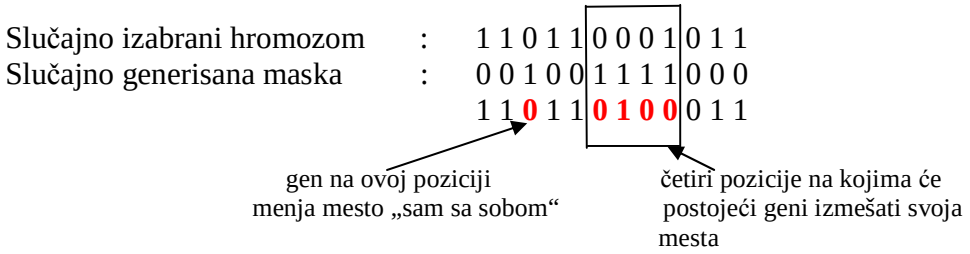
Dete 2 = (1 8 2 | 4 5 6 7 | 9 3) .

5.1.7. Mutacija

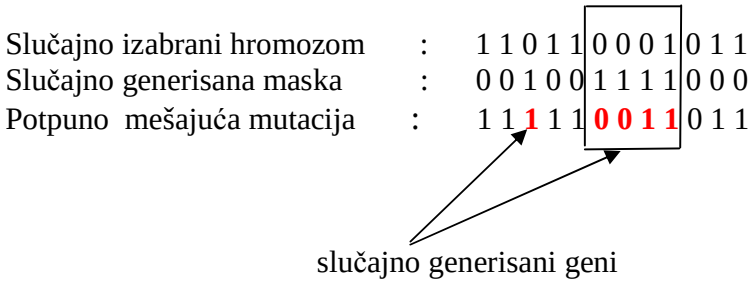
Drugi operator koji je karakterističan za GA je mutacija ili slučajna promena jednog ili više gena. Mutacija je unarni operator jer deluje nad jednom jedinkom. Rezultat mutacije je izmenjena jedinka.
Parametar koji određuje verovatnoću mutacije jednog gena (bita) je ujedno i parametar algoritma. Taj parametar, p_m , se najčešće nalazi u granicama od 0.001 za populaciju od 100 jedinki, do 0.01 za populaciju od 30 jedinki. Ako verovatnoća mutacije teži jedinici, tada se algoritam pretvara u algoritam slučajnog pretraživanja prostora rešenja. S druge strane, ako verovatnoća mutacije teži nuli , postupak će najverovatnije već u početku procesa stati u nekom lokalnom optimumu.
Jednostavna ili **prosta mutacija** svaki bit hromozoma menja sa jednakom verovatnoćom p_m.



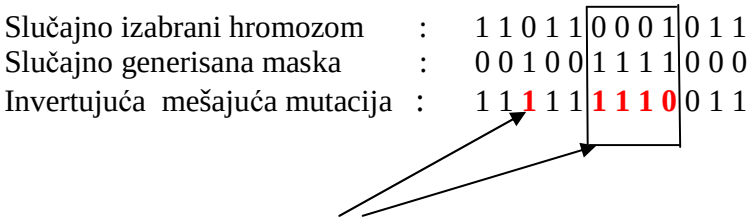
Postoje i složenije vrste mutacija, u kojima se slučajnim postupkom bira hromozom (a ne gen) za mutaciju. **Mešajuća mutacija** je vrsta mutacije koja slučajnim postupkom bira hromozom za mutaciju i masku i onda izmeša gene [6].



Potpuno mešajuća mutacija je vrsta mešajuće mutacije gde se na određenim mestima koja su definisana maskom, geni umesto mešanja slučajno generišu [6].



Invertujuća mešajuća mutacija je takođe vrsta mešajuće mutacije, s tim što se kod nje geni određeni maskom invertuju.



Invertovane vrednosti slučajno izabranog hromozoma

Za druge vrste kodiranja koje nisu binarne, mutacija se može postići malim pomeranjem vrednosti gena ili slučajnom selekcijom nove vrednosti iz dozvoljenih granica. Naučnici Wright, Janikow i Michalewicz [2] su demonstrirali kako realno kodirani genetski algoritmi mogu bolje iskoristiti višu stopu mutacije od binarno kodiranih genetskih algoritama povećavajući na taj način prostor mogućih rešenja bez suprotnog efekta na karakteristike konvergencije.
Mutacijom se pretražuje prostor rešenja i upravo je mutacija mehanizam za izbegavanje lokalnih minimuma. Naime, ako cela populacija završi u nekom od lokalnih minimuma jedino slučajnim pretraživanjem prostora rešenja pronalazi se bolje rešenje. Dovoljno je da jedna jedinka nastala mutacijom bude bolja od ostalih, pa da se u nekoliko sledećih generacija sve jedinke „presele“ u prostor gde se nalazi bolje rešenje.
Uloga mutacije je takođe u obnavljanju izgubljenog genetskog materijala. Dogodi li se, na primer, da sve jedinke populacije imaju isti gen na određenom mestu u hromozomu, samo ukrštanjem se taj gen ne bi mogao promeniti. Ako je reč o binarnom kodiranju, time je izgubljeno čak pola prostora pretraživanja [20].

Na primeru minimizacije Rozenbrokove funkcije, koji je korišćen da bi se pobliže objasnile VNS i TS metaheuristike, biće prikazan i jedan od načina na koji funkcionišu genetski algoritmi.
Kako je prostor pretraživanja kod Rozenbrokove funkcije sužen radi lakšeg prikaza koraka algoritama, taj isti prostor rešenja će biti iskorišćen i kod prikaza rada GA.
Sve tačke tog prostora su predstavljene kao realni brojevi, koji čine ekvidistantnu mrežu. Tako će i hromozomi biti predstavljeni kao uređeni parovi tačaka, gde prvi gen u hromozomu predstavlja promenljivu x_1, a drugi gen promenljivu x_2 . Hromozomi su označeni sa S, i za inicijalnu populaciju bira se pet slučajno odabranih hromozoma iz različitih ˝područja˝ prostora rešenja. Slučajno izabrana inicijalna populacija data je u tabeli, kao i vrednosti fittness funkcije za svaki od odabranih hromozoma.

