SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2740

RJEŠAVANJE PROBLEMA USMJERAVANJA VOZILA OGRANIČENOG KAPACITETA UZ POMOĆ METAHEURISTIKA

Ivan Rissi

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2740

RJEŠAVANJE PROBLEMA USMJERAVANJA VOZILA OGRANIČENOG KAPACITETA UZ POMOĆ METAHEURISTIKA

Ivan Rissi

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 2740

Pristupnik:	**Ivan Rissi (0036506108)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Marin Golub

Zadatak: **Rješavanje problema usmjeravanja vozila ograničenog kapaciteta uz pomoć metaheuristika**

Opis zadatka:

Definirati i opisati problem usmjeravanja vozila uzimajući u obzir kapacitet vozila (engl. Capacitated Vehicle Routing Problem, CVRP). Navesti taksonomiju heurističkih optimizacijskih algoritama. Odabrati nekoliko metaheuristika prikladnih za rješavanje problema CVRP i opisati ih. Ostvariti i opisati programski sustav za zadavanje i rješavanje problema CVRP koji omogućuje podešavanje parametara i mogućnosti odabranih metaheurističkih optimizacijskih algoritama. Ispitati utjecaj početne populacije, parametara algoritma i utjecaj pretrage susjedstva na kvalitetu konačnog rješenja.

Rok za predaju rada: 27. lipnja 2022.

SADRŽAJ

1. Uvod

Problem usmjeravanja vozila danas je jedan od najpopularnijih i najzastupljenijih optimizacijskih problema. Svoju popularnost stječe zbog svakodnevne primjene u logistici — velike tvrtke neprestano šalju i dostavljaju proizvode brojnim kupcima. Međutim, organizacija vozila na način da se svim kupcima dostave svi potrebni resursi, a da se pritom minimizira potrošnja nije jednostavan zadatak. Ručno raspoređivanje zbog kompleksnosti velikog broja takvih problema ne dolazi u obzir.

Vozila je moguće optimalno organizirati računalno, međutim, izvršavanje determinističkih algoritama koji mogu pronaći optimalno rješenje u konačnom vremenu traje predugo — ovisno o njihovoj veličini, rješavanje nekih problema moglo bi potrajati godinama, tisućama godina, a kod nekih problema čak i mnogo duže.

S druge strane, nedeterministički algoritmi ne pronalaze nužno optimalno rješenje, štoviše, vrlo rijetko se događa da nedeterministički algoritam pronađe optimalno rješenje, posebice na ogromnim problemima. Unatoč tome, rješenja koja nedeterministički algoritmi pronalaze najčešće su „dovoljno dobra“, a njihovo vrijeme izvođenja je više nego prihvatljivo u usporedbi s vremenom izvođenja determinističkih algoritama.

U ovom se radu promatra jedna varijanta problema usmjeravanja vozila — problem usmjeravanja vozila ograničenog kapaciteta. Uz navedeni problem, promatra se i skup od osam nedeterminističkih algoritama čijim se programskim ostvarenjem nastoje naći što bolja rješenja konkretnih problema.

2. Problem usmjeravanja vozila

2.1. Opis problema usmjeravanja vozila

Problem usmjeravanja vozila ograničenog kapaciteta (engl. *Capacitated vehicle routing problem*, u daljnjem tekstu „*CVRP*“) spada u klasu problema usmjeravanja vozila (engl. *Vehicle routing problem*, u daljnjem tekstu „*VRP*“) — generalizacije problema trgovačkog putnika (engl. *Travelling salesman problem*, u daljnjem tekstu „*TSP*“).

TSP postavlja pitanje [9]: „Uz danu listu gradova te poznatu udaljenost između svakog para, koja je najkraća moguća ruta koja posjećuje svaki grad točno jednom te se vraća u početni grad?“. Problem je NP-težak, a zbog svoje jednostavne formulacije je vrlo pogodan kao optimizacijski problem.

Najjednostavniji oblik VRP-a generalizira TSP na način da uvodi više trgovačkih putnika, koji se u kontekstu VRP-a nazivaju vozilima. Svako od ovih vozila posjećuje disjunktni skup gradova te se naposljetku vraća u početnu točku (skladište). Drugim riječima, VRP postavlja pitanje [10]: „Koji je optimalan skup ruta koji bi flota vozila morala proći kako bi dostavila resurse danom skupu gradova (kupaca)?“.

VRP može definirati dodatna ograničenja kako bi se oponašale situacije iz pravog života, primjerice:

- kapacitetno ograničenje — ukupna potražnja kupaca na jednoj ruti ne smije premašiti kapacitet vozila;
- vremenski prozori — vozilo kupcu mora dostaviti dobra unutar zadanog vremenskog intervala;
- višeskladišno ograničenje — postoji više skladišta iz kojih vozila mogu krenuti te se na kraju u ista moraju i vratiti.

CVRP koristi kapacitetno ograničenje — nastoji minimizirati ukupnu udaljenost koju sva vozila prelaze na svojim rutama, pri čemu su svi kupci posjećeni točno jednom te kapacitet nijednog vozila nije premašen.

2.2. Definicija problema usmjeravanja vozila

CVRP se može definirati kao neusmjereni simetrični težinski graf $G = (V, E)$, pri čemu V označava skup svih kupaca zajedno sa skladištem, a E označava skup cesta između gradova. Graf je dvostruko otežan:

- svakom vrhu (kupcu) pridružen je pozitivni realni broj koji predstavlja kupčevu potražnju;
- svakom bridu (cesti) pridružen je pozitivni realni broj koji predstavlja duljinu te ceste.

CVRP je zadan na sljedeći način:

- N — broj kupaca (kupci su predstavljeni indeksima $\{1, \ldots, n\}$, dok indeks 0 predstavlja skladište);
- K — broj vozila na raspolaganju;
- d_i — potražnja i-tog kupca;
- $c_{i,j}$ — duljina puta između i-tog i j-tog kupca;
- q — kapacitet vozila;
- $x_{i,j}^k$ — binarna varijabla odluke, označava je li k-to vozilo putovalo bridom $e_{i,j}$.

Funkcija dobrote koju je potrebno minimizirati glasi:

$$f = \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=1}^{K} c_{i,j} x_{i,j}^k \tag{2.1}$$

te podliježe sljedećim ograničenjima [3, 12]:

- ograničenje (2.2) osigurava da je varijabla $x_{i,j}^k$ binarna — jedine opcije su da vozilo k *jest* putovalo bridom $e_{i,j}$ ($x_{i,j}^k = 1$) te da vozilo k *nije* putovalo bridom $e_{i,j}$ ($x_{i,j}^k = 0$):

$$x_{i,j}^k \in \{0, 1\}, \quad \forall i, j \in \{0, \ldots N\}, k \in \{1, \ldots K\}; \tag{2.2}$$

- ograničenje (2.3) osigurava da vozilo s neke lokacije uvijek mora otputovati na drugu lokaciju:

$$x_{i,i}^k = 0, \quad \forall i \in \{0, \ldots, N\}, k \in \{1, \ldots, K\}; \tag{2.3}$$

- ograničenje (2.4) osigurava da svako vozilo napušta skladište točno jednom te odlazi k određenom kupcu:

$$\sum_{j=1}^{N} x_{0,j}^k = 1, \quad \forall k \in \{1, \ldots, K\}; \tag{2.4}$$

– ograničenje (2.5) osigurava da svaki kupac koji je posjećen mora biti i napušten:

$$\sum_{i=0}^{N} x_{i,p}^{k} - \sum_{j=0}^{N} x_{p,j}^{k} = 0, \quad \forall p \in \{1, \ldots, N\}, k \in \{1, \ldots, K\}; \tag{2.5}$$

– ograničenje (2.6) osigurava da se svako vozilo koje napusti skladište u isto mora i vratiti:

$$\sum_{j=1}^{N} x_{0,j}^{k} - \sum_{i=1}^{N} x_{i,0}^{k} = 0, \quad \forall k \in \{1, \ldots, K\}; \tag{2.6}$$

– ograničenje (2.7) osigurava da je svaki kupac posjećen od strane točno jednog vozila:

$$\sum_{k=1}^{K} \sum_{i=0, i \neq j}^{N} x_{i,j}^{k} = 1, \quad \forall j \in \{1, \ldots, N\}; \tag{2.7}$$

– ograničenje (2.8) osigurava nemogućnost prekoračenja kapaciteta vozila:

$$\sum_{j=0}^{N} \sum_{i=0, i \neq j}^{N} d_j x_{i,j}^{k} \leq q, \quad \forall k \in \{1, \ldots, K\}. \tag{2.8}$$

2.3. Prikaz rješenja

Svako rješenje problema CVRP je, bez obzira na valjanost, skup od K ruta koje počinju i završavaju u skladištu. Kako bi prikaz rješenja kao skup od K vektora bio nezgrapan i nepogodan za korištenje u metaheurističkim algoritmima, rješenje je potrebno kodirati na način koji odgovara konkretnom algoritmu koji se koristi.

Uvode se sljedeća dva prikaza rješenja:

– permutacijski prikaz rješenja — permutacijski vektor nenegativnih cijelih brojeva koji je pogodan za većinu algoritama;

– prikaz rješenja realnim brojevima — vektor realnih brojeva koji koristi kodiranje nasumičnim ključem; koristi se u slučaju kada permutacijski prikaz rješenja nije prikladan.

2.3.1. Permutacijski prikaz rješenja

U slučaju N gradova (ne uključujući početni grad — skladište) i K vozila, rješenje je kod permutacijskog prikaza prikazano kao permutacijski vektor veličine $N + K - 1$. Prvih N nenegativnih cijelih brojeva u zapisu predstavlja gradove redom (0 predstavlja prvi, 1 predstavlja drugi, itd.), dok sljedećih K cijelih brojeva ($N, N + 1, \ldots, N +$

$K - 1$) predstavlja *graničnike* — povratak vozila u skladište. Valja primijetiti da se skladište kao početak i kraj rute svakog vozila eksplicitno ne navodi jer je implicitno definirano graničnicima. Graničnici će u nastavku teksta uvijek biti podcrtani.

Primjerice, u slučaju osam gradova (ne uključujući skladište) i dva vozila, niz:

$$\begin{bmatrix} 4 & 6 & \underline{9} & 1 & 3 & 7 & 5 & 2 & \underline{8} & 0 \end{bmatrix}$$

predstavlja rješenje u kojem jedno od vozila posjećuje redom gradove: $1, 3, 7, 5, 2$ pa se vraća u skladište, dok drugo vozilo posjećuje redom gradove: $0, 4, 6$ pa se vraća u skladište. Valja primijetiti da permutacija ignorira početak i kraj niza te da su jedini graničnici između ruta upravo dva podcrtana broja te da bi, primjerice, niz:

$$\begin{bmatrix} 6 & \underline{9} & 1 & 3 & 7 & 5 & 2 & \underline{8} & 0 & 4 \end{bmatrix}$$

predstavljao identične rute vozila kao i u prethodnom slučaju.

2.3.2. Prikaz rješenja realnim brojevima

Kod nekih algoritama, permutacijski prikaz rješenja nije izravno prikladan. Iz tog razloga se uvodi prikaz rješenja vektorom realnih brojeva koji se pomoću kodiranja nasumičnim ključem (engl. *random key encoding*, u daljnjem tekstu „RK kodiranje“) svodi na permutacijski prikaz. RK kodiranje kodira rješenje pomoću vektora nasumičnih realnih brojeva koji se koriste kao sortirni ključevi kod dekodiranja rješenja [2] — brojevi u nizu se sortiraju te se njihov originalni poredak zamjenjuje odgovarajućim indeksima iz sortiranog niza.

Primjerice, niz:

$$\begin{bmatrix} 0.78 & 0.39 & 0.58 & 0.08 & 0.81 & 0.66 & 0.18 & 0.92 & 0.12 & 0.42 \end{bmatrix}$$

bi sortiranjem postao:

$$\begin{bmatrix} 0.08 & 0.12 & 0.18 & 0.39 & 0.42 & 0.58 & 0.66 & 0.78 & 0.81 & 0.92 \end{bmatrix}.$$

Ako su indeksi brojeva u sortiranom nizu:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

te se originalni niz zamijeni indeksima brojeva na odgovarajućim pozicijama, početni niz:

$$\begin{bmatrix} 0.78 & 0.39 & 0.58 & 0.08 & 0.81 & 0.66 & 0.18 & 0.92 & 0.12 & 0.42 \end{bmatrix}$$

postao bi permutacijskim vektorom:

$$\left[7 \quad 3 \quad 5 \quad 0 \quad \underline{8} \quad 6 \quad 2 \quad \underline{9} \quad 1 \quad 4\right]$$

koji predstavlja rješenje u kojem jedno vozilo posjećuje redom gradove $6, 2$ pa se vraća u skladište, dok drugo vozilo posjećuje redom gradove $1, 4, 7, 3, 5, 0$ pa se vraća u skladište.

2.3.3. Dekodiranje rješenja

Vrijednost evaluacijske funkcije (kazne) računa se kao suma ukupnih udaljenosti koje sva vozila prevale na svojim rutama. Ako rješenje nije valjano (prekoračen je kapacitet nekog vozila), ono se kažnjava proporcionalno prekoračenju, pri čemu se koeficijent kažnjavanja može regulirati.

2.4. Definicija susjedstava

Neki algoritmi u svom osnovnom obliku (dok preostali algoritmi kao opcionalno proširenje) prilikom rješavanja problema koriste lokalnu pretragu. U tu svrhu definirana su tri različita para susjedstava:

- susjedstvo intra-*swap* i susjedstvo inter-*swap*;
- susjedstvo intra-*shift* i susjedstvo inter-*shift*;
- susjedstvo intra-*2-opt* i susjedstvo inter-*2-opt*.

2.4.1. Susjedstva intra-*swap* i inter-*swap*

Susjedstvo intra-*swap* definirano je kao skup svih rješenja S_{intra_swap} koja iz originalnog rješenja S nastaju zamjenom mjesta dvaju njegovih elemenata (gradova) *koji se nalaze na istoj ruti*. Za originalno rješenje:

$$S = \left[4 \quad 6 \quad \underline{9} \quad \mathbf{1} \quad 3 \quad 7 \quad \mathbf{5} \quad 2 \quad \underline{8} \quad 0\right],$$

primjer intra-*swap* susjeda bio bi:

$$S^{1,1,4}_{intra\_swap} = \left[4 \quad 6 \quad \underline{9} \quad \mathbf{5} \quad 3 \quad 7 \quad \mathbf{1} \quad 2 \quad \underline{8} \quad 0\right].$$

Slično tome, susjedstvo inter-*swap* definirano je kao skup svih rješenja S_{inter_swap} koja iz originalnog rješenja S nastaju zamjenom *bilo kojih dvaju elemenata (gradova*

ili skladišta). Za originalno rješenje:

$$S = \left[4 \quad 6 \quad \underline{9} \quad 1 \quad \mathbf{3} \quad 7 \quad 5 \quad 2 \quad \underline{\mathbf{8}} \quad 0\right],$$

primjer inter-*swap* susjeda bio bi:

$$S^{5,9}_{inter\_swap} = \left[4 \quad 6 \quad \underline{9} \quad 1 \quad \underline{\mathbf{8}} \quad 7 \quad 5 \quad 2 \quad \mathbf{3} \quad 0\right].$$

Može se primijetiti da se prilikom pretraživanja susjedstva inter-*swap* graničnici (skladišta) mogu premještati te na taj način rezultirati rutama različitih duljina u odnosu na originalno rješenje S.