Hromozomi S_i	(x_1, x_2)	$100(x_1^2 - x_2)^2 + (1 - x_1)^2$
S_1	(-1.436 , - 1.640)	1376.486
S_2	(1.012 , - 1.028)	421.130
S_3	(1.216 , 0.808)	45.025
S_4	(1.624 , 1.420)	148.590
S_5	(-0.008 , 0.400)	17.011

Za fitness funkciju u ovom primeru je odabrana upravo Rozenbrokova funkcija koja se optimizira. Kako su niže vrednosti fitness-a poželjnije jer se traži minimum te funkcije, potrebno je formirati sistem po kome će hromozomi sa manjim fitness vrednostima biti izabrani za roditelje sa većom verovatnoćom od onih hromozoma sa većim vrednostima fitness funkcije. Zbog toga je potrebno prvo izračunati procenat(verovatnoća) sa kojim će svaki od hromozoma biti izabran.
Jedna od opcija je da se uzme suma recipročnih vrednosti fitness-a, i da se uz pomoć tako dobijenog koeficijenta izračuna verovatnoća izbora svakog od hromozoma:

$$\sum \frac{1}{f_i} = 1/1376.486 + 1/421.130 + 1/45.025 + 1/148.590 + 1/17.011 = 0.0908$$

Hromozomi S_i	Verovatnoća izbora	
S_1	1/1376.486/0.0908= 0.0080	0.80%
S_2	1/421.130/0.0908 = 0.0262	2.62%
S_3	1/45.025/0.0908 = 0.2446	24.46%
S_4	1/148.590/0.0908 = 0.0741	7.41%
S_5	1/17.011/0.0908 = 0.6471	64.71%

Sledeći korak GA je selekcija. U ovom primeru će biti korišćena rulet selekcija. Rulet selekcija u svakom koraku generiše slučajni broj *r* u intervalu [0,1], tako da raspodela verovatnoće na pojedine hromozome iz ovog primera dobija sledeći, uprošćen, oblik:

S_1 , $0 \leq r < 0.0080$
S_2 , $0.0080 \leq r < 0.0342$
S_3 , $0.0342 \leq r < 0.2788$
S_4 , $0.2788 \leq r < 0.3529$
S_5 , $0.3529 \leq r < 1$

To znači da će biti odabran za selekciju onaj hromozom u čijem odgovarajućem intervalu se nađe slučajan broj *r*. Na primer, ako je generisan *r* =0.456, biće odabrana jedinka S_5, jer se broj *r* nalazi u intervalu $0.3529 \leq 0.456 < 1$. Proporcionalna selekcija će u ovom primeru slučajno odabrati broj *r* 10 puta, pri čemu je moguće da će neke jedinke biti više puta odabrane. Tako će biti odabrano 5 parova roditelja, i svaki par će ukrštanjem dati po jednog potomka. Ti potomci će činiti novu populaciju.
Neka su slučajni odabiri broja *r* u 10 serija, sledeći (za izbor slučajnog broja *r* iz intervala [0,1] u primeru je korišćen računar, i generator slučajnog broja):
0.429 , 0.126 , 0.495 , 0.528 , 0.323 , 0.561 , 0.495 , 0.627 , 0.227 , 0.315 .
To znači da će sledeće jedinke biti odabrane:
S_5 , S_3 , S_5 , S_5 , S_4 , S_5 , S_5 , S_5 , S_3 , S_4 .

Operator ukrštanja će biti primenjen redom, na sledeće parove jedinki:

Roditelj 1	S_5	S_5	S_4	S_5	S_3
Roditelj 2	S_3	S_5	S_5	S_5	S_4

Potomci svakog od ovih roditelja treba da sadrže genetske osobine oba roditelja. Načini kojima se to postiže su brojni i zavise od definicije operatora ukrštanja. U ovom primeru će operator ukrštanja biti definisan na sledeći način: potomak će za prvi (drugi) gen imati srednju vrednost prvih(drugih) gena oba roditelja. U slučaju da novodobijena vrednost gena potomka odstupa od vrednosti koje se nalaze u prostoru mogućih rešenja, za taj gen se uzima najbliža vrednost koja tom prostoru pripada, tj. vrednost koja se razlikuje za $\pm k/2$ (k korak ekvidistantne mreže) od vrednosti dobijene ukrštanjem. Pošto će se skoro uvek pojaviti dve vrednosti koje u tom slučaju gen potomka može da preuzme, biraće se ona koja tom potomku omogućava nižu vrednost fitness funkcije. To znači da ako je srednja vrednost prvog gena oba roditelja npr. 0.502, a drugog 0.808, za vrednost prvog gena potomka uzima se jedna od vrednosti 0.400 ili 0.604 jer se 0.502 ne nalazi u prostoru rešenja. Tako potomak može biti (0.400,0.808) ili (0.604,0.808). Kako je vrednost fitness-a potomka (0.604,0.808) manja, taj potomak ulazi u novu generaciju. Na taj način se održava stalan broj jedinki u generaciji, i omogućava se potomcima ostalih parova roditelja da u njoj učestvuju.
Za odabrane roditelje dobijaju se sledeći potomci:

Roditelj 1	Roditelj 2	Potomci
S_5	S_3	**S_1** (0.604 , 0.604)
S_5	S_5	**S_2** (-0.008 , 0.400)
S_4	S_5	**S_3** (0.808 , 0.808)
S_5	S_5	**S_4** (-0.008 , 0.400)
S_3	S_4	**S_5** (1.420 , 1.216)

Kod trećeg i petog potomka, drugi geni su imali vrednost 0.91 i 1.114 pa su zamenjeni sa jednom od dve najbliže vrednosti koja daje nižu vrednost fitness-a. Takođe se vidi da drugi i četvrti potomak imaju iste gene, koji se samo ukrštanjem ne bi mogli promeniti. Na jedan od njih biće primenjena mutacija kako se u narednim generacijama ne bi izgubio deo prostora pretraživanja. Mutacija će biti realizovana slučajnim pomeranjem vrednosti oba gena odgovarajućeg hromozoma za $+k$ ili $-k$, gde je k=0.204 korak ekvidistantne mreže po kojoj je formiran prostor mogućih rešenja. Četvrti potomak, **S_4**, će tako u ovom primeru biti promenjen po oba gena, za na primer +0.204, tako da će posle mutacije vrednosti njegovih gena biti (0.196 , 0.604).
Nova generacija je formirana i njihove vrednosti fitness-a su date u tabeli:

Potomci	f_i (*fitness*)
S_1 (0.604 , 0.604)	5.878
S_2 (-0.008 , 0.400)	17.011
S_3 (0.808 , 0.808)	2.444
S_4 (0.196 , 0.604)	32.635
S_5 (1.420 , 1.216)	64.240

U novoj generaciji najbolja jedinka, **S_3**, dovodi pretraživanje u lokalni minimum 2.444.
Prosečna vrednost fitness-a prve generacije je 401.65, a već u ovoj generaciji ta vrednost je znatno manja i iznosi 24.44, što je dobar pokazatelj kvaliteta potomaka koji su dobijeni.

Postupak se sada ponavlja od početka. Izračunava se suma recipročnih vrednosti fitness-a hromozoma iz nove generacije. Ona u ovoj generaciji iznosi 0.6843. Na osnovu nje izračunava se istim postupkom kao i pre verovatnoća selekcije za svaki hromozom, a zatim se pravi odgovarajuća raspodela i bira novih pet parova roditelja uz pomoć slučajno izabranog broja *r* iz intervala [0,1]. Za ovu generaciju, zavisnost izbora određene jedinke od broja *r* i izračunate verovatnoće selekcije izgleda ovako:

$\mathbf{S_1}$, $0 \leq r < 0.2486$
$\mathbf{S_2}$, $0.2486 \leq r < 0.3345$
$\mathbf{S_3}$, $0.3345 \leq r < 0.9324$
$\mathbf{S_4}$, $0.9324 \leq r < 0.9772$
$\mathbf{S_5}$, $0.9772 \leq r < 1$

Najveću verovatnoću selekcije ima jedinka $\mathbf{S_3}$, 59.79%, zatim jedinka $\mathbf{S_1}$, 24.86%, a zatim jedinke $\mathbf{S_2}$, $\mathbf{S_4}$, $\mathbf{S_5}$ sa verovatnoćama selekcije redom 8.59%, 4.48% i 2.27%.
Neka je slučajan broj *r* izabran deset puta na sledeći način:

0.975 0.643 0.159 0.439 0.012 0.996 0.231 0.807 0.529 0.042.

Kako prva dva broja određuju prvi par roditelja za ukrštanje, i tako redom, dobijamo sledeće potomke:

Roditelj 1	Roditelj 2	Potomci	f_i
$\mathbf{S_4}$	$\mathbf{S_3}$	$\boldsymbol{S_1}$ (0.604 , 0.604)	5.878
$\mathbf{S_1}$	$\mathbf{S_3}$	$\boldsymbol{S_2}$ (0.808 , 0.604)	0.276
$\mathbf{S_1}$	$\mathbf{S_5}$	$\boldsymbol{S_3}$ (1.012 , 1.012)	0.015
$\mathbf{S_1}$	$\mathbf{S_3}$	$\boldsymbol{S_4}$ (1.012 , 0.808)	4.672
$\mathbf{S_3}$	$\mathbf{S_1}$	$\boldsymbol{S_5}$ (0.604 , 0.400)	0.281

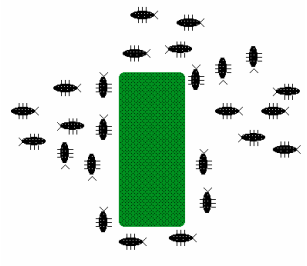
Prilikom ukrštanja roditelja $\mathbf{S_4}$ i $\mathbf{S_3}$, pojavila su se čak četiri moguća potomka zbog odstupanja vrednosti oba gena potomka od definisanog prostora rešenja. Naime operatorom ukrštanja, za gene potomka dobijene su vrednosti (0.502,0.706) koje se ne nalaze u prostoru rešenja. To znači da prvi gen potomka treba da promeni vrednost u 0.400 ili 0.604, a drugi gen u 0.604 ili 0.808. Od ova četiri moguća potomka najmanju vrednost fitness funkcije ima potomak (0.604,0.604) i on će ući u narednu generaciju a ostala tri neće.
Takođe, tri puta se primenjuje ukrštanje na iste roditelje $\mathbf{S_1}$ i $\mathbf{S_3}$, i dobijaju se tri ista potomka. Na dva od njih je iz tog razloga primenjena mutacija. Jedan potomak mutira sa pomeranjem vrednosti oba gena za $+k$, a drugi sa pomeranjem vrednosti oba gena za $-k$.
Potomci koji su dobijeni navedenim ukrštanjem i mutacijom nalaze se u gornjoj tabeli.
Najbolji potomak iz ove generacije je $\boldsymbol{S_3}$ (1.012 , 1.012) čija je vrednost fitness funkcije 0.015, što jeste globalni minimum u ovako definisanom prostoru rešenja Rozenbrokove funkcije. Treba napomenuti da se operator ukrštanja i mutacije mogu definisati na razne druge načine, i da za veličinu i izbor inicijalne populacije postoje mnoge druge opcije. Treba definisati i kriterijum zaustavljanja algoritma, jer se ne može uvek ovako brzo doći do globalnog minimuma.

5.2. Mravlji algoritmi

Ponašanje insekata najviše proučavaju biolozi, ali u novije vreme sve više i naučnici drugih usmerenja. Na primer, istraživanja kolektivnog ponašanja insekata omogućila su naučnicima iz oblasti računarskih nauka da razviju moćne metode i algoritme za distribuirano upravljanje i optimizaciju. Pored tzv. statičkih, ovi metodi i algoritmi su fleksibilni i robustni u dinamičkim okruženjima kao što su saobraćaj na Internetu ili standardna telefonija. Radi se o problemima koji spadaju u klasu teško rešivih zbog ogromnih prostora mogućih rešenja koje treba pretražiti u razumnom vremenu pomoću jakih računara.
Kretanje mrava i pčela inspirisali su razvoj nekih graničnih oblasti moderne nauke kao što je, na primer, kolektivna inteligencija. U poslednjoj deceniji profilisana je nova oblast optimizacije za koju na srpskom jeziku ne postoji odomaćen naziv. Radi se o Ant Colony Optimization (ACO), ili u radnom prevodu ‘Optimizaciji pomoću mravljih kolonija (OMK)’, oblasti koju karakterišu specifični optimizacioni algoritmi inspirisani ponašanjem prirodnih mravljih kolonija. Ove procedure optimizacije se zasnivaju na indirektnoj komunikaciji i saradnji više agenata, slično kako u prirodi komuniciraju i ponašaju se neke vrste mrava [32]. Mravlji algoritmi (MA) i sistemi mravljih kolonija (SMK) su osnovni algoritmi OMK, a svi zajedno spadaju u oblast veštačke inteligencije zbog toga što: (1) na konceptualnom nivou sadrže jaku komponentu samoučenja i (2) rešavaju tzv. NP-teške i NP-kompletne kombinatorne probleme.

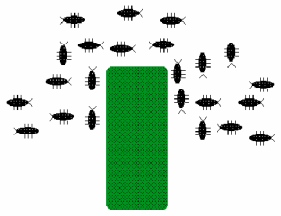
Mravlje kolonije

Dok se kreću (ili trče), pravi mravi polažu na tle određenu količinu hemijske supstance poznate kao feromon. Istraživanja biologa i inženjera su pokazala da svaki mrav sa određenom verovatnoćom preferira praćenje putanje (traga) na kojoj postoji deblji (bogatiji) sloj feromona. Ova jednostavna činjenica objašnjava zašto su mravi sposobni da se prilagode promenama u svom okruženju, kao što je npr. pojava neke prepreke na njihovom putu. Pitanje koje otvara diskusiju jeste: ‘Šta se događa kada kolona mrava koja se kreće najkraćim putem između staništa i hrane, bude sprečena da nastavi takvo kretanje zato što se negde na putanji pojavila prepreka (slika 1) ?’