2.4.2. Susjedstva intra-*shift* i inter-*shift*

Susjedstvo inter-*shift* definirano je kao skup svih rješenja S_{intra_shift} koja iz originalnog rješenja S nastaju pomicanjem jednog njegovog elementa (grada) za m mjesta *unutar iste rute*. Za originalno rješenje:

$$S = \left[4 \quad 6 \quad \underline{9} \quad 1 \quad \mathbf{3} \quad 7 \quad 5 \quad 2 \quad \underline{8} \quad 0\right],$$

primjer intra-*shift* susjeda bio bi:

$$S^{1,2,3}_{intra\_shift} = \left[4 \quad 6 \quad \underline{9} \quad 1 \quad 7 \quad 5 \quad 2 \quad \mathbf{3} \quad \underline{8} \quad 0\right].$$

Slično tome, susjedstvo inter-*shift* definirano je kao skup svih rješenja S_{inter_shift} koja iz originalnog rješenja S nastaju pomicanjem jednog njegovog elementa (grada ili skladišta) za m mjesta *preko cijelog niza*. Za originalno rješenje:

$$S = \left[4 \quad 6 \quad \underline{9} \quad 1 \quad 3 \quad 7 \quad 5 \quad \mathbf{2} \quad \underline{8} \quad 0\right],$$

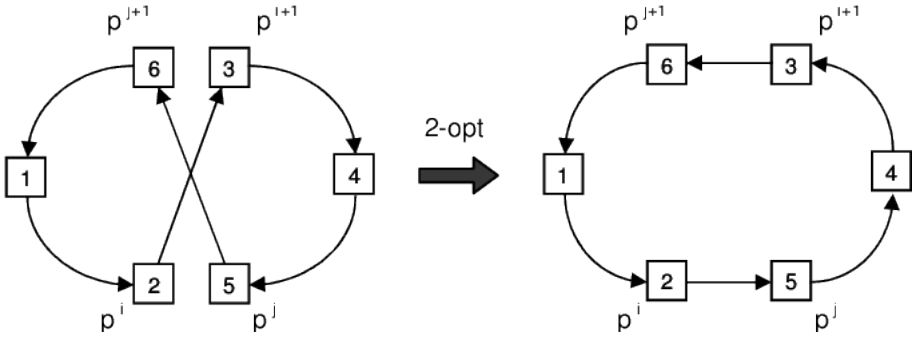
primjer inter-*shift* susjeda bio bi:

$$S^{8,3}_{inter\_shift} = \left[4 \quad \mathbf{2} \quad 6 \quad \underline{9} \quad 1 \quad 3 \quad 7 \quad 5 \quad \underline{8} \quad 0\right].$$

Može se primijetiti da se prilikom pretraživanja susjedstva inter-*shift* gradovi mogu pomicati izvan svojih originalnih ruta, dok se graničnici (skladišta) također mogu pomicati, što rezultira rutama različitih duljina u odnosu na originalno rješenje.

2.4.3. Susjedstva intra-*2-opt* i inter-*2-opt*

Susjedstvo intra-*2-opt* definirano je kao skup svih rješenja S_{intra_2-opt} koja iz originalnog rješenja S nastaju prekidanjem i suprotnim spajanjem dviju veza između gradova



Slika 2.1: Ilustracija procedure *2-opt* [4]

koje se *nalaze na istoj ruti*. Drugim riječima, riječ je o skupu svih rješenja koja iz originalnog rješenja S nastaju okretanjem (inverzijom) podniza gradova koji se nalaze *na istoj ruti*. Općeniti primjer lokalne pretrage *2-opt* vidljiv je na slici 2.1.

Za originalno rješenje:

$$S = \left[4 \quad 6 \quad \underline{9} \quad \mathbf{1} \quad \mathbf{3} \quad \mathbf{7} \quad \mathbf{5} \quad 2 \quad \underline{8} \quad 0\right],$$

primjer intra-*2-opt* susjeda bio bi:

$$S^{1,1,4}_{intra\_2-opt} = \left[4 \quad 6 \quad \underline{9} \quad \mathbf{5} \quad \mathbf{7} \quad \mathbf{3} \quad \mathbf{1} \quad 2 \quad \underline{8} \quad 0\right].$$

Slično tome, susjedstvo inter-*2-opt* definirano je kao skup svih rješenja S_{inter_2-opt} koja iz originalnog rješenja S nastaju prekidanjem i suprotnim spajanjem dviju veza između gradova koje se nalaze *bilo gdje u nizu*. Drugim riječima, riječ je o skupu svih rješenja koja iz originalnog rješenja S nastaju okretanjem (inverzijom) podniza gradova koji se nalaze bilo gdje u nizu. Za originalno rješenje:

$$S = \left[4 \quad \mathbf{6} \quad \mathbf{\underline{9}} \quad \mathbf{1} \quad \mathbf{3} \quad \mathbf{7} \quad 5 \quad 2 \quad \underline{8} \quad 0\right],$$

primjer inter-*2-opt* susjeda bio bi:

$$S^{2,6}_{inter\_2-opt} = \left[4 \quad \mathbf{7} \quad \mathbf{3} \quad \mathbf{1} \quad \mathbf{\underline{9}} \quad \mathbf{6} \quad 5 \quad 2 \quad \underline{8} \quad 0\right].$$

Lako je uočljivo da, kao i kod susjedstava inter-*swap* te inter-*shift*, pretraživanje susjedstva inter-*2-opt* može rezultirati rutama različitih duljina u odnosu na originalno rješenje S.

3. Taksonomija metaheuristika

Metaheuristike su algoritamski okviri visoke razine koji pružaju smjernice ili strategije za razvijanje heurističkih optimizacijskih algoritama [6]. Prema definiciji, metaheuristike su problemski neovisne. Međutim, problemski specifične implementacije heurističkih optimizacijskih algoritama koje su nastale prema smjernicama metaheuristike, također se nazivaju metaheuristikama.

Metaheuristički algoritmi uvijek su heuristički — nisu egzaktni, već se u velikoj mjeri oslanjaju na nasumičnost. Iako egzaktni algoritmi uvijek rezultiraju optimalnim rješenjem u konačnom vremenu, njihovo vrijeme izvršavanja je u velikom broju slučajeva nekoliko redova veličine preveliko i neisplativo, posebice kod problema velikih dimenzija. Nedeterministički metaheuristički algoritmi će, s druge strane, u „prihvatljivom“ vremenu rezultirati „dovoljno dobrim“ rješenjima.

Metaheuristike se generalno mogu taksonomski mogu razložiti na nekoliko skupina [6]:

- metaheuristike bazirane na lokalnom pretraživanju — nova (bolja) rješenja se pronalaze iterativnim nanošenjem relativno malih promjena jednom rješenju;
- konstruktivne metaheuristike — počevši od nule, rješenje se postupno gradi sve dok se ne konstruira potpuno rješenje;
- metaheuristike bazirane na populaciji — rješenja se pronalaze iterativnim kombiniranjem obilježja rješenja koja se nalaze u postojećem skupu rješenja — populaciji.

Navedene skupine nisu međusobno isključive — naprotiv, velik broj metaheuristika kombinira obilježja više skupina (hibridne metaheuristike).

Za potrebe ovog rada razmatraju se sljedeće metaheuristike:

1. Genetski algoritam (GA) (engl. *Genetic algorithm*, u daljnjem tekstu „GA“) — metaheuristika bazirana na populaciji s elementima lokalnog pretraživanja;
2. Diferencijalna evolucija (engl. *Differential evolution*, u daljnjem tekstu „DE“) — metaheuristika bazirana na populaciji;

3. Optimizacijski algoritam zasnovan na unatražnom pretraživanju (engl. *Backtracking search optimization algorithm*, u daljnjem tekstu „BSA“) — metaheuristika bazirana na populaciji;

4. Algoritam optimizacije rojem čestica (engl. *Particle swarm optimization*, u daljnjem tekstu „PSO“) — metaheuristika bazirana na populaciji;

5. Algoritam mravlje kolonije (engl. *Ant colony optimization*, u daljnjem tekstu „ACO“) — konstruktivna metaheuristika bazirana na populaciji;

6. Algoritam umjetne kolonije pčela (engl. *Artificial bee colony*, u daljnjem tekstu „ABC“) — metaheuristika bazirana na populaciji i lokalnom pretraživanju;

7. Tabu pretraživanje (engl. *Tabu search*, u daljnjem tekstu „TS“)— metaheuristika bazirana na lokalnom pretraživanju koja pamti već pretraženi prostor;

8. Simulirano kaljenje (engl. *Simulated annealing*, u daljnjem tekstu „SA“) — metaheuristika bazirana na lokalnom pretraživanju.

Implementacije svih navedenih metaheuristika paralelizirane su pomoću „*bazena dretvi*“ (engl. *thread pool*). Detalji načina paralelizacije bit će opisani za svaki algoritam zasebno.

3.1. Genetski algoritam

GA ostvaren u programskom rješenju je paralelni eliminacijski GA s troturnirskom selekcijom koji koristi permutacijski prikaz rješenja. Algoritam koristi sljedeće parametre:

- n — veličina populacije;
- m — broj generacija;
- p_m — vjerojatnost mutacije;
- m_r — stopa mortaliteta.

Na početku izvođenja algoritma, populacija se popunjava nasumičnim *valjanim* jedinkama — jedinkama koje zadovoljavaju kapacitetno ograničenje. Tijekom svake iteracije algoritma, nasumično se biraju po tri jedinke. Najlošija od triju jedinki (jedinka s najvećom vrijednošću evaluacijske funkcije) se izbacuje, a zamjenjuje ju nova — dijete preostalih dviju jedinki. Ovaj postupak se ponavlja do ispunjenja stope mortaliteta, nakon čega se prelazi na sljedeću iteraciju.

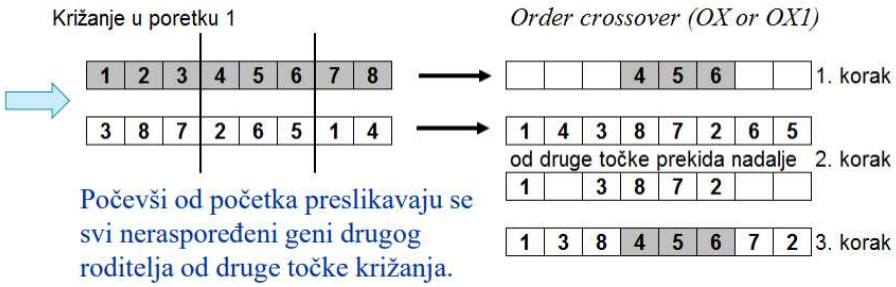
Stvaranje i evaluacija jedinki čine računski zahtjevnu proceduru koja je neovisna o stvaranju i evaluaciji drugih jedinki, stoga zajedno tvore jedan neovisan zadatak za bazen dretvi. Stvaranje jedinke provodi se na standardan način:

1. genetski materijal jedinki roditelja kombinira se pomoću operatora križanja, stvarajući novu jedinku;
2. genetski materijal novostvorene jedinke se blago mijenja pomoću operatora mutacije.

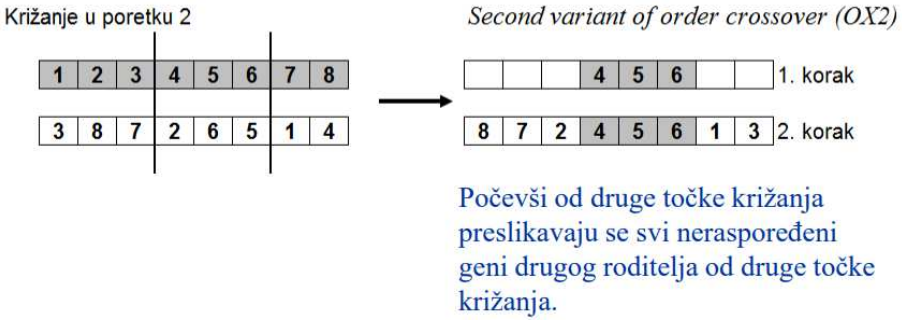
3.1.1. Križanje

Križanje se provodi na način da se nasumično odabere jedan od tri operatora križanja:

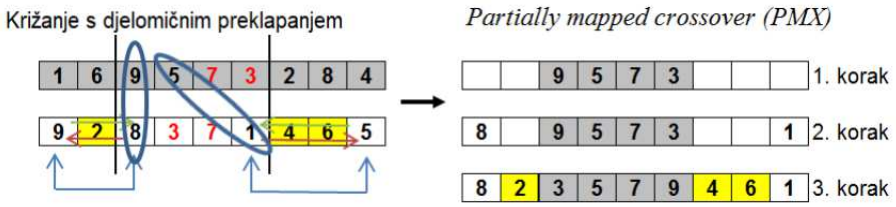
- križanje u poretku 1 (engl. *order crossover 1 — OX1*), slika 3.1;
- križanje u poretku 2 (engl. *order crossover 2 — OX2*), slika 3.2;
- križanje s djelomičnim preklapanjem (engl. *partially mapped crossover — PMX*), slika 3.3.



Slika 3.1: Križanje u poretku 1 [7]



Slika 3.2: Križanje u poretku 2 [7]



Slika 3.3: Križanje s djelomičnim preklapanjem [7]

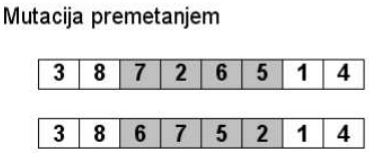
3.1.2. Mutacija

Slično križanju, mutacija se provodi na način da se nasumično odabere jedan od tri operatora mutacije:

- jednostavna mutacija (engl. *swap mutation*), slika 3.4;
- mutacija umetanjem (engl. *insert mutation*), slika 3.5;
- mutacija premetanjem (engl. *scramble mutation*), slika 3.6.

Slika 3.4: Jednostavna mutacija [7]

Slika 3.5: Mutacija umetanjem [7]



Slika 3.6: Mutacija premetanjem [7]

Pseudokod GA ostvarenog u programskom rješenju prikazan je na slici 3.7.