Slika 1. Prepreka na putanji kolone mrava

Ono što se realno događa je da kada mravi naiđu na prepreku, oni slučajno biraju način da je zaobiđu tako što kreću levo, desno, gore ili dole. Zbog pojednostavljenja koje ne smanjuje opštost daljih izlaganja, pretpostavimo da mravi jedino mogu levo ili desno u odnosu na prepreku. Sa velikom sigurnošću se može reći da će približno polovina mrava u početku ići na jednu, a druga polovina na drugu stranu. Mravi koji krenu kraćim putem, brže će formirati jači trag feromona nego oni koji idu dužim putem. To će dalje izazvati da sve više i više mrava kreće za jačim feromonskim tragom dok konačno svi mravi ne nađu taj kraći put (slika 2). Kako se na kraćem putu feromoni ostavljaju ćešće, mravi kroz niz iteracija biraju kraći put. Pošto se mravi koji prate najkraći put prvi i vraćaju odakle su krenuli (gnezdo ili hrana), njihov feromonski trag postepeno nestaje usled isparavanja, ali ga novi mravi istovremeno i pojačavaju. Treba napomenuti da iako jači trag privlači više mrava koji taj trag još više pojačavaju, uvek ima mrava-solista koji u međuvremenu idu svojim putem bez obzira na trag i tako istražuju nove puteve. Kada prepreka na putu izazove saobraćajnu gužvu, mravi su spremni da slede rezervne pravce. Pravci koji se manje koriste konačno se eliminišu pošto feromon koji je na njih položen jednostavno ispari.



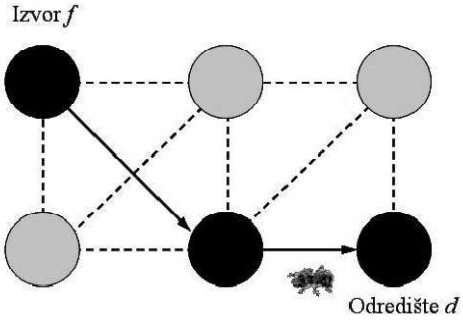
Slika 2. Kolona mrava koja je našla najkraći put

Novija istraživanja su dokazala sposobnost mrava da brzo nađu sledeći najkraći put kada prepreka blokira direktan (najkraći) put od gnezda do hrane [32]. Kada se ova sposobnost mrava preslika na mrežu komunikacionih pravaca na Internetu, ili mrežu puteva ili sistem telefonskih linija, tada mravi nude neka zanimljiva rešenja. Radi se o preusmeravanju saobraćaja kada ima zagušenja ili prekida na postojećim pravcima.
Opisane ideje o paralelizmu saobraćaja koje mi poznajemo i kretanju kolonija mrava dovele su do razvoja nove oblasti za koju su u paralelnoj upotrebi nazivi ‘mravlja metodologija’ (Ant Methodology) i ’algoritmi mravljih kolonija’ (Ant Colony Algorithms). Tvorcima mravlje metodologije optimizacije smatraju se belgijski naučnik Dorigo i njegov saradnik Djambardela [33, 34].

Glavna ideja novog pristupa diskretnoj optimizaciji je da se simulira ponašanje veštačkih mrava (pokretljivi agenti inspirisani ponašanjem stvarnih mrava) i generišu nova rešenja za dati problem. Veštački mravi koriste informaciju sakupljenu tokom prethodnih simulacija i usmeravaju tok daljeg traženja rešenja problema, s tim da je ta informacija raspoloživa i promenljiva u datom okruženju.
Ni jedan od mravljih algoritama ne počiva na jedinstvenoj definiciji veštačkog mrava, niti potpuno imitira ponašanje stvarnih mrava. Štaviše, pošto se radi o novoj oblasti, nisu unificirani ni terminologija, ni definicije niti matematička notacija, pre svega zbog heurističke komponente koja dopušta maštovitost naučnika u traženju najboljih simulatora prirode. U ovom poglavlju se elaboriraju neke osnovne ideje iz ove oblasti na primeru problema trgovačkog putnika (Traveling Salesman Problem - TSP) koji predstavlja paradigmu kombinatorne optimizacije.

Pronalazak minimalnog puta u grafu

Analogno prirodnim mravima, zadatak veštačkih mrava je pronaći minimalni put između čvorova grafa. Neka je $G = (N, A)$ povezani graf, gde je N skup svih čvorova, a A skup svih lukova. Broj čvorova je $N = n$. Rešenje problema predstavlja put na grafu koji povezuje izvorišni čvor f s odredišnim d , čija je daljina određena brojem lukova na putu. Slika 2 prikazuje minimalni put na grafu.



Sa svakim je lukom (i, j) grafa G povezana promenljiva τ_{ij} koja predstavlja veštački trag feromona. Tragove feromona čitaju i pišu mravi. U svakom čvoru grafa dolazi do stohastičke odluke koji je čvor sledeći. k -ti mrav lociran u čvoru i koristi trag feromona τ_{ij} za izračunavanje verovatnoće u koji čvor $j \in Ni$ treba otići. N_i je skup suseda čvora i . Verovatnoća izbora čvora j je data izrazom [33]:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}}{\sum_{j \epsilon N_i} \tau_{ij}}, j \epsilon N_i \\ 0 \qquad , j \notin N_i \end{cases}$$

Dok traži rešenje mrav ostavlja trag feromona na pripadnom luku

$$\tau_{ij}(t) \leftarrow \tau_{ij}(t-1) + \Delta\tau \ .$$

Međutim, tako definisano rešenje može dovesti do konvergencije prema suboptimalnom rešenju tako da se uvodi mehanizam isparavanja. U svakoj se iteraciji eksponencijalno smanjuje količina feromona prema izrazu [33]:

$$\tau \leftarrow (1-\rho)\tau \ , \rho\epsilon[0,1)$$

Ponašanje algoritma zavisi od broja suseda svakog čvora. Ukoliko čvorovi grafa imaju više od dva suseda, algoritam gubi na stabilnosti. Odnosno, postaje kritičan izbor parametara.
Algoritam je moguće poboljšati sagledavanjem svojstava kolonije s jedne strane i sagledavanjem svojstava pojedinog mrava.

Svojstva kolonije kao celine:

- Dobra rešenja mogu proizaći samo kao rezultat kolektivne interakcije;
- Svaki mrav koristi samo privatne informacije i lokalne informacije čvora koji posećuje;
- Mravi komuniciraju samo indirektno;
- Mravi se sami po sebi ne adaptiraju, naprotiv, menjaju način prezentacije problema i percepcije drugih mrava.

Svojstva mrava kao individue:

- Mrav traži najisplativije rešenje;
- Svaki mrav ima memoriju M_k za smeštanje informacija o putu i može da je koristi:
  a. za generisanje smislenog rešenja,
  b. za evaluaciju pronađenog rešenja,
  c. za povratak po istom putu;
- Mrav *k* može se pomaći u bilo koji čvor *j* u susedstvu N_i^k ;
- Mravu *k* može se dodeliti početno stanje i jedan ili više terminirajućih uslova;
- Mrav stvara rešenje inkrementalno i konstrukcija završava kada je zadovoljen jedan od uslova zaustavljanja;
- Izbor sledećeg čvora temelji se na verovatnoći;
- Verovatnoća pojedinačne odluke mrava temelji se na:
  a. vrednostima o strukturi čvora $A_{ij} = \lfloor a_{ij} \rfloor$ (tablica ruta) koja se sastoji od kombinacije lokalnih tragova feromona i vrednosti heurističke funkcije,
  b. privatnoj memoriji mrava,
  c. ograničenjima problema;
- Pomeranjem iz čvora *i* u čvor *j* mrav ažurira trag feromona;
- Kada je izgradio rešenje mrav se može vratiti istim putem i ažurirati trag feromona na povratku;
- Kada je izgradio rešenje i vratio se na izvorište, mrav umire i oslobađa resurse.

Problem trgovačkog putnika (TSP)

Problem trgovackog putnika (engl. Travelling Salesman Problem) predstavlja prvi problem koji je optimiziran kolonijom mrava [33]. On je definisan:
Neka je *C* skup gradova, a *L* skup veza koje povezuju elemente od *C* . Sa $J_{c_i c_j}$ označena je cena (udaljenost) između c_i i c_j , odnosno, udaljenost između gradova *i* i *j* . Problem trgovačkog putnika je optimizacioni problem pronalaska minimalnog Hamiltonovog puta na grafu *G* = (*C*, *L*). Hamiltonov put je zatvorena tura Y koja obilazi samo jednom svih N^C gradova. Njegova je dužina suma svih udaljenosti $J_{c_i c_j}$ čvorova od kojih se sastoji. Dodatno, udaljenosti ne moraju biti simetrične.

Rešavanje TSP-a pomoću ACO

Svaki veštački mrav se slučajnim izborom, smešta u neki od gradova.Tokom svake iteracije mrav bira sledeći grad koji će posetiti. Ovaj se izbor vrši pomoću sledećeg izraza.
Mrav smešten u gradu *i* odlazi u grad *j*, koji se bira iz grupe još neposećenih gradova prema verovatnoći:

$$p_k(i,j) = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot d_{ij}^{\beta}}{\sum_{g \in J_k(i)} \tau_{ig}^{\alpha} \cdot d_{ig}^{\beta}} \\ 0 \end{cases}$$

gde je:

p_k (*i j*) verovatnoća da će mrav *k* iz grada *i* otići grad *j* ,
$g \in J_k$ (*i*) skup gradova koje mrav *k* još nije obišao,
α relativna važnost traga feromona,
β relativna važnost udaljenosti između gradova.

Verovatnoća da će grad biti odabran je funkcija udaljenosti između gradova i količine feromona na putu. Pomoću parametara α i β je takođe moguće odrediti koji od ova dva faktora ima veću težinu. Jednom kada je ruta završena, odnosno kada mrav poseti svaki grad tačno jednom, vrši se isparavanje feromona i svaki mrav ostavlja feromone po celoj ruti. Količina feromona je data izrazom:

$$\tau(i,j) = p * \tau(i,j) + \sum_{k=1}^{m} \Delta\, \tau_k\,(i,j)$$

gde je :

$$\Delta\ \tau_k = \begin{cases} \frac{1}{L_k},\ (i,j) \in ruta \\ 0, (i,j) \notin ruta \end{cases}$$

$p * \tau(i,j)$: proizvod koji određuje isparavanje feromona. p se naziva konstanta isparavanja. Njena vrednost može biti između 1 i 0. Kod nižih vrednosti feromoni isparavaju brže nego kod viših vrednosti konstante isparavanja.

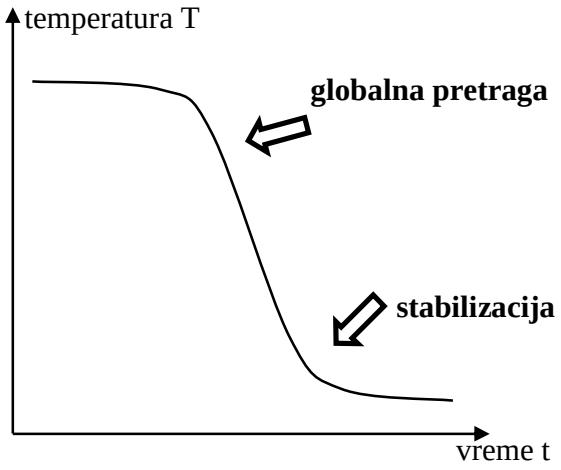
$\sum_{k=1}^{m} \Delta\ \tau_k\ (i,j)$: Količina feromona koju ostavlja mrav *k* definisana s daljinom rute koju je prešao L_k . Intuitivno, kraće će rute imati veće količine feromona[35].

5.3. Algoritam simuliranog kaljenja

Algoritam simuliranog kaljenja (**Simulated Annealing, SA**) jedan je od stohastičkih optimizacionih algoritama, a razlikuje se od algoritama lokalnog pretraživanja po tome što omogućuje beg iz lokalnog optimuma. Algoritam je nastao po analogiji sa metalurškim kaljenjem, čiji je cilj oplemenjivanje metala tako da oni postanu čvršći. Ako se želi postići čvrstoća metala, potrebno je kristalnu rešetku metala pomeriti tako da ima minimalnu potencijalnu energiju. U procesu metalurškog kaljenja metal se prvo zagreva do visoke temperature, a potom se, nakon kraćeg zadržavanja na toj temperaturi, polagano hladi do sobne temperature. Prebrzo hlađenje bi moglo uzrokovati pucanje metala. Posledica toga je da atomi metala nakon procesa kaljenja formiraju pravilnu kristalnu rešetku. Time je postignut i energetski minimum kristalne rešetke.