3.2. Diferencijalna evolucija

DE ostvarena u programskom rješenju je paralelna `DE/rand/1` (`rand` — jedinka koja mutira bira se nasumično, `1` — u mutaciji se koristi jedan diferencijalni vektor) koja koristi prikaz rješenja realnim brojevima [13, 16]. Algoritam koristi sljedeće parametre:

- n — veličina populacije;

```
procedure STVORI_JEDINKU(p1, p2)
    nova_jedinka := križaj(p1, p2)
    nova_jedinka.mutiraj()
    nova_jedinka.evaluiraj()
    return nova_jedinka
end procedure

procedure GA
    pop := inicijaliziraj_populaciju()
    najbolja := izaberi_najbolju(pop)
    tp := inicijaliziraj_bazen_dretvi()
    for (G := 0 to broj_generacija) do
        for (i := 0 to mortalitet) do
            roditelji := troturnirska_selekcija(pop)
            tp.dodaj(stvori_jedinku(roditelji[0], roditelji[1])
        end for

        tp.izvrši_zadatke()

        umetni_jedinke_u_populaciju(pop, tp.rezultati())
        najbolja := ažuriraj_najbolju(pop)
    end for
end procedure
```

Slika 3.7: GA — pseudokod

- m — broj generacija (iteracija);
- $D = N + K$ — duljina kromosoma (ovisi o dimenziji konkretnog problema);
- $F \in [0, 2]$ — faktor pojačanja u fazi mutacije;
- $CR \in [0, 1]$ — konstanta korištena u fazi križanja.

Na početku izvođenja algoritma generira se odgovarajući broj *valjanih* jedinki u permutacijskoj reprezentaciji. Populacija u permutacijskoj reprezentaciji se potom mapira u realnu domenu na način da se cjelobrojni interval $[0, N + K - 1]$ jednoliko preslika u realni interval $[0, 1]$, stvarajući populaciju reprezentiranu realnim brojevima.

Nakon inicijalizacije populacije, postupak se provodi kroz tri ponavljajuća koraka:

– mutacija;

– križanje;

– selekcija.

Navedeni koraci ponavljaju se do zadovoljenja uvjeta broja generacija (iteracija). Stvaranje (mutacija, križanje i selekcija) te evaluacija novih jedinki čine proceduru koja je neovisna o stvaranju i evaluaciji drugih jedinki, stoga zajedno tvore jedan neovisan zadatak za bazen dretvi.

3.2.1. Mutacija

Za svaku jedinku $x_{i,G}$, $i = 1, 2, \ldots, n$ u danoj generaciji G, generira se *jedinka-mutant*:

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}), \tag{3.1}$$

gdje su $r_1, r_2, r_3 \in \{1, 2, \ldots, n\}$ slučajno generirani, međusobno različiti cjelobrojni indeksi koji su također različiti od indeksa i.

Drugim riječima, *jedinka-mutant* generira se na način da se nasumična jedinka iz populacije translatira u smjeru razlike drugih dviju nasumičnih jedinki iz populacije.

3.2.2. Križanje

Za svaku jedinku $x_{i,G}$, $i = 1, 2, \ldots, n$ u danoj generaciji G, generira se „probna“ jedinka:

$$u_{ij,G+1} = \begin{cases} v_{ij,G+1}, & randn_{[0,1]} \leq CR \vee j = rand_{\{1,D\}} \\ x_{ij,G}, & randn_{[0,1]} > CR \wedge j \neq rand_{\{1,D\}} \end{cases}, j = 1, 2, \ldots, D \tag{3.2}$$

gdje $randn_{[0,1]}$ označava slučajan realni broj iz jednolike razdiobe na intervalu $[0, 1]$, a $rand_{\{1,D\}}$ označava slučajan cijeli broj iz jednolike razdiobe na skupu $\{1, 2, \ldots, D\}$. Broj $rand_{\{1,D\}}$ brine da se barem jedan parametar iz *jedinke-mutanta* $v_{i,G+1}$ prenese u probnu jedinku $u_{i,G+1}$.

3.2.3. Selekcija

Prilikom odabira jedinke koja će se u generaciji $G + 1$ nalaziti na indeksu i, jedinke $x_{i,G}$ i $u_{i,G+1}$ se evaluiraju. Jedinka koja po evaluaciji ima manji iznos funkcije kazne prelazi u sljedeću generaciju na mjesto $x_{i,G+1}$, dok se druga jedinka odbacuje.

Pseudokod DE ostvarene u programskom rješenju prikazan je na slici 3.8.

```
procedure STVORI_JEDINKU(x, b, d1, d2)
    v = stvori_mutanta(b, d1, d2)
    u = križaj(x, v)
    u.evaluiraj()
    if (u.f < x.f) then
        return u
    else
        return x
    end if
end procedure

procedure DE
    pop ≔ inicijaliziraj_populaciju()
    najbolja ≔ izaberi_najbolju(pop)
    tp ≔ inicijaliziraj_bazen_dretvi()

    for (G ≔ 0 to broj_generacija) do
        for (i ≔ 0 to n) do
            indeksi ≔ tri_nasumična_indeksa()
            tp.dodaj_zadatak(stvori_jedinku(pop[i],
pop[indeksi[0]], pop[indeksi[1]], pop[indeksi[2]]))
        end for

        tp.izvrši_zadatke()

        pop ≔ tp.rezultati()
        najbolja ≔ ažuriraj_najbolju(pop)
    end for
end procedure
```

Slika 3.8: DE — pseudokod

3.3. Optimizacijski algoritam zasnovan na unatražnom pretraživanju

BSA ostvaren u programskom rješenju je paralelni klasični BSA koji koristi prikaz rješenja realnim brojevima [5, 13]. Algoritam koristi sljedeće parametre:

- n — veličina populacije;
- m — broj generacija (iteracija);
- $D = N + K$ — duljina kromosoma (ovisi o konkretnom problemu);
- $F > 0$ — amplituda matrice smjerova pretraživanja.

Početak izvođenja algoritma identičan je početku izvođenja DE — generira se odgovarajući broj *valjanih* jedinki u permutacijskoj reprezentaciji koje se potom mapiraju u realnu domenu na način da se cjelobrojni interval $[0, N + K - 1]$ jednoliko preslika u realni interval $[0, 1]$. Novostvorene jedinke potom popunjavaju *početnu populaciju* te *početnu povijesnu populaciju*.

Nakon inicijalizacije populacija, postupak se provodi kroz četiri ponavljajuća koraka:

- selekcija-`I`;
- mutacija;
- križanje;
- selekcija-`II`.

Navedeni koraci ponavljaju se do zadovoljenja uvjeta broja generacija. Eksplicitna ograničenja kao takva ne postoje te se stoga preskače korak provjere zadovoljenja istih.

Stvaranje novih jedinki (mutacija, križanje i selekcija-`II`) zajedno s njihovom evaluacijom čini proceduru koja je neovisna o stvaranju i evaluaciji drugih jedinki, stoga zajedno tvore jedan neovisan zadatak za bazen dretvi.

3.3.1. Selekcija-`I`

U prvoj od dvije operacije selekcije utvrđuje se povijesna populacija $\mathbf{P}^{old}$. U tu se svrhu generiraju dva realna broja, a i b:

$$a, b \sim U(0, 1) \tag{3.3}$$

te se potom populacija $\mathbf{P}^{old}$ ažurira prema sljedećoj formuli:

$$\mathbf{P}^{old} \coloneqq \begin{cases} \mathbf{P}, & a < b \\ \mathbf{P}^{old}, & a \not< b \end{cases}. \tag{3.4}$$

Drugim riječima, populacija $\mathbf{P}^{old}$ se s vjerojatnošću od 50% postavlja na vrijednost trenutne populacije $\mathbf{P}$. Na kraju prve operacije selekcije potrebno je permutirati jedinke u populaciji $\mathbf{P}^{old}$.

3.3.2. Mutacija

Mutantska populacija $\mathbf{P}^{mutant}$ generira se na način da se na mjesto svake jedinke $\mathbf{P}_i^{mutant}$ upiše jedinka koja nastaje kao rezultat operacije:

$$\mathbf{P}_i^{mutant} \coloneqq \mathbf{P}_i + F \cdot \left(\mathbf{P}_i^{old} - \mathbf{P}_i\right), \tag{3.5}$$

gdje je $\left(\mathbf{P}_i^{old} - \mathbf{P}_i\right)$ redak matrice smjerova pretraživanja, a F njena amplituda.

3.3.3. Križanje

U procesu križanja generira se probna populacija $\mathbf{P}^t$. Prije početka križanja svake od jedinki, generira se binarni vektor **map** dimenzije D kojem se vrijednost svih elemenata inicijalno postavlja na 1, a potom mu se vrijednosti ažuriraju u ovisnosti o dva slučajno generirana broja $a, b \sim (0, 1)$:

$$\mathbf{map}_{u\left(1:\lfloor randn_{[0,1]} \cdot D \rfloor\right)} \coloneqq 0, \quad a < b, \tag{3.6}$$

$$\mathbf{map}_{rand_{\{1,D\}}} \coloneqq 0, \quad a \not< b, \tag{3.7}$$

gdje $randn_{[0,1]}$ označava slučajan realni broj iz jednolike razdiobe na intervalu $[0, 1]$, a $rand_{\{1,D\}}$ označava slučajan cijeli broj iz jednolike razdiobe na skupu $\{1, 2, \ldots, D\}$. Vektor u je permutirani vektor koji sadrži sve elemente iz skupa $\{1, 2, \ldots, D\}$, a $u\left(1 : \lfloor randn_{[0,1]} \cdot D \rfloor\right)$ predstavlja isječak od prvih $\lfloor randn_{[0,1]} \cdot D \rfloor$ elemenata takvog vektora.

Drugim riječima, pri svakom generiranju vektora **map** postoji 50% šanse da se točno jedan njegov element postavi na vrijednost 0 te 50% šanse da se slučajan uzorak njegovih elemenata postavi na vrijednost 0.

Nakon što je definiran vektor **map**, jedinka probne populacije $\mathbf{P}_i^t$ generira se na sljedeći način:

$$\mathbf{P}_{ij}^t \coloneqq \begin{cases} \mathbf{P}_{ij}^{mutant}, & \mathbf{map}_j = 1 \\ \mathbf{P}_{ij}, & \mathbf{map}_j = 0 \end{cases}. \tag{3.8}$$

3.3.4. Selekcija-II

Konačno, u drugoj operaciji selekcije se na pohlepan način određuju jedinke koje prelaze u sljedeću generaciju:

$$\mathbf{P}_i \coloneqq \begin{cases} \mathbf{P}_i^t, & f(\mathbf{P}_i^t) < f(\mathbf{P}_i) \\ \mathbf{P}_i, & f(\mathbf{P}_i^t) \not< f(\mathbf{P}_i) \end{cases}. \tag{3.9}$$

Drugim riječima, ako jedinka iz probne populacije $\mathbf{P}^t$ ima manju vrijednost funkcije kazne od odgovarajuće jedinke iz populacije $\mathbf{P}$, ona prelazi u sljedeću generaciju umjesto odgovarajuće jedinke iz populacije $\mathbf{P}$.

Pseudokod BSA ostvarenog u programskom rješenju prikazan je na slici 3.9.

3.4. Algoritam optimizacije rojem čestica

PSO ostvaren u programskom rješenju je paralelni klasični PSO koji koristi prikaz rješenja realnim brojevima [15]. Algoritam koristi sljedeće parametre:

- n — veličina populacije;
- m — broj iteracija;
- w — koeficijent inercije;
- c_l — koeficijent ubrzanja prema najboljem lokalnom rješenju;
- c_g — koeficijent ubrzanja prema najboljem globalnom rješenju.

Na početku izvođenja algoritma se, kao i kod prethodnih dvaju algoritama, generira odgovarajući broj *valjanih* jedinki u permutacijskoj reprezentaciji koje se potom mapiraju u realnu domenu na način da se cjelobrojni interval $[0, N + K - 1]$ jednoliko preslika u realni interval $[0, 1]$, stvarajući početnu populaciju. Brzine svih elemenata svake čestice inicijaliziraju se na vrijednost 0.

Nakon inicijalizacije populacije, postupak se provodi kroz dva ponavljajuća koraka:

- ažuriranje brzina u ovisnosti o pozicijama čestica;
- ažuriranje pozicija čestica u ovisnosti o njihovim brzinama.

Navedeni koraci ponavljaju se do zadovoljenja uvjeta broja iteracija. Ažuriranje brzina, ažuriranje pozicija te evaluacija novonastale čestice čine proceduru koja je neovisna o stvaranju i evaluaciji drugih čestica, stoga zajedno tvore neovisan zadatak za bazen dretvi.

```
procedure STVORI_JEDINKU(index)
    mutant = stvori_mutanta(index)
    probna = križaj(index, mutant)
    probna.evaluiraj()
    return selekcija2(index, probna)
end procedure

procedure BSA
    pop := inicijaliziraj_populaciju()
    pop_p := inicijaliziraj_populaciju()
    najbolja := izaberi_najbolju(pop, pop_p)
    tp := inicijaliziraj_bazen_dretvi()

    for (G := 0 to broj_generacija) do

        selekcija1()
        permutiraj(pop_p)

        for (i := 0 to N) do
            tp.dodaj_zadatak(stvori_jedinku(i))
        end for

        tp.izvrši_zadatke()

        pop := tp.rezultati()
        najbolja := ažuriraj_najbolju(pop)
    end for
end procedure
```

Slika 3.9: BSA — pseudokod

3.4.1. Ažuriranje brzina

Kako bi se ažurirale brzine čestica, za svaki element svake čestice potrebno je generirati brojeve r_l i r_g:

$$r_l, r_g \sim U(0, 1), \tag{3.10}$$

nakon čega se brzina elementa na indeksu j čestice x_i ažurira prema formuli:

$$v_{i,j} \coloneqq w \cdot v_{i,j} + c_l r_l(najbolje\_lokalno_{i,j} - x_{i,j}) + c_g r_g(najbolje\_globalno_j - x_{i,j}). \tag{3.11}$$

Dakle, svaka čestica će težiti održati svoj smjer kretanja, ali će istovremeno težiti kretati se u smjeru svoje najbolje, kao i globalno najbolje zapamćene pozicije.

3.4.2. Ažuriranje pozicija

Svaka čestica pomiče se u smjeru svoje brzine:

$$x_i \leftarrow x_i + v_i. \tag{3.12}$$

Nakon ažuriranja pozicija, svaka čestica provjerava je li nadmašila svoje lokalno ili sveukupno globalno najbolje rješenje te ih po potrebi ažurira.