U skladu s navedenim, SA će sadržati parametar temperature, a funkciju kojoj želimo odrediti globalni optimum možemo posmatrati kao energiju rešetke, ukoliko određujemo minimum, odnosno negativnu energiju rešetke, ukoliko određujemo maksimum. Algoritam počinje izborom nekog početnog rešenja, a početna temperatura ima relativno veliku vrednost. Postojeće rešenje se zamenjuje sa boljim, ali se može zameniti i sa lošijim uz određenu verovatnoću prihvatanja [36]. Ta verovatnoća se određuje izborom slučajnog broja a iz intervala [0,1] i uslova da je $a < e^{\frac{E(staro)-E(novo)}{T}}$, gdje je *E(x)* funkcija za koju se traži globalni minimum, a *T* temperatura. Ukoliko je izraz istinit novo rešenje se prihvata. Ovo nam omogućava beg iz lokalnog minimuma. Isto tako, verovatnoća da će biti odabrano lošije rešenje je veća kada je veća temperatura. To znači da je u početku prostor pretrage rešenja dosta veliki i da se smanjuje padom temperature, a pri kraju procesa je usko lokalizovan. Ponašanje funkcije je u velikom određeno

početnom vrednosti temperature i njenom brzinom padanja. Najčešće temperatura opada eksponencijalno s parametrom *a* iz intervala 0<a<1 (a je blizak jedinici). Za premali *a* hlađenje je prebrzo (pucanje metala) i veća je verovatnoća upada u lokalni optimum. Za preveliki *a* temperatura presporo pada, i zbog velike verovatnoće da se u početku zamenjuju bolja rešenja s lošijim, algoritam ima svojstvo da nasumično pretražuje veliki prostor rešenja. To dodatno usporava rad, pa se nastoji da se *a* odabere da bude dovoljno veliko, ali da temperatura pri kraju procesa bude niska. Time se rešenje pred kraj procesa stabilizuje u lokalnom području. Zbog velike osetljivosti na koeficijent *a*, posebnu pažnju treba obratiti prilikom njegovog izbora [36, 37].



Slika Globalna pretraga i stabilizacija kod SA-a

6.Ostali heuristički algoritmi

U prethodnim poglavljima je data podela heurističkih algoritama i opisana nekolicina najpoznatijih metaheurističkih algoritama. U ovom poglavlju se izlaže kratki opis ostalih heurističkih algoritama.

Pretraživanje najboljim prvim (**Best-first search**) je heuristički algoritam koji optimizuje *pretraživanje po širini* (*Breadth-first search*). Funkcijom heuristike se procenjuju čvorovi u grafu i bira se najbolje procenjeni čvor za dalje istraživanje. U tu grupu algoritama spada i A* pretraživanje. **A* pretraživanje** i njegova posebna varijanta **Dijkstrin algoritam** su heuristički algoritmi koji služe za pronalaženje najkraćeg puta između dva vrha u grafu. A* pretraživanje i Dijkstrin algoritam nas uvek dovode do optimalnog rešenja ako ono postoji, tj. oni su završni algoritmi.

Algoritam penjanja uzbrdo (**Hill Climbing**, **HC**) po svom načinu optimizacije pripada familiji algoritama lokalnog pretraživanja. Temeljen je na logici uspona: algoritam počinje od nekog rešenja i ukoliko postoji *sused* koji bolje optimizira funkciju, novo rešenje postaje taj *sused*. Veliki problem HC-a je mogućnost zapinjanja u lokalnom optimumu. Ovakva skraćenica algoritma je poznata pod nazivom **jednostavni HC** (**Simple HC**). Bolji od predhodnog HC-a je **stohastičko-restartovan HC** (**Random-Restart HC, RRHC**). RRHC u svakoj iteraciji slučajno bira početno rešenje i nad njime provodi SHC. Kao svoje rešenje bira najbolje rešenje iz svih iteracija. Ovakvo rešenje je slično *višestartnom lokalnom pretraživanju*.

Ponavljajuća pohlepa (**Iterated Greedy, IG**) je algoritam koji uzastopnim iteracijama sprovodi pohlepnu destrukciju i konstrukciju postojećeg rešenja. U svakoj iteraciji se novostvoreno rešenje prihvata u zavisnosti od kriterijuma prihvatanja.

GRASP (**The Greedy Randomized Adaptive Search Procedure**, **GRASP**) je hibridna metaheuristička metoda. GRASP je kombinacija konstruktivne heuristike i lokalnog pretraživanja. Rešenje se sastavlja korak po korak uzimajući u svakoj iteraciji, slučajnim izborom, komponentu rešenja iz ograničene liste kandidata. Ograničena lista kandidata se sastoji od α najboljih elemenata rangiranih prema vrednosti pohlepne funkcije, tj. prema kvalitetu rešenja koje bi mogli postići. Ukoliko je α jednak jedinici dobijamo osnovni pohlepni algoritam, a ukoliko je α jednak broju komponenata dobijamo slučajno pretraživanje. Algoritam je osetljiv na parametar α i najčešće ga algoritam menja iz iteracije u iteraciju (adaptacija algoritma).

Squeeky Wheel (**SW**) je metaheuristčki algoritam koji se može smatrati posebnim slučajem IG-a ili GRASP-a. SW gradi rešenje pomoću pohlepnog algoritma, a elementima rešenja koja uzrokuju grešku dodeljuje kaznene bodove. Kazneni bodovi služe da bi se popravio prethodno odabran poredak elemenata rešenja, pri čemu se elementi sa više kaznenih bodova guraju napred.

Usmereno lokalno pretraživanje (**Guided Local Search, GLS**), menja funkciju cilja da bi izbegao upadanje u lokalni optimum. Prilikom optimizacije funkcije se koristi lokalno pretraživanje koje često može dati suboptimalno rešenje. Neko suboptimalno rešenje se može često pojavljivati i treba ga učiniti nepoželjnim. GLS se koristi svojstvima rešenja, na primer ivice grafa u TSP-u, da bi diskriminirao rešenja koja vode ka lokalnom optimumu. U svakoj iteraciji se za svako svojstvo koje se pojavljuje u rešenju povećava kazna, a funkcija cilja se preoblikuje tako da se poveća za izraz koji zavisi o kaznama svojstava koja su uključena u rešenje. Dakle, GLS ne menja okolinu poput VNS-a da bi što bolje pretražilo prostor rešenja, već to čini menjanjem funkcije koju optimizuje (funkcija cilja).

Kvantno kaljenje (**Quantum Annealing, QA**) je heuristički algoritam koji nadograđuje simulirano kaljenje. Velike prednosti ima nad funkcijama koje imaju velike, ali uske barijere. Simulirano kaljenje tada ima veću mogućnost zapinjanja u lokalnom optimumu,tj. teži beg iz lokalnog optimuma zbog visine barijere. QA uvodi pojam neodređenosti po uzoru na talasno svojstvo kvantnog sveta i ima mogućnost tuneliranja kroz takve barijere. Smatra se da bi se QA trebalo puno bolje izvoditi na kvantnim računarima.

Pretraživanje promenjivom širinom (**Variable Depth Search**, **VDS**) je generalizacija lokalnog pretraživanja. Osnovna ideja VDS-a je promeniti veličinu okoline po kojoj se pretražuje tako da algoritam može pretraživati složene probleme u razumnom vremenu. U grupu VDS-a spada i **Lin-Kernighanov algoritam**.

Ekstremna optimizacija (**Extremal optimization, EO**) vuče inspiraciju iz Bak-Sneppenovog modela samo-organizovane kritičnosti iz područja statističke fizike [37]. EO optimizuje probleme koji se mogu rastaviti na komponente i nezavisno procenjivati. Vrši se evolucija pojedinih komponenti rešenja: najgore komponente se menjaju. Početno rešenje se može odabrati nasumično.

Harmonijsko pretraživanje (**Harmony search**, **HS**) je metaheuristički algoritam koji oponaša grupu muzičara u potrazi za savršenom harmonijom. Analogija je jednostavna: kao što muzičari traže savršenu harmoniju i slušaoci daju estetsku procenu, tako u HS-u pokušavamo rešiti optimizacioni problem procenom pojedinih rešenja. Četiri su osnovna koraka u HS-a: inicijalizacija parametara i harmonijske memorije, improvizovanje nove memorije, obnavljanje harmonijske memorije, provera kriterijuma zaustavljanja. Zadnja tri koraka pripadaju iterativnom postupku. HS se dobro pokazao u rešavanju NP-teških problema i dizajniranju cevovodne mreže [36].

Bakteriološki algoritmi (**Bacteriologic Algorithms, BA**) su algoritmi inspirisani evolucijskom ekologijom, tj. adaptacijom bakterija na okolinu u kojoj obituju [3]. Algoritam simulira adaptaciju bakterija na okolinu, a bitno svojstvo koje uzima u obzir je nemogućnost prilagođavanja populacije bakterija na sve uslove okoline. BA su se pokazali uspešnije od EA-a u područjima dubinske analize podataka i u problemima kompleksnog pozicioniranja [37].

Memetski Algoritam (**Memetic Algorithm**, **MA**) je kombinacija EA sa lokalnim pretraživanjem. Naziv je dobio po kulturološkom genu *mem* koji prenosi informacije iz jednog uma u drugi. Za razliku od gena, *mem* se ne prenosi razmnožavanjem.

MA imaju svojstvo da dobro pretražuju i iskorišćavaju prostor rešenja, ali su naročito osetljivi na odnos poremećaja informacije i njenog očuvanja prilikom mutacije: da bi se postigao beg iz lokalnog optimuma i zadovoljio uslov stabilnosti algoritma, potrebno je taj odnos uravnotežiti.

Razbacano pretraživanje (**Scatter Search, SS**) je metaheuristička metoda koja se od EA-a razlikuje po tome što omogućuje opšte principe za rekombinaciju rešenja zasnovanih na konstrukciji puta u Euklidovom prostoru [36]. Koristi male populacije čijom kombinacijom nastaju nova rešenja. Jedinke se prikazuju kao tačke u Euklidovom prostoru i njihovom linearnom kombinacijom nastaju rešenja sledeće generacije. Opštija verzija SS je **ponovno vezivanje puta** (**Path relinking** , **PR**).

Procena distribucijskih algoritama (**Estimation of Distribution Algorithms, EDA**) su grupa algoritama koji imaju podlogu u teoriji verovatnoće, ali koriste i populaciju koja se razvija kroz proces pretraživanja [35]. Nedostatak EA je u činjenici da se rekombinacijom mogu razbiti blokovi dobrih rešenja. EDA se koristi verovatnosnim modelom kojim se procenjuje distribucija dobrih rešenja preko prostora pretraživanja. Uzorkom po toj distribuciji dobijamo rešenja sledeće generacije [36]. Procena distribucije se vrši u svakoj iteraciji na temelju postojećih rešenja.

Metoda ukrštene entropije (**Cross-Entropy Method, CEM**) je metaheuristika koja potiče iz područja simulacije retkih događaja gde je potrebno što tačnije odrediti verovatnoću pojave događaja male verovatnoće. Simulacija retkih događaja i uzorkovanje po važnosti osnove su ove metode. CEM ima dve faze:

- generisanje slučajnih uzoraka rešenja prema određenom mehanizmu
- obnavljanje parametara mehanizma u svrhu dobijanja boljih uzoraka

Diferencijalna evolucija (**Differential evolution, DE**) je metoda koja spada u grupu ES-a. DE se koristi za optimizaciju funkcija više promenljivih i ima veliku verovatnoću pronalaska globalnog optimuma. Najvažniji deo DE-a je shema za generisanje probnih parametarskih vektora [37]. Shema je samoorganizujuća pa nisu potrebne verovatnosne distribucije koje koriste ES. Diferencijalna evolucija se od drugih evolucionih strategija najviše razlikuje u fazi mutacije.

Grupišući genetski algoritam (**Gruping Genetic Algorithm, GGA**) je unapređenje GA. Kod rešavanja nekih problema je bolje operisati nad grupom jedinki nego nad pojedinom jedinkom. GGA optimizuje uz pomoć grupe jedinki. Skup jedinki se razdvaja u disjunktne grupe i svakoj se grupi dodeljuju svojstva ekvivalentna genima u običnom

GA. Ovakav način optimizacije zahteva promenjive veličine hromozoma pojedinih grupa jedinki i posebne operatore manipulacije nad grupama jedinki.

Interaktivni genetski algoritmi (**Interactve genetic agorithms, IGA**) su grupa algoritama koji uključuju ljudsku procenu, a pronalazimo ih u područjima gde funkciju prilagđenosti nije lako pronaći (umetnost) [37].

7. Zaključak

Metaheuristički algoritmi su se pokazali uspešni u rešavanju mnogih optimizacionih problema. Posebno zanimljiva grupa problema koje ovi algoritmi optimiziraju su NP-teški problemi. Razlika između iscrpnog pretraživanja i metaheurističkih algoritama u rešavanju NP-teških problema može se dočarati na sledeći način. Neka NP-teški problemi teže nekoliko tona, a P problemi nekoliko grama. Iscrpno pretraživanje bi predstavljala vaga koja je skalirana na grame i može izvagati svega nekoliko kilograma. Metaehurističke algoritme bi predstavljala vaga skalirana na tone. Stoga, ukoliko želimo izvagati NP-teške probleme moramo koristiti tonsku vagu, ali preciznost nam u tom slučaju, zbog skale koju koristimo, nije zagarantovana. P probleme je bolje vagati s vagom skaliranom na gramima jer se brže (zbog problema iščitavanja težine na tonskoj vagi) i preciznije dolazi do težine.

Drugim rečima, prednost metaheurističkih algoritama je što ne vršimo iscrpno pretraživanje, a to znači da nam optimizacija može biti vrlo brza. Ipak, rešenja koja dobijemo ne moraju biti kvalitetna, a treba istaknuti da metaheuristički algoritmi nemaju uvek prednost brzine.

Metaheuristički algoritmi se primenjuju u različitim područjima računarske nauke: od veštačke inteligencije do telekomunikacija.