Pseudokod PSO ostvarenog u programskom rješenju prikazan je na slici 3.10.

3.5. Algoritam mravlje kolonije

ACO ostvaren u programskom rješenju je nalik paralelnom sustavu mrava baziranom na rangu (AS_{Rank}) uz nekoliko dodataka, po uzoru na [17]. Tijekom izvođenja algoritma, mravi grade svoje putanje iz kojih se po završetku izgradnje stvaraju rješenja u permutacijskoj reprezentaciji. Algoritam koristi sljedeće parametre:

- n — broj mrava;
- m — broj iteracija;
- w — broj mrava koji ažuriraju feromone;
- τ_0 — početni intenzitet feromona;
- τ — matrica intenziteta feromona (ažurira se tijekom izvođenja algoritma);
- η — matrica vrijednosti heuristike (udaljenosti između gradova);
- μ — matrica vrijednosti uštede;
- $\alpha > 0$ — faktor prikupljanja informacije;
- $\beta > 0$ — faktor očekivane heuristike;
- $\gamma \geq 0$ — faktor utjecaja uštede;
- $\rho \in (0, 1]$ — stopa isparavanja feromona;

```
procedure ITERACIJA(ind)
    r_l, r_g = generiraj_brojeve_01()
    v[ind] := w · v[ind] + c_l · r_l · (best_l[ind] − pop[ind]) +
c_g · r_g · (best_g − pop[ind])
    pop[ind] = pop[ind] + v[ind]
    pop[ind].evaluiraj()
    return pop[ind]
end procedure

procedure PSO
    tp := inicijaliziraj_bazen_dretvi()
    pop := inicijaliziraj_populaciju()
    best_g := izaberi_najbolju(pop)
    best_l := pop
    for (G := 0 to broj_iteracija) do
        for (i := 0 to N) do
            tp.dodaj_zadatak(iteracija(i))
        end for
        tp.izvrši_zadatke()
        pop := tp.rezultati()
        best_g := ažuriraj_najbolju(pop)
        best_l := ažuriraj(pop)
    end for
end procedure
```

Slika 3.10: PSO — pseudokod

– $\theta > 0$ — konstanta korištena prilikom ažuriranja feromona.

U algoritam se, u svrhu poboljšanja procesa odabira sljedećeg grada, osim vrijednosti feromona τ i vrijednosti heurističke informacije η uvodi i vrijednost uštede μ. μ opisuje koliku duljinu puta mrav „uštedi“ ako iz trenutnog grada ode direktno u sljedeći grad umjesto da se iz trenutnog grada prvo vrati u skladište, a tek nakon toga ode u sljedeći grad:

$$\mu_{i,j} = d_{i,0} + d_{0,j} - d_{i,j}. \tag{3.13}$$

Na početku izvođenja algoritma, vrijednost feromona se na svim bridovima (cestama između gradova) postavlja na početnu vrijednost τ_0. Nakon toga, algoritam se

provodi kroz dva ponavljajuća koraka:

- svaki mrav pokušava izgraditi put sve dok ne izgradi *valjan* put;
- nakon što svi mravi izgrade svoje puteve, ažurira se vrijednost feromona.

Navedeni koraci ponavljaju se do zadovoljenja uvjeta broja iteracija. Izgradnja puta svakog od mrava neovisna je o izgradnji puteva ostalih mrava, stoga čini zaseban zadatak za bazen dretvi.

3.5.1. Izgradnja puta

Izgradnja puta započinje od skladišta. Pri odabiru svakog sljedećeg grada, pretražuju se svi neposjećeni gradovi čije dodavanje u rutu ne krši kapacitetno ograničenje te se dodaju u skup „mogućih“ gradova.

U slučaju da je skup mogućih gradova prazan te još vozila stoji na raspolaganju, trenutno vozilo vraća se u skladište (u put se dodaje graničnik koji predstavlja povratak u skladište). U suprotnom, ako više nema slobodnih vozila, rješenje ne može biti valjano te se stoga odbacuje, a gradnja puta započinje ispočetka.

S druge strane, ako skup mogućih gradova nije prazan, izglednost svakog od gradova iz skupa računa se kao sljedeći produkt:

$$\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta} \cdot \mu_{i,j}^{\gamma}, \tag{3.14}$$

gdje i predstavlja indeks trenutnog grada, a j predstavlja indeks mogućeg grada.

Vjerojatnost odabira svakog od mogućih gradova računa se kao njegova izglednost podijeljena sa sumom izglednosti svih mogućih gradova:

$$p(j) = \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta} \cdot \mu_{i,j}^{\gamma}}{\sum_{r=1}^{R} \tau_{i,r}^{\alpha} \cdot \eta_{i,r}^{\beta} \cdot \mu_{i,r}^{\gamma}}, \tag{3.15}$$

gdje je R broj mogućih gradova.

Sljedeći grad se potom bira slučajno, proporcionalno vjerojatnosti njegovog odabira. Postupak se ponavlja do izgradnje valjanog rješenja.

3.5.2. Ažuriranje feromona

Nakon što svi mravi izgrade svoja rješenja, slijedi ažuriranje feromona. Mravi se sortiraju u ovisnosti o kvaliteti pronađenog rješenja kako bi mravi s boljim rješenjima u većoj mjeri pojačali intenzitet feromona na svojem putu. Prvi korak ažuriranja je globalno smanjenje intenziteta feromona (isparavanje):

$$\tau_{i,j} \leftarrow \left(\rho + \frac{\theta}{L_{avg}}\right) \tau_{i,j}, \tag{3.16}$$

gdje je L_{avg} srednja vrijednost duljine puteva koje su mravi izgradili u trenutnoj iteraciji, dok je θ konstanta zadana kao parametar na razini algoritma [17].

Sljedeći korak ažuriranja je povećanje intenziteta feromona na putevima mrava koji su pronašli najbolja rješenja. Globalno najbolje rješenje te $w-1$ najboljih rješenja iz trenutne iteracije ažuriraju feromone na svojim putevima prema formuli:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \frac{(w-\lambda)}{L_\lambda}, \lambda = \{0, 1, \ldots, w-1\} \tag{3.17}$$

gdje je $\lambda = 0$ u slučaju globalno najboljeg rješenja, a $\lambda = \{1, \ldots, w-1\}$ za $\lambda - to$ najbolje rješenje iz trenutne iteracije. L_λ je ukupna duljina $\lambda - tog$ rješenja (L_0 je ukupna duljina globalno najboljeg rješenja).

Pseudokod ACO ostvarenog u programskom rješenju prikazan je na slici 3.11.

3.6. Algoritam umjetne kolonije pčela

ABC ostvaren u programskom rješenju je paralelni klasični ABC koji koristi permutacijsku reprezentaciju rješenja [8, 18]. Algoritam koristi sljedeće parametre:

- $n = n_r + n_p$ — veličina populacije (ukupan broj pčela);
- m — broj iteracija;
- n_r — početni broj pčela radnica;
- n_p — broj pčela promatrača;
- F — dugovječnost izvora hrane.

ABC kao osnovni mehanizam pretraživanja prostora stanja koristi nasumičnu lokalnu pretragu — iz nasumičnog susjedstva od susjedstava definiranih u odlomku 2.4 generira se nasumično rješenje.

Na početku izvođenja algoritma populacija pčela radnica popunjava se nasumično generiranim *valjanim* jedinkama u permutacijskoj reprezentaciji. Pčele promatrači se eksplicitno ne inicijaliziraju, budući da se koriste samo kao mehanizam dodatnog pretraživanja susjedstva.

Nakon inicijalizacije populacije, postupak se provodi kroz tri ponavljajuća koraka:

- let pčela radnica;
- let pčela promatrača;
- let pčela izviđača.

```
procedure STVORI_PUT
    put := prazan_niz()
    while put.nije_gotov() do
        mogući := neposjećeni_gradovi_koji_ne_premašuju_kapacitet
        if (mogući.prazan() && postoji_još_vozila()) then
            put.dodaj(graničnik)
        else if (mogući.prazan() && !postoji_još_vozila()) then
            počni_ispočetka()
        else
            izglednosti := izračunaj_izglednosti(mogući)
            vjerojatnosti := izglednosti / sum(izglednosti)
            put.dodaj(proporcionalno_odaberi(vjerojatnosti))
        end if
    end while
end procedure

procedure ACO
    tp := inicijaliziraj_bazen_dretvi()
    for (G := 0 to broj_iteracija) do
        for (i := 0 to N) do
            tp.dodaj_zadatak(izgradi_put())
        end for

        tp.izvrši_zadatke()
        rezultati := tp.rezultati()
        najbolji := ažuriraj_najboljeg(rezultati)
        ažuriraj_feromone()
    end for
end procedure
```

Slika 3.11: ACO — pseudokod

Navedeni koraci ponavljaju se do zadovoljenja uvjeta broja iteracija. Budući da letovi pčela radnica i letovi pčela promatrača uključuju stvaranje i evaluaciju novih rješenja te da je svaki let međusobno neovisan o svim drugim letovima, svaki od njih tvori neovisan zadatak za bazen dretvi.

3.6.1. Let pčela radnica

Svaka pčela radnica leti do svojeg trenutnog izvora hrane (rješenja) te u njegovom susjedstvu traži novi izvor hrane (generiranje nasumičnog susjeda).

Ako je novi izvor hrane bolji od prethodnog, pčela ga pamti i prenosi informaciju o njegovoj kvaliteti ostatku kolonije provodeći „ples“. Intenzitet plesa i-te pčele jednak je funkciji dobrote njenog rješenja, a računa se prema formuli:

$$f_{d,i} = \max_{j=1}^{n_r}(f_{k,j}) - f_{k,i}. \tag{3.18}$$

Drugim riječima, intenzitet plesa i-te pčele jednak je vrijednosti funkcije kazne rješenja i-te pčele radnice oduzetoj od maksimalne vrijednost funkcije kazne svih rješenja pčela radnica.

U slučaju da novi izvor hrane nije bolji od prethodnog, a pčela oko njega leti manje od F iteracija, ona ostatku kolonije prenosi informaciju o kvaliteti *starog izvora hrane*. U suprotnom, ako pčela oko izvora hrane leti barem F iteracija, taj izvor hrane smatra se potrošenim. Pčela radnica ga potom napušta i postaje pčela izviđač.

3.6.2. Let pčela promatrača

Pčele promatrači čekaju da se pčele radnice vrate iz pretraživanja susjedstava svojih izvora hrane te promatraju njihove plesove. Svaka od njih potom bira jedan od izvora hrane koje su pronašle pčele radnice vjerojatnošću proporcionalnom intenzitetu plesa odgovarajuće pčele radnice. Pčela promatrač potom traži novi izvor hrane pretražujući susjedstvo odabranog izvora (generiranje nasumičnog susjeda). Ako je pronađeni izvor hrane bolji, pčela promatrač dojavljuje tu informaciju odgovarajućoj pčeli radnici, koja se potom seli na novopronađeni izvor hrane. Na ovaj se način intenzificira pretraga u susjedstvima boljih rješenja.

3.6.3. Let pčela izviđača

Pčele radnice čiji su izvori hrane presušili postaju pčele izviđači koje traže novi izvor hrane pretražujući susjedstvo najboljeg pronađenog rješenja (generiranje nasumičnih susjeda). Nakon što pčela izviđač pronađe bilo koje valjano rješenje iz susjedstva najboljeg pronađenog rješenja, ona ga pamti te ponovno postaje pčelom radnicom.

Pseudokod ABC ostvarenog u programskom rješenju prikazan je na slici 3.12.

```
procedure LET_RADNICE(pčela)
    novi_izvor := nasumični_valjani_susjed(pčela.izvor)
    return bolji_izvor(novi_izvor, pčela.izvor)
end procedure
procedure LET_PROMATRAČA(izvor)
    novi_izvor := nasumični_valjani_susjed(izvor)
    if (novi_izvor.bolji_od(izvor)) then
        return novi_izvor
    end if
end procedure
procedure ABC
    tp := inicijaliziraj_bazen_dretvi()
    pop_radnice := inicijaliziraj_populaciju()
    for (G := 0 to broj_iteracija) do
        for (i := 0 to n_r) do
            tp.dodaj_zadatak(let_radnice(pop_radnice[i]))
        end for
        tp.izvrši_zadatke()
        rez_radnice := tp.rezultati()
        najbolji := ažuriraj_najbolji(rez_radnice)

        for (i := 0 to n_p) do
            izvor := izaberi_proporcionalno_prema_plesu(rez_radnice)
            tp.dodaj_zadatak(let_promatrača(izvor))
        end for
        tp.izvrši_zadatke()
        rez_promatrači := tp.rezultati()
        ažuriraj_rješenja_radnica(pop_radnice, rez_promatrači)
        najbolji := ažuriraj_najbolji(rez_promatrači)

        for (i := 0 to n_r) do
            if (pop_radnice[i].izvor.presušio()) then
                pop_radnice[i].izvor = nasumični_valjani_susjed(najbolji)
            end if
        end for
    end for
end procedure
```

Slika 3.12: ABC — pseudokod

3.7. Tabu pretraživanje

TS ostvaren u programskom rješenju je paralelno klasično TS koje koristi permutacijsku reprezentaciju rješenja, a u tabu listu dodaje konkretna rješenja [15]. Algoritam koristi sljedeće parametre:

- m — broj iteracija;
- T — veličina tabu liste (engl. *tabu tenure*).

Na početku izvođenja algoritma generira se *valjano* rješenje, postavlja se kao trenutno te se dodaje u tabu listu. Nakon inicijalizacije, postupak se provodi kroz tri ponavljajuća koraka:

- pretraživanje cijelog susjedstva trenutnog rješenja;
- odabir sljedećeg rješenja;
- dodavanje novog rješenja u tabu listu.

Navedeni koraci ponavljaju se do zadovoljenja uvjeta broja iteracija. Budući da je generiranje svakog susjeda (uključujući njegovu evaluaciju) neovisno o generiranju ostalih susjeda, svako generiranje susjeda čini neovisan zadatak za bazen dretvi.

3.7.1. Pretraživanje susjedstva

Kao kandidati za sljedeće trenutno rješenje generiraju se svi susjedi iz svakog od susjedstava definiranih u odlomku 2.4.

3.7.2. Odabir sljedećeg rješenja i dodavanje u tabu listu

Svi valjani susjedi generirani u prošlom koraku sortiraju se uzlazno prema funkciji kazne (bolja rješenja su na početku niza, a lošija na kraju). Nakon što je niz sortiran, kroz njega se prolazi te se kao trenutno rješenje za sljedeću iteraciju postavlja prvo rješenje u nizu koje se ne nalazi na tabu listi, a potom se i ono dodaje na tabu listu.

Pseudokod TS ostvarenog u programskom rješenju prikazan je na slici 3.13.

3.8. Simulirano kaljenje

SA ostvareno u programskom rješenju je klasično SA s više paralelnih hlađenja koje koristi permutacijsku reprezentaciju rješenja [15]. Algoritam koristi sljedeće parametre:

```
procedure TS
   tp ≔ inicijaliziraj_bazen_dretvi()
   trenutno ≔ generiraj_valjano_rješenje()
   tl ≔ inicijalizira_tabu_listu()

   for (G ≔ 0 to broj_iteracija) do
      for (s in sva_susjedstva(trenutno)) do
         tp.dodaj_zadatak(stvori_i_evaluiraj(s))
      end for
      tp.izvrši_zadatke()

      susjedi ≔ tp.rezultati()
      sortiraj(susjedi)
      najbolji ≔ ažuriraj_najboljeg(susjedi)

      for (s in susjedi) do
         if (tl.ne_sadrži(s)) then
            trenutno ≔ s
            tl.dodaj(s)
            break
         end if
      end for
   end for
end procedure
```

Slika 3.13: TS — pseudokod

- n — broj paralelnih hlađenja;
- T_s — početna temperatura;
- T_f — konačna temperatura;
- f_d — funkcija snižavanja temperatura (dekrementa) — linearni dekrement, geometrijski dekrement, vrlo spori dekrement;
- $B > 0$ (samo u slučaju linearnog dekrementa) — vrijednost za koju se trenutna temperatura snižava;
- $\alpha \in (0, 1)$ (samo u slučaju geometrijskog dekrementa) — faktor za koji se

trenutna temperatura snižava;

– $\beta > 0$ (samo u slučaju vrlo sporog dekrementa) — konstanta korištena pri snižavanju temperature.

Na početku izvođenja algoritma, trenutna temperatura T_t postavlja se na vrijednost T_s, a potom se generira nasumično *valjano* rješenje i postavlja se kao trenutno. Postupak se potom, sve do zadovoljenja uvjeta konačne temperature, provodi kroz tri ponavljajuća koraka:

– generiranje nasumičnog susjeda trenutnog rješenja;

– odabir sljedećeg rješenja;

– snižavanje temperature.

Budući da se u algoritmu paralelno provodi n hlađenja te da su hlađenja međusobno neovisna, generiranje svakog susjeda kod svakog od hlađenja čini zaseban zadatak za bazen dretvi.

3.8.1. Generiranje nasumičnog susjeda

Kao kandidat za sljedeće trenutno rješenje generira se jedan nasumični susjed iz nasumičnog susjedstva od susjedstava definiranih u odlomku 2.4.

3.8.2. Odabir sljedećeg rješenja

Trenutno rješenje može ostati isto ili može prijeći u novo rješenje generirano u prošlom koraku. Vjerojatnost za prelazak u novo rješenje je:

$$p(susjed) = \frac{1}{1 + e^{\frac{L_{susjed} - L_{trenutno}}{T_t}}}, \tag{3.19}$$

pri čemu je L_x ukupna duljina puta rješenja x.

3.8.3. Snižavanje temperature

U ovisnosti o odabranoj funkciji snižavanja temperature, trenutna temperatura snižava se na sljedeći način:

– ako je odabrana funkcija linearnog dekrementa:

$$T_t \leftarrow T_t - B; \tag{3.20}$$

– ako je odabrana funkcija geometrijskog dekrementa:

$$T_t \leftarrow T_t \cdot \alpha; \tag{3.21}$$

– ako je odabrana funkcija vrlo sporog dekrementa:

$$T_t \leftarrow \frac{T_t}{1 + \beta \cdot T_t}. \tag{3.22}$$

Pseudokod SA ostvarenog u programskom rješenju prikazan je na slici 3.14.

```
procedure SA
   tp := inicijaliziraj_bazen_dretvi()
   trenutna := generiraj_valjana_rješenja()
   t_t := t_s

   while (t_t > t_f) do
      for (i := 0 to N) do
         tp.dodaj_zadatak(stvori_nasumičnog_susjeda(trenutna[i]))
      end for

      tp.izvrši_zadatke()

      susjedi := tp.rezultati()
      najbolje := ažuriraj_najbolje(susjedi)

      for (i := 0 to veličina(susjedi)) do
         p := (1 + e^((L_susjedi[i] - L_trenutna[i]) / T_t))^-1

         if (generiraj_realni_broj_01() < p) then
            trenutna[i] := susjedi[i]
         end if

      end for
      t_t := reduciraj_temperaturu(t_t)
   end while
end procedure
```

Slika 3.14: SA — pseudokod

4. Opis programskog ostvarenja

Programsko ostvarenje napisano je u programskom jeziku C++20. Za rad s dretvama i bazenima dretvi korištene su biblioteke `<bind.hpp>`, `<asio.hpp>` i `<thread/future.hpp>` iz radnog okvira Boost [1]. Za pseudo-nasumično generiranje brojeva koristi se generator iz biblioteke `<random>`.

4.1. Struktura programskog ostvarenja

UML dijagram razreda programskog ostvarenja prikazan je na slici 4.1.

U razredu `VRP` definirane su sve varijable koje opisuju konkretan problem usmjeravanja vozila ograničenog kapaciteta na način prikladan potrebama algoritama. Razred `BasePhenotype` definira zajedničko sučelje za razrede `PermutationPhenotype` i `RKRealPhenotype` koji predstavljaju programsko ostvarenje permutacijskog prikaza, odnosno prikaza rješenja realnim brojevima na način prikladan potrebama algoritama.

Konačno, razred `AbstractAlgorithm` sadrži referencu na primjerak razreda `VRP` te deklarira apstraktne metode `run()` i `printBest()` koje svaka od konkretnih implementacija algoritama definira na odgovarajući način. Svaka od konkretnih implementacija algoritama sadrži i pokazivače na jedan ili više primjeraka razreda `PermutationPhenotype`, odnosno `RKRealPhenotype`.

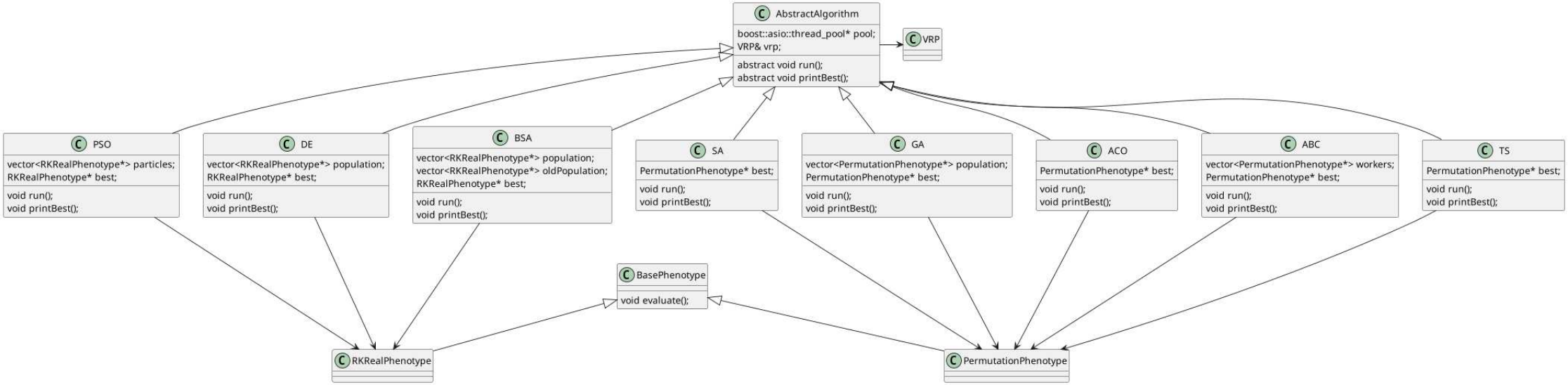
4.2. Učitavanje podataka

Primjerak razreda `VRP` može se instancirati sljedećom naredbom:

```
VRP vrp("../data/A/A-n32-k5.vrp");
```

Unutar zagrada navodi se put do datoteke [1] u kojoj je zapisan konkretan problem

[1] Sve ulazne datoteke s opisima konkretnih problema preuzete su s [14]



Slika 4.1: UML dijagram razreda programskog ostvarenja

usmjeravanja vozila. Unutar konstruktora razreda `VRP` poziva se parser ulazne datoteke koji učitava koordinate gradova, potražnje, broj vozila i njihov kapacitet.

Primjer strukture ulazne datoteke prikazan je na slici 4.2.

```
NAME : A-n32-k5
COMMENT : (Augerat et al, No of trucks: 5, Optimal value: 784)
TYPE : CVRP
DIMENSION : 32
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 100
NODE_COORD_SECTION
 1 82 76
 2 96 44
 3 50 5
...
 32 98 5
DEMAND_SECTION
1 0
2 19
3 21
...
32 9
DEPOT_SECTION
 1
 -1
EOF
```

Slika 4.2: Primjer strukture ulazne datoteke

Iz naziva datoteke mogu se iščitati dimenzije problema — problem `A-n32-k5` je problem s 32 grada (uključujući skladište) i 5 vozila. Nakon nekoliko opisnih linija koje se pri učitavanju mogu ignorirati, u liniji `CAPACITY` naveden je kapacitet vozila. U odsječku koji započinje linijom `NODE_COORD_SECTION`, u svakoj liniji navedena su tri broja — prvi označava indeks grada, dok druga dva označavaju x i y koordinate grada, redom. Slično tome, u odsječku `DEMAND_SECTION` navedene su potražnje u svakom od gradova. Konačno, odsječak `DEPOT_SECTION` označava grad u kojem se nalazi skladište — u gornjem slučaju je to grad broj 1. Datoteka završava linijama `-1`

i `EOF`.

Nakon što se podaci učitaju, na temelju koordinata računa se euklidska udaljenost između svakog para gradova. Izračunate udaljenosti se potom spremaju u kvadratnu matricu namijenjenu samo za čitanje (engl. *read-only*).

4.3. Instanciranje primjerka konkretnog algoritma

Nakon što je instanciran primjerak razreda `VRP`, potrebno je instancirati primjerak konkretnog algoritma sljedećom naredbom:

```
GA ga(vrp);
```

Razred `GA` označava genetski algoritam, a primjerci ostalih algoritama mogu se instancirati pozivanjem konstruktora odgovarajućeg razreda (`DE`, `BSA`, `PSO`, `ACO`, `ABC`, `TS`, `SA`).

Stvaranjem primjerka bilo kojeg od navedenih razreda, parametri algoritma postavljaju se na pretpostavljene vrijednost. Parametri se mogu podešavati i ručno, kao što je pokazano u sljedećem isječku koda:

```
ga.setPopulationSize(100);
ga.setPM(0.03);
```

Za algoritme koji u svom osnovnom obliku ne pretražuju susjedstvo (GA, DE, BSA, PSO, ACO), moguće je „aktivirati“ pretraživanje susjedstva:

```
ga.setSearchNeighbourhood(true);
```

Susjedstvo se pretražuje na način da se nakon generiranja svakog novog rješenja generira jedan njegov nasumičan susjed. Ako je slučajni susjed bolje rješenje od originalnog novog rješenja, susjed se sprema kao novo rješenje.

Također, za algoritme koji pri početku izvođenja generiraju jedno ili više rješenja (svi algoritmi osim ACO), moguće je definirati metodu generiranja početnih rješenja:

```
ga.setConstructionMode(BasePhenotype::GUIDED_CLASSIC);
```

Ako se kao metoda generiranja početnih rješenja zada `GUIDED_CLASSIC`, početna rješenja će biti generirana „polu-pohlepnom“ metodom, po uzoru na izgradnju rješenja kod algoritma GRASP. U suprotnom, ako se kao metoda generiranja početnih rješenja zada `RANDOM_FEASIBLE`, početna rješenja će biti *valjana* nasumična rješenja.

Konačno, izvođenje algoritma može se pokrenuti pozivom metode `run()`:

```
ga.run();
```

5. Eksperimentalni rezultati

U nastavku slijede eksperimentalni rezultati i statistika izvođenja programskog ostvarenja na tri konkretna problema usmjeravanja vozila ograničenog kapaciteta:

1. `A-n32-k5` — problem manjih dimenzija, 32 grada i 5 vozila;
2. `A-n60-k9` — problem srednjih dimenzija, 60 gradova i 9 vozila;
3. `A-n80-k10` — problem većih dimenzija, 80 gradova i 10 vozila.

Za svaki od navedenih problema, pokrenuto je svih osam algoritama. Za svaki od algoritama, cilj je bio pronaći skup parametara koji rezultiraju najboljim rješenjem. Za pronađeni optimalni skup parametara, provedeno je testiranje utjecaja pretraživanja susjedstva (kod algoritama koji u svojem osnovnom obliku ne pretražuju susjedstvo) te utjecaja metode generiranja početnih rješenja (polu-pohlepno — „PP“, ili nasumično valjano — „NV“) kod algoritama za koje to ima smisla. Za najbolje tako dobiveno rješenje iscrtan je graf kvalitete rješenja kroz iteracije, a najbolje rješenje dobiveno tijekom svih testova zabilježeno je u svrhu konačne zajedničke statistike.

Za svaku konfiguraciju provedeno je deset testova. U tablicama su zabilježena *najbolja* rješenja dobivena kroz deset testova (bijele tablice), kao i *prosječna* vremena izvođenja algoritama (sive tablice). Najbolji rezultat u svakoj tablici istaknut je drugom bojom.

Svi testovi provedeni su na istom procesoru — $\mathrm{AMD\ Ryzen^{TM}\ 7\ 4800}H$ s osam fizičkih jezgri baznog takta $2.9\,GHz$, koristeći bazen dretvi veličine 16.

5.1. Genetski algoritam

Genetski algoritam korišten u svim testovima odgovara algoritmu opisanom u odlomku 3.1. Veličina populacije fiksirana je na $n = 100$, broj iteracija fiksiran je na $m = 100000$, dok je stopa mortaliteta fiksirana na $m_r = 30\%$. Za svaki od triju pro-

blema ispituje se utjecaj vrijednosti vjerojatnosti mutacije na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja algoritma.

Konkretne vrijednosti vjerojatnosti mutacije za koje su provedeni testovi na sva tri problema su: 0.25%, 0.5%, 1%, 2% i 3%.

Najbolja rješenja postignuta kroz deset testova za svaku od vrijednosti vjerojatnosti mutacije na sva tri problema navedena su u tablici 5.1, a vizualizirana na grafovima 5.1, 5.2 i 5.3. Prosječna vremena izvođenja algoritama u ovisnosti o vjerojatnosti mutacije navedena su u tablici 5.2.

Instanca	Optimum	Vjerojatnost mutacije				
		0.25%	0.5%	1.0%	2.0%	3.0%
A-n32-k5	787.082	787.809	787.082	787.082	787.082	787.082
A-n60-k9	1355.799	1477.140	1458.651	1441.409	1437.864	1435.694
A-n80-k10	1766.5	2014.550	1966.706	1950.774	1936.439	1906.272

Tablica 5.1: Vrijednosti najboljih rješenja GA u ovisnosti o vjerojatnosti mutacije

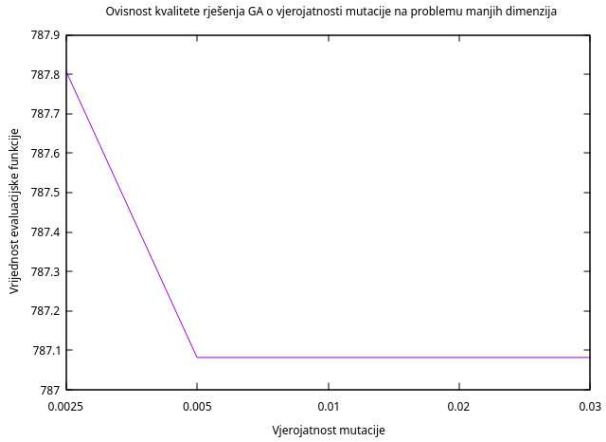
Instanca	Vjerojatnost mutacije				
	0.25%	0.5%	1.0%	2.0%	3.0%
A-n32-k5	27088.6	27303.5	27412.6	27419.9	27306.2
A-n60-k9	30291.0	30145.1	30087.9	29877.7	29849.1
A-n80-k10	32412.2	32470.3	31788.5	32073.9	32394.2

Tablica 5.2: Prosječna duljina izvođenja (u ms) GA u ovisnosti o vjerojatnosti mutacije

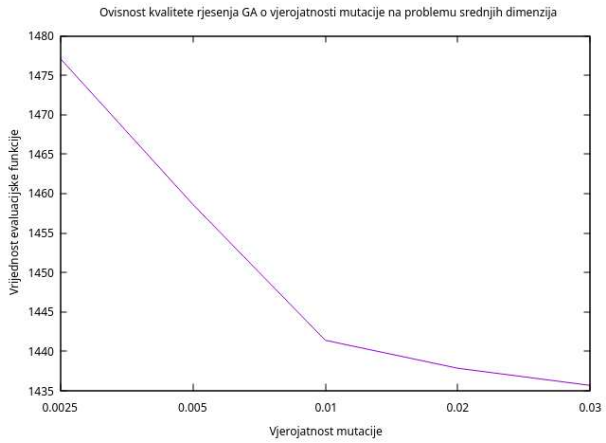
Kao optimalna vrijednost vjerojatnosti mutacije kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema srednjih i većih dimenzija postiže minimum za vrijednost vjerojatnosti mutacije $p_m = 3\%$.

Kod problema manjih dimenzija, minimum se postiže za više vrijednosti vjerojatnosti mutacije. Kao optimalna vjerojatnost mutacije među njima bira se ona za koju je postignuto minimalno prosječno vrijeme izvođenja, a to je $p_m = 0.5\%$.

Nakon utvrđivanja optimalnih vrijednosti vjerojatnosti mutacije, za svaki od triju problema ispitan je utjecaj pretraživanja susjedstva i metode generiranja početne populacije na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu je vjerojatnost mutacije fiksirana na pronađenu optimalnu vrijednost. Najbolja rješenja za navedene testove prikazana su u tablici 5.3, dok su prosječna vremena izvođenja algoritma prikazana u tablici 5.4.



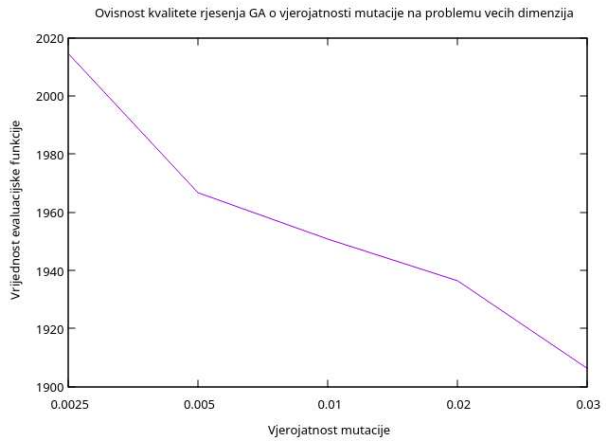
Graf 5.1: Ovisnost kvalitete rješenja GA o vjerojatnosti mutacije na problemu manjih dimenzija



Graf 5.2: Ovisnost kvalitete rješenja GA o vjerojatnosti mutacije na problemu srednjih dimenzija

Instanca	Optimum	Početna populacija	Pretraživanje susjedstva	
			NE	DA
A-n32-k5	787.082	NV	787.082	787.082
		PP	811.230	787.082
A-n60-k9	1355.799	NV	1393.368	1400.537
		PP	1397.461	1431.434
A-n80-k10	1766.5	NV	1902.691	1838.116
		PP	1880.324	1844.975

Tablica 5.3: Vrijednosti najboljih rješenja GA u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije



Graf 5.3: Ovisnost kvalitete rješenja GA o vjerojatnosti mutacije na problemu većih dimenzija

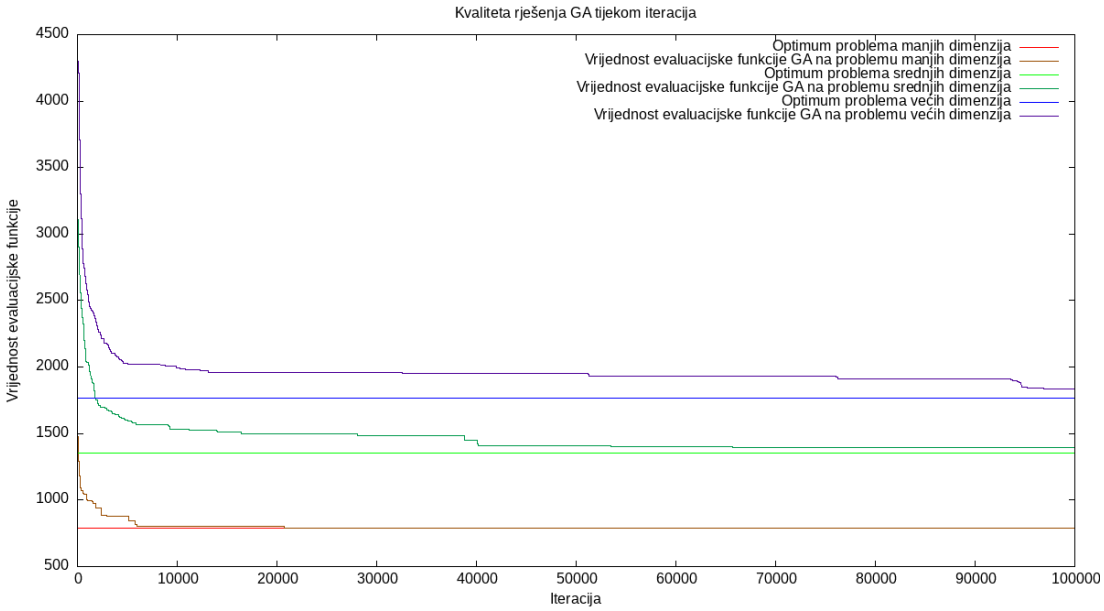
Instanca	Početna populacija	Pretraživanje susjedstva	
		NE	DA
A-n32-k5	NV	28504.2	30627.3
	PP	29004.6	31663.2
A-n60-k9	NV	31487.8	36454.5
	PP	33019.4	38322.8
A-n80-k10	NV	33325.8	41052.5
	PP	36968.7	44392.7

Tablica 5.4: Prosječna duljina izvođenja (u ms) GA u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Kod problema srednjih dimenzija, to je konfiguracija koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima i koja *ne provodi* lokalnu pretragu, dok je kod problema većih dimenzija optimalna konfiguracija ona koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima i koja *provodi* lokalnu pretragu.

Kod problema manjih dimenzija, minimum se ponovno postiže za više konfiguracija pa se kao optimalna među njima bira konfiguracija za koju je postignuto minimalno prosječno vrijeme izvođenja. To je konfiguracija koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima i koja *ne provodi* lokalnu pretragu.

Napredak rješenja kroz iteracije uz navedene optimalne konfiguracije za svaki od triju problema vidljiv je na grafu 5.4.



Graf 5.4: Kvaliteta rješenja GA kroz iteracije

Konačno, najbolja rješenja koja je GA pronašao na definiranim problemima su:

$$X_{GA}^{n32} = 787.082 \tag{5.1}$$

$$X_{GA}^{n60} = 1393.368 \tag{5.2}$$

$$X_{GA}^{n80} = 1838.116 \tag{5.3}$$

5.2. Diferencijalna evolucija

Diferencijalna evolucija korištena u svim testovima odgovara algoritmu opisanom u odlomku 3.2. Veličina populacije fiksirana je na $n = 50$, a maksimalni broj iteracija na $m = 100000$. Izvođenje algoritma može se prekinuti i ranije, ako kroz 20000 iteracija ne dođe do poboljšanja najboljeg rješenja. Za svaki od triju problema ispituje se utjecaj faktora pojačanja u fazi mutacije (F) te utjecaj konstante korištene u fazi križanja (CR) na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja algoritma.

Konkretne vrijednosti faktora F za koje su provedeni testovi na sva tri problema su: 0.1, 0.25, 0.5, 1, 1.5 i 2; dok su konkretne vrijednosti konstante CR: 0.1, 0.25, 0.5, 0.75, 0.9, 1.

Najbolja rješenja postignuta kroz deset testova za svaku od kombinacija vrijednosti parametara F i CR na sva tri problema navedena su u tablici 5.5, dok su prosječna vremena izvođenja algoritma u ovisnosti o parametrima F i CR navedena u tablici 5.6.

Inst.	Opt.	F	CR					
			0.1	0.25	0.5	0.75	0.9	1
A-n32-k5	787.082	0.1	1188.460	1161.920	1307.900	1276.240	1090.490	1711.890
		0.25	1191.970	1026.730	1359.260	1241.220	1028.690	1719.960
		0.5	1157.580	1348.670	1384.450	1327.890	1047.820	1682.380
		1	1071.350	1143.550	1397.730	1278.380	1077.630	1548.040
		1.5	1120.140	1114.560	1379.430	1355.110	1087.760	1591.640
		2	1145.990	1150.260	1316.960	1384.750	1124.840	1696.840
A-n60-k9	1355.799	0.1	3112.570	3188.770	3129.040	2914.860	2612.260	3131.920
		0.25	3072.390	3158.780	3098.250	3096.870	2544.820	3015.210
		0.5	2973.540	3212.400	3011.370	3214.230	2685.710	3126.130
		1	3140.850	3135.430	3143.510	3028.630	2672.320	3114.480
		1.5	2994.200	3035.520	3137.300	3069.170	2739.610	3108.760
		2	3140.840	3050.890	3131.230	3109.710	2718.280	3175.110
A-n80-k10	1766.5	0.1	4348.730	4311.350	4362.390	4384.280	4192.540	4307.350
		0.25	4415.330	4270.820	4288.230	4233.950	4181.400	4364.770
		0.5	4174.940	4307.350	4375.770	4326.870	4322.340	4322.730
		1	4280.010	4456.800	4397.840	4366.920	4282.250	4387.830
		1.5	4246.880	4391.290	4345.260	4295.080	4345.330	4315.630
		2	4394.580	4322.650	4243.060	4390.080	4183.810	4304.170

Tablica 5.5: Vrijednosti najboljih rješenja DE u ovisnosti o vrijednostima parametara F i CR

Inst.	F	CR					
		0.1	0.25	0.5	0.75	0.9	1
A-n32-k5	0.1	7529.4	12444.7	18946.6	23678.9	25036.1	7461.5
	0.25	9621.9	20353.3	14956.4	23793.7	29495.3	7411.9
	0.5	17709.8	14963.0	16668.6	20804.9	24242.3	7395.4
	1	15564.0	17789.6	16376.9	17117.2	25119.0	7369.1
	1.5	13125.8	19150.4	14664.5	19748.3	20058.0	7360.5
	2	14228.0	18689.1	16299.1	19271.6	24328.4	7360.5
A-n60-k9	0.1	10445.0	10922.8	10930.8	11208.2	44265.9	10905.7
	0.25	10806.4	10825.4	10842.2	10832.8	35135.3	10835.6
	0.5	10704.2	10717.2	10781.9	10773.8	38973.3	10806.9
	1	10698.1	10728.0	10762.7	10750.8	31456.0	10736.0
	1.5	10493.7	10551.6	10794.5	10765.1	32485.8	10842.1
	2	10719.5	10738.6	10766.8	10757.7	40366.9	10826.8
A-n80-k10	0.1	13161.1	13176.1	13210.0	13201.6	20172.4	13213.9
	0.25	13111.3	13136.7	13200.5	13192.0	13198.6	13189.1
	0.5	13090.9	13114.7	13191.9	13172.4	19232.4	13200.1
	1	13098.5	13110.2	13143.2	13135.8	17169.5	13205.2
	1.5	13092.0	13130.1	13169.7	13142.5	16076.9	13231.7
	2	13095.6	13120.3	13176.9	13127.5	17754.3	13008.2

Tablica 5.6: Prosječna duljina izvođenja (u ms) DE u ovisnosti o vrijednostima parametara F i CR

Instanca	Optimum	Početna populacija	Pretraživanje susjedstva	
			NE	DA
A-n32-k5	787.082	NV	1113.370	1312.680
		PP	1128.950	1331.790
A-n60-k9	1355.799	NV	2661.830	2391.500
		PP	2731.870	2397.630
A-n80-k10	1766.5	NV	4198.230	4328.450
		PP	4402.550	4335.660

Tablica 5.7: Vrijednosti najboljih rješenja DE u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

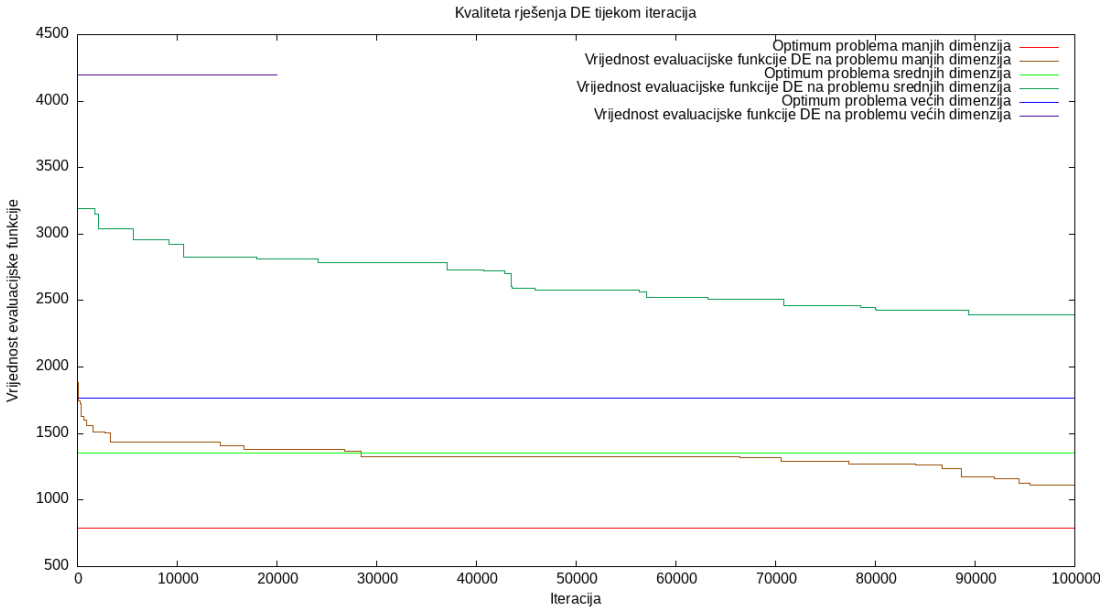
Instanca	Početna populacija	Pretraživanje susjedstva	
		NE	DA
A-n32-k5	NV	19538.1	25303.9
	PP	20878.7	20506.5
A-n60-k9	NV	42118.0	75467.0
	PP	32320.7	81681.7
A-n80-k10	NV	13741.5	20584.7
	PP	15244.1	22417.6

Tablica 5.8: Prosječna duljina izvođenja DE (u ms) u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Kao optimalna kombinacija vrijednosti parametara F i CR kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema manjih dimenzija postiže minimum za vrijednosti parametara $(F, CR) = (0.25, 0.25)$, kod problema srednjih dimenzija postiže minimum za vrijednosti parametara $(F, CR) = (0.25, 0.9)$, dok kod problema većih dimenzija postiže minimum za vrijednosti parametara $(F, CR) = (0.5, 0.1)$.

Nakon utvrđivanja optimalnih vrijednosti parametara F i CR, za svaki od triju problema ispitan je utjecaj pretraživanja susjedstva i metode generiranja početne populacije na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu su vrijednosti parametara F i CR fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.7, dok su prosječna vremena izvođenja algoritma prikazana u tablici 5.8.

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost



Graf 5.5: Kvaliteta rješenja DE kroz iteracije

evaluacijske funkcije postiže minimum. Kod problema manjih i većih dimenzija, to je konfiguracija koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima i koja *ne provodi* lokalnu pretragu. S druge strane, kod problema srednjih dimenzija, optimalna konfiguracija je ona koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima i koja *provodi* lokalnu pretragu.

Napredak rješenja kroz iteracije uz navedene optimalne konfiguracije za svaki od triju problema vidljiv je na grafu 5.5.

Konačno, najbolja rješenja koja je DE pronašla na definiranim problemima su:

$$X_{DE}^{n32} = 1026.730 \tag{5.4}$$

$$X_{DE}^{n60} = 2391.500 \tag{5.5}$$

$$X_{DE}^{n80} = 4174.94 \tag{5.6}$$

5.3. Optimizacijski algoritam zasnovan na unatražnom pretraživanju

BSA korišten u svim testovima odgovara algoritmu opisanom u odlomku 3.3. Veličina populacije fiksirana je na $n = 80$, a maksimalni broj iteracija na $m = 100000$. Izvođenje algoritma može se prekinuti i ranije, ako kroz 20000 iteracija ne dođe do promjene najboljeg rješenja. Za svaki od triju problema ispituje se utjecaj amplitude matrice

smjerova pretraživanja (F) na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja algoritma.

Konkretne vrijednosti faktora F za koje su provedeni testovi na sva tri problema su: 0.1, 0.25, 0.5, 1, 1.5 i 2.

Najbolja rješenja postignuta kroz deset testova za svaku od vrijednosti parametra F na sva tri problema navedena su u tablici 5.9, dok su prosječna vremena izvođenja algoritma u ovisnosti o parametru F navedena u tablici 5.10.

Instanca	Optimum	F					
		0.1	0.25	0.5	1	1.5	2
A-n32-k5	787.082	1347.740	1390.390	1355.580	1440.480	1207.010	1207.240
A-n60-k9	1355.799	2921.510	2938.490	3018.400	2921.500	3061.830	3110.820
A-n80-k10	1766.5	4270.100	4166.260	4190.280	4072.480	4325.190	4307.750

Tablica 5.9: Vrijednosti najboljih rješenja BSA u ovisnosti o vrijednosti faktora F

Instanca	F					
	0.1	0.25	0.5	1	1.5	2
A-n32-k5	12038.1	12799.1	14841.6	8566.7	12064.1	14915.5
A-n60-k9	19266.9	16773.8	16813.5	12487.3	13581.9	15938.5
A-n80-k10	18230.3	23658.8	22176.5	15479.5	15417.5	15895.6

Tablica 5.10: Prosječna duljina izvođenja BSA (u ms) u ovisnosti o vrijednosti faktora F

Kao optimalna vrijednost parametra F kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema manjih dimenzija postiže minimum za vrijednost parametra $F = 1.5$, dok kod problema srednjih i većih dimenzija postiže minimum za vrijednost parametra $F = 1$.

Nakon utvrđivanja optimalnih vrijednosti parametra F, za svaki od triju problema ispitan je utjecaj pretraživanja susjedstva i metode generiranja početne populacije na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu je vrijednost parametra F fiksirana na pronađenu optimalnu vrijednost. Najbolja rješenja za navedene testove prikazana su u tablici 5.11, dok su prosječna vremena izvođenja algoritma prikazana u tablici 5.12.

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Kod problema manjih i srednjih dimenzija, to je konfiguracija koja početnu populaciju popunjava *polu-pohlepnim* rješenjima i koja

Instanca	Optimum	Početna populacija	Pretraživanje susjedstva	
			NE	DA
A-n32-k5	787.082	NV	1028.550	789.350
		PP	1186.280	789.170
A-n60-k9	1355.799	NV	2763.360	1488.700
		PP	2953.180	1427.780
A-n80-k10	1766.5	NV	4125.770	1893.750
		PP	4147.730	1912.030

Tablica 5.11: Vrijednosti najboljih rješenja BSA u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Instanca	Početna populacija	Pretraživanje susjedstva	
		NE	DA
A-n32-k5	NV	14625.2	61205.8
	PP	15240.1	53221.7
A-n60-k9	NV	12498.2	22906.5
	PP	14901.0	25334.1
A-n80-k10	NV	15917.8	30175.9
	PP	20968.1	36389.5

Tablica 5.12: Prosječna duljina izvođenja BSA (u ms) u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

provodi lokalnu pretragu. S druge strane, kod problema većih dimenzija, optimalna konfiguracija je ona koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima i koja *provodi* lokalnu pretragu.

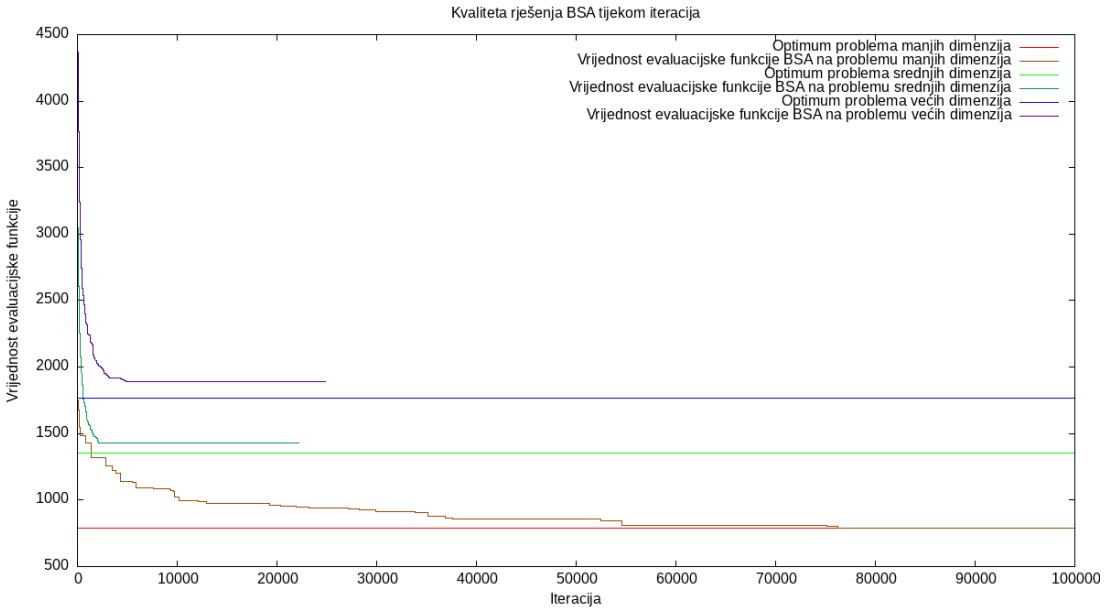
Napredak rješenja kroz iteracije uz navedene optimalne konfiguracije za svaki od triju problema vidljiv je na grafu 5.6.

Konačno, najbolja rješenja koja je BSA pronašao na definiranim problemima su:

$$X_{BSA}^{n32} = 789.170 \tag{5.7}$$

$$X_{BSA}^{n60} = 1427.780 \tag{5.8}$$

$$X_{BSA}^{n80} = 1893.750 \tag{5.9}$$



Graf 5.6: Kvaliteta rješenja BSA kroz iteracije

5.4. Algoritam optimizacije rojem čestica

PSO korišten u svim testovima odgovara algoritmu opisanom u odlomku 3.4. Veličina populacije fiksirana je na $n = 80$, a maksimalni broj iteracija na $m = 100000$. Izvođenje algoritma može se prekinuti i ranije, ako kroz 20000 iteracija ne dođe do promjene bilo kojeg lokalno ili globalno najboljeg rješenja. Za svaki od triju problema ispituje se utjecaj koeficijenta ubrzanja prema najboljem lokalnom rješenju (c_l), koeficijenta ubrzanja prema najboljem globalnom rješenju (c_g) i koeficijenta inercije (w) na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja algoritma.

Budući da se ispituje ovisnost kvalitete rješenja i vremena izvođenja o tri parametra, testovi su prvo provedeni za parametre c_l i c_g uz treći parametar fiksiran na vrijednost $w = 0.4$, nakon čega su testovi ponovno provedeni za parametar w uz parametre c_l i c_g fiksirane na vrijednosti koje rezultiraju najboljim rješenjem.

Konkretne vrijednosti parametara c_l i c_g za koje su provedeni testovi na sva tri problema su: 1, 1.5, 2 i 3; dok su konkretne vrijednosti parametra w: 0.1, 0.2, 0.4, 0.5, 0.75 i 1.

Najbolja rješenja postignuta kroz deset testova za svaku od kombinacija vrijednosti parametara c_l i c_g uz fiksirani parametar w na sva tri problema navedena su u tablici 5.13, dok su prosječna vremena izvođenja algoritma u ovisnosti o parametrima c_l i c_g navedena u tablici 5.14.

Kao optimalna kombinacija vrijednosti parametara c_l i c_g kod svakog problema

Inst.	Opt.	c_l	c_g			
			1	1.5	2	3
A-n32-k5	787.082	1	1288.080	1347.940	1289.060	1064.130
		1.5	1192.440	1237.510	1217.020	1408.290
		2	1173.610	1194.660	1042.990	1428.410
		3	1076.940	1414.840	1419.400	1679.060
A-n60-k9	1355.799	1	3095.870	3125.620	2960.770	3140.940
		1.5	3031.370	3177.920	3126.630	2916.690
		2	2960.250	3154.820	3095.680	2848.470
		3	3116.370	3034.700	3093.360	3141.370
A-n80-k10	1766.5	1	4365.340	4394.870	4320.520	4297.410
		1.5	4331.730	4246.120	4249.500	4233.730
		2	4213.080	4350.490	4315.990	4263.270
		3	4209.220	4323.280	4317.340	4278.330

Tablica 5.13: Vrijednosti najboljih rješenja PSO u ovisnosti o vrijednostima parametara c_l i c_g

Inst.	c_l	c_g			
		1	1.5	2	3
A-n32-k5	1	28231.5	24826.0	18593.5	48473.6
	1.5	45140.0	30313.4	30237.4	48565.6
	2	48068.4	38690.6	48530.1	48845.0
	3	48801.4	48526.2	48366.9	10783.4
A-n60-k9	1	15676.9	15679.8	15659.9	15637.0
	1.5	15677.8	15680.6	15649.9	15636.3
	2	15675.6	16228.8	15626.2	15602.1
	3	15632.4	15628.4	15620.6	14861.9
A-n80-k10	1	19585.8	19572.7	19575.7	19473.1
	1.5	19740.5	19616.6	19584.7	19454.2
	2	19547.1	19486.7	19451.8	19408.0
	3	19440.8	19427.6	19455.7	17984.6

Tablica 5.14: Prosječna duljina izvođenja PSO (u ms) u ovisnosti o vrijednostima parametara c_l i c_g

bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema manjih dimenzija postiže minimum za vrijednosti parametara

$(c_l, c_g) = (2, 2)$, kod problema srednjih dimenzija za vrijednosti parametara $(c_l, c_g) = (2, 3)$, a kod problema većih dimenzija za vrijednosti parametara $(c_l, c_g) = (3, 1)$.

Nakon utvrđivanja optimalnih vrijednosti parametara c_l i c_g uz fiksiranu vrijednost parametra w, za svaki od problema ispitan je utjecaj vrijednosti parametra w na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu su vrijednosti parametara c_l i c_g fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.15, dok su prosječna vremena izvođenja algoritma prikazana u tablici 5.16.

Instanca	Optimum	w					
		0.1	0.2	0.4	0.5	0.75	1
A-n32-k5	787.082	1073.090	1030.170	1042.990	1134.800	1366.180	1579.420
A-n60-k9	1355.799	3010.750	3098.260	2848.470	3067.780	3147.920	3047.070
A-n80-k10	1766.5	4379.300	4184.400	4209.220	4330.690	4186.580	4229.880

Tablica 5.15: Vrijednosti najboljih rješenja PSO u ovisnosti o vrijednosti parametra w

Instanca	w					
	0.1	0.2	0.4	0.5	0.75	1
A-n32-k5	48058.5	48534.8	48530.1	48007.3	48020.6	13365.5
A-n60-k9	15243.0	15485.8	15602.1	15484.9	15149.7	14810.3
A-n80-k10	19295.9	19305.7	19440.8	20108.5	19493.3	18474.4

Tablica 5.16: Prosječna duljina izvođenja PSO (u ms) u ovisnosti o vrijednosti parametra w

Kao optimalna vrijednost parametra w kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema manjih dimenzija postiže minimum za vrijednosti parametara $(c_l, c_g, w) = (2, 2, 0.2)$, kod problema srednjih dimenzija za vrijednosti parametara $(c_l, c_g, w) = (2, 3, 0.4)$, a kod problema većih dimenzija za vrijednosti parametara $(c_l, c_g, w) = (3, 1, 0.2)$.

Nakon utvrđivanja optimalnih vrijednosti parametara, za svaki od triju problema ispitan je utjecaj pretraživanja susjedstva i metode generiranja početne populacije na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu su vrijednosti parametara c_l, c_g i w fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.17, dok su prosječna vremena izvođenja algoritma prikazana u tablici 5.18.

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Kod problema srednjih i većih dimenzija, to

Instanca	Optimum	Početna populacija	Pretraživanje susjedstva	
			NE	DA
A-n32-k5	787.082	NV	1078.480	787.082
		PP	1036.170	787.082
A-n60-k9	1355.799	NV	3017.270	1428.750
		PP	3129.180	1417.630
A-n80-k10	1766.5	NV	4354.120	1874.300
		PP	4329.900	1872.790

Tablica 5.17: Vrijednosti najboljih rješenja PSO u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Instanca	Početna populacija	Pretraživanje susjedstva	
		NE	DA
A-n32-k5	NV	50012.8	71637.8
	PP	51245.4	72958.4
A-n60-k9	NV	15817.5	79913.6
	PP	17085.4	82803.3
A-n80-k10	NV	19621.7	132994.6
	PP	22233.8	137971.6

Tablica 5.18: Prosječna duljina izvođenja PSO (u ms) u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

je konfiguracija koja početnu populaciju popunjava *polu-pohlepnim* rješenjima i koja *provodi* lokalnu pretragu.

Kod problema manjih dimenzija, minimum se postiže za više konfiguracija pa se kao optimalna među njima bira konfiguracija za koju je postignuto minimalno prosječno vrijeme izvođenja. To je konfiguracija koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima i koja *provodi* lokalnu pretragu.

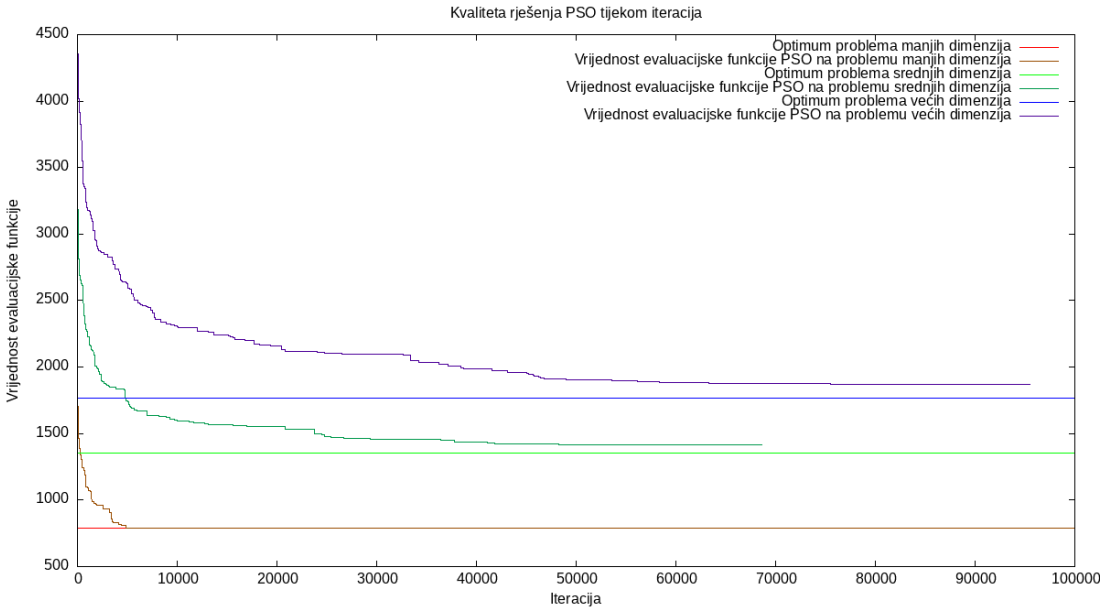
Napredak rješenja kroz iteracije uz navedene optimalne konfiguracije za svaki od triju problema vidljiv je na grafu 5.7.

Konačno, najbolja rješenja koja je PSO pronašao na definiranim problemima su:

$$X_{PSO}^{n32} = 787.082 \tag{5.10}$$

$$X_{PSO}^{n60} = 1417.630 \tag{5.11}$$

$$X_{PSO}^{n80} = 1872.790 \tag{5.12}$$



Graf 5.7: Kvaliteta rješenja PSO kroz iteracije

5.5. Algoritam mravlje kolonije

ACO korišten u svim testovima odgovara algoritmu opisanom u odlomku 3.5. Veličina populacije fiksirana je na $n = 30$, maksimalni broj iteracija na $m = 100000$, a početni intenzitet feromona na $\tau_0 = 10^{-6}$. Broj mrava koji ažuriraju feromone fiksiran je na $w = 3$, stopa isparavanja feromona na $\rho = 0.8$, a konstanta θ na 80 [17]. Izvođenje algoritma može se prekinuti i ranije, ako kroz 20000 iteracija ne dođe do promjene najboljeg rješenja. Za svaki od triju problema ispituje se utjecaj faktora prikupljanja informacije (α), faktora očekivane heuristike (β) i faktora utjecaja uštede (γ) na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja algoritma.

Budući da se ispituje ovisnost kvalitete rješenja i vremena izvođenja o tri parametra, testovi su prvo provedeni za parametre α i β, uz treći parametar fiksiran na vrijednost $\gamma = 3$, nakon čega su testovi ponovno provedeni za parametar γ uz parametre α i β fiksirane na vrijednosti koje rezultiraju najboljim rješenjem.

Konkretne vrijednosti parametara α i β za koje su provedeni testovi na sva tri problema su: 0.5, 1, 2, 3.5 i 5; dok su konkretne vrijednosti parametra γ: 0.5, 1, 2, 3 i 5.

Najbolja rješenja postignuta kroz deset testova za svaku od kombinacija vrijednosti parametara α i β uz fiksirani parametar γ na sva tri problema navedena su u tablici 5.19, dok su prosječna vremena izvođenja algoritma navedena u tablici 5.20.

Kao optimalna kombinacija vrijednosti parametara α i β kod svakog problema

Inst.	Opt.	α	β				
			0.5	1	2	3.5	5
A-n32-k5	787.082	0.5	807.714	805.171	814.544	824.438	824.558
		1	804.038	804.038	807.627	817.577	827.677
		2	819.135	817.229	822.746	826.952	824.837
		3.5	846.046	836.415	826.566	846.007	834.617
		5	846.143	863.765	857.317	821.854	840.455
A-n60-k9	1355.799	0.5	1500.727	1541.385	1495.827	1514.667	1544.959
		1	1391.141	1444.044	1463.998	1449.669	1506.483
		2	1498.301	1489.447	1493.979	1529.042	1495.972
		3.5	1562.339	1499.827	1572.976	1511.912	1515.322
		5	1581.045	1544.795	1568.733	1520.986	1531.114
A-n80-k10	1766.5	0.5	2160.577	2099.773	2042.779	2023.224	2014.432
		1	1927.131	1898.858	1954.182	1986.859	1998.545
		2	2026.794	2040.433	2001.315	2002.141	2033.200
		3.5	2101.338	2080.279	2049.831	2010.445	2018.994
		5	2179.723	2133.557	2125.932	2054.976	2050.583

Tablica 5.19: Vrijednosti najboljih rješenja ACO u ovisnosti o vrijednostima parametara α i β

bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema srednjih dimenzija postiže minimum za vrijednosti parametara $(\alpha, \beta) = (1, 0.5)$, a kod problema većih dimenzija za vrijednosti parametara $(\alpha, \beta) = (1, 1)$.

Kod problema manjih dimenzija, minimum se postiže za više kombinacija vrijednosti parametara α i β pa se kao optimalna među njima bira kombinacija za koju je postignuto minimalno prosječno vrijeme izvođenja, a to je kombinacija $(\alpha, \beta) = (1, 1)$.

Nakon utvrđivanja optimalnih vrijednosti parametara α i β uz fiksiranu vrijednost parametra γ, za svaki od problema ispitan je utjecaj vrijednosti parametra γ na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu su vrijednosti parametara α i β fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.21, dok su prosječna vremena izvođenja algoritma prikazana u tablici 5.22.

Kao optimalna vrijednost parametra γ kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema manjih i većih dimenzija postiže minimum za vrijednosti parametara $(\alpha, \beta, \gamma) =$

Inst.	α	β				
		0.5	1	2	3.5	5
A-n32-k5	0.5	48456.3	56901.8	39062.9	39263.4	37959.8
	1	43101.3	36052.2	42119.3	28035.5	35083.1
	2	24749.5	24878.0	29249.6	24347.8	30053.4
	3.5	20464.9	26830.2	23126.3	28071.2	26557.5
	5	24614.9	24167.1	27966.6	23373.6	26947.3
A-n60-k9	0.5	93490.2	85405.3	85605.8	91890.9	113248.1
	1	166353.8	139647.6	130255.2	109114.5	116287.8
	2	114849.9	104198.4	99387.0	111408.4	104877.0
	3.5	111889.5	98724.2	111669.6	107775.5	80160.2
	5	93759.7	102225.2	116860.4	112732.8	88646.0
A-n80-k10	0.5	198931.0	181486.5	149595.5	158554.9	167230.7
	1	242509.7	237585.3	245536.2	176451.5	211093.4
	2	259505.7	256069.7	205124.9	155423.5	166640.3
	3.5	302210.2	245766.6	259251.4	181210.6	209150.6
	5	217966.1	263874.9	213957.0	213947.4	167521.2

Tablica 5.20: Prosječna duljina izvođenja (u ms) ACO u ovisnosti o vrijednostima parametara α i β

Instanca	Optimum	γ				
		0.5	1	2	3	5
A-n32-k5	787.082	803.432	804.038	804.037	804.038	804.037
A-n60-k9	1355.799	1438.097	1432.326	1419.410	1391.141	1412.705
A-n80-k10	1766.5	1870.925	1928.410	1926.189	1898.858	1929.771

Tablica 5.21: Vrijednosti najboljih rješenja ACO u ovisnosti o vrijednosti parametra γ

Instanca	γ				
	0.5	1	2	3	5
A-n32-k5	34108.6	33359.9	30563.0	36052.2	42893.7
A-n60-k9	181252.9	169846.2	169265.0	166353.8	176308.2
A-n80-k10	263046.7	278404.1	183062.5	237585.3	230192.9

Tablica 5.22: Prosječna duljina izvođenja (u ms) ACO u ovisnosti o vrijednosti parametra γ

$(1, 1, 0.5)$, a kod problema srednjih dimenzija za vrijednosti parametara $(\alpha, \beta, \gamma) = (1, 0.5, 3)$.

Nakon utvrđivanja optimalnih vrijednosti parametara, za svaki od triju problema ispitan je utjecaj pretraživanja susjedstva na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu su vrijednosti parametara α, β i γ fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.23, dok su prosječna vremena izvođenja algoritma prikazana u tablici 5.24.

Instanca	Optimum	Pretraživanje susjedstva	
		NE	DA
A-n32-k5	787.082	814.764	793.377
A-n60-k9	1355.799	1419.740	1424.306
A-n80-k10	1766.5	1955.379	1908.561

Tablica 5.23: Vrijednosti najboljih rješenja ACO u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Instanca	Pretraživanje susjedstva	
	NE	DA
A-n32-k5	31242.2	30852.8
A-n60-k9	143155.2	147862.9
A-n80-k10	284942.3	299968.7

Tablica 5.24: Prosječna duljina izvođenja (u ms) ACO u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Kod problema manjih i većih dimenzija, to je konfiguracija koja *provodi* lokalnu pretragu. S druge strane, kod problema srednjih dimenzija, to je konfiguracija koja *ne provodi* lokalnu pretragu.

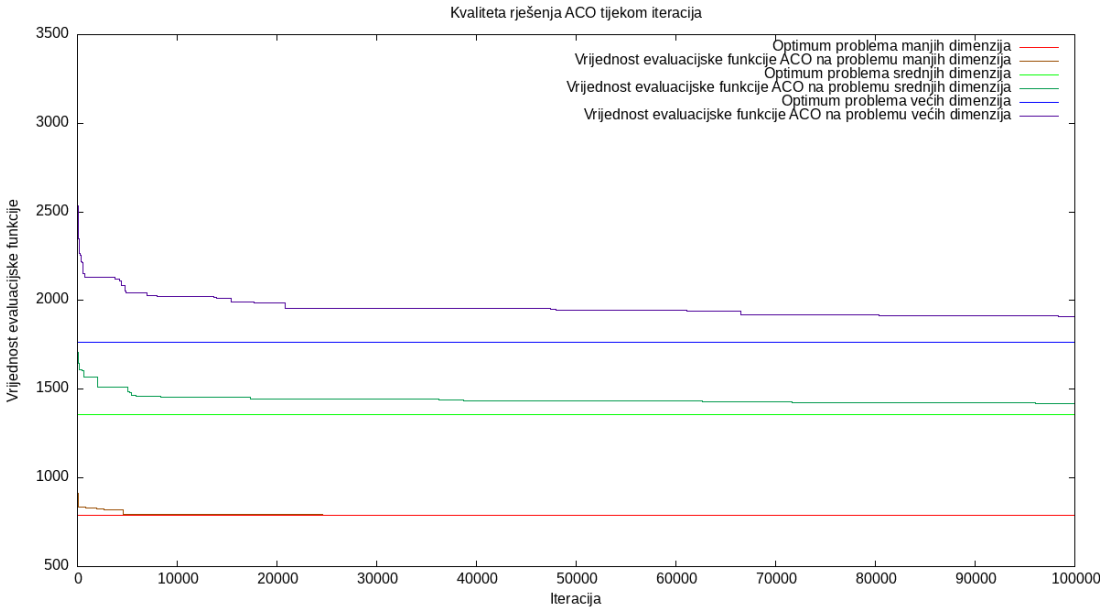
Napredak rješenja kroz iteracije uz navedene optimalne konfiguracija za svaki od triju problema vidljiv je na grafu