I na kraju o pokušajima ovog rada. Cilj rada je uvesti čitaoce u svet heurističkih algoritama, dati im drugi pogled na rešavanje problema i zainteresovati ih za dalje istraživanje na ovom području. Kroz zanimljive i ilustrativne primere pokušalo se olakšati usvajanje vrlo zanimljive i intrigantne teme.

LITERATURA

[1] Golub M., *Genetski algoritam*, drugi dio, Zagreb, 2004. Dostupno na Internet adresi: http://www.zemris.fer.hr/~golub

[2] Michalewicz, Z. and C. Z. Janikow (1991). *Handling constraints in genetic algorithms*. Genetic Algorithms: Proceedings of the Fourth International Conference,San Mateo, pages: 151-157.

[3] Michalewicz Z., Fogel D.B. *How to solve it.. modern heuristics* (Springer, 1996) (chapters 1 to 112.)

[4] Michalewicz, Z., "*Genetic Algorithms + Data Structures = Evolution Programs*", Springer-Verlag, Berlin, 1994.

[5] Liu, Yang, Whidborne. *Multiobjective Optimisation and Control*, 2004.

[6] Golub, M., *Vrednovanje uporabe genetskih algoritama za aproksimaciju vremenskih nizova*, magistarski rad, Zagreb, 1996. Dostupno na Internet adresi: http://www.zemris.fer.hr/~golub/magrad.html

[7] Filipović V., *Operatori selekcije i migracije i web servisi kod paralelnih evolutivnih algoritama*, doktorska disertacija, Matematički fakultet, Beograd, 2006.

[8] Davidović T., *Raspoređivanje zadataka na višeprocesorske sisteme primenom metaheuristika*, doktorska disertacija, Matematički fakultet, Beograd, 2006.

[9] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenovic. *Parallel variable neighborhood search for the p-median*. Technical report,GERAD report G-2003-04, 2003.

[10] F. Glover. *Future paths for integer programming and links to artificialintelligence*. Computers Ops. Res., 1986.,pages:533-549.

[11] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers,1997.

[12] P. Hansen. *The steepest ascent mildest descent heuristics for combinatorial programming*. In Congress on Numerical Methods ni Combinatorial Optimization, Capri, Italy, 1986.

[13] P. Hansen and N. Mladenovic. *An introduction to variable neighborhood search.* In Voss, S. *et al.*, editor, *Metaheuristics, Advances andTrends in Local Search Paradigms for Optimization*, pages 433-458. Kluwer, Dordrecht, 1999.
[14] P. Hansen and N. Mladenovic. *Fundamentals of variable neighborhoodsearch*. In Proc. 10. Congr. Yugoslav Mathematicians, pages 57-72, Beograd, 2001.

[15] P. Hansen and N. Mladenovic. *Variable neighborhood search: Principles and applications.* Europ. J. Oper. Res.,2001., pages:449-467.

[16] P. Hansen and N. Mladenovic. *Developments of the variable neighbor-hood search*. In C. Ribeiro and P. Hansen, editors, *Essays and Surveysin Metaheuristics*, pages 415-439. Kluwer Academic Publishers, 2002.

[17] P. Hansen and N. Mladenovic. *Variable neighbourhood search*. InF. Glover and G. Kochenagen, editors, *Handbook of Metaheuristics*, pages 145-184. Kluwer Academic Publishers, Dordrecht, 2003.

[18] N. Mladenovic. *A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization applications*. In Optimization Days, page 112, Montreal, 1995.

[19] N. Mladenovic and P. Hansen. *Variable neighborhood search*. Comput.& OR, 1997.

[20] Golderg, D. E. (1989). *Genetic Algorithm in Search, Optimization, and Machine Learning*. Addison Wesley Publishing Company.

[21] Bäck, T., Schwefel, H.-P., *An Overview of Evolutionary Algorithms for Parameter Optimization*, Evolutionary Computation 1(1), pages: 1-23, 1993.

[22] Bäck, T., *Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms*, Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway NJ, pages: 57-62, 1994.

[23] Bäck, T., *Generalized Convergence Models for Tournament- and (*m,l*)-Selection*, Proceedings of the SixthInternational Conference on Genetic Algorithms, San Francisco, CA, pages: 2-8, 1995.

[24] Bäck, T., *Order Statistics for Convergence Velocity Analysis in Simplified Evolutionary Algorithms*, Foundations of Genetic Algorithms, Vol. 3, Morgan Kaufmann, pages: 91-102, San Mateo CA, 1995.

[25] Blickle, T., Thiele, L., *A Mathematical Analisys of Tournament Selection*, Proceedings of the Sixth International Conference on Genetic Algorithms, San Francisco, CA, pages: 2-8, 1995.

[26] Cantú-Paz, E., *Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms*, 1999. Dostupno na Internet adresi:
ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs
[27] Davis, L., "*Handbook of Genetic Algorithms*", Van Nostrand Reinhold, New York, 1991.

[28] Miller, B.L., Goldberg, D.E., *GAs Tournament Selection and the Effects of Noise*, 1995. Dostupno na adresi: http://www.dai.ed.ac.uk/groups

[29] Mühlenbein, H., *The optimal population size for uniform crossover and truncation selection*, 1994.

[30] Tomassini, M., *Parallel and Distributed Evolutionary Algorithms: A Review*, clanak u knjizi "Evolutionary Algorithms in Engineering and Computer Science", Miettinen, K., Neittaanmaki, P., Makela, M.M., Periaux, J., John Wiley & sons, LTD, 1999.

[31] Whitley, D., *A Genetic Algorithm Tutorial*, 1996. Dostupno na Internet adresi: http://www.dai.ed.ac.uk/groups

[32] Bonabeau E., Dorigo M., and Theraulaz G. (1999), *Swarm Intelligence: From Natural to Artificial Systems*, Oxford Univ. Press, New York.

[33] Dorigo M., Maniezzo V., and Colorni A. (1996), *The Ant System: Optimization by a Colony of Cooperating Agents*, IEEE Trans. Syst. Man Cybern. B 6, pages: 29-41.

[34] Gambardella L. M., Taillard E. D., and Dorigo M. (1999), *Ant Colonies for the Quadratic Assignment Problem*, J. Op. Res. Soc. 50, pages:167-176.

[35] Blum C., Roli A., *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*, ACM Computing Surveys (CSUR), 3. edition , 2003., pages:268-308.

[36] Chiarandini M., *Heuristics for Combinatorial Optimization Problems*,2006., http://www.imada.sdu.dk/Courses

[37] Heuristic algorithms, *Heuristic (computer science)*, http://en.wikipedia.org/wiki/Heuristic_algorithm, 2007.

[38] Soić N., *Oblikovanje algoritama*, http://lnrpc2.irb.hr/soya/nastava/predavanje-6.ppt, .

[39] Karr,C.L. (1991). Design *of an adaptive fuzzy logic controller using a genetic algorithm*. Proc. ICGA, vol. 4, pages: 450-457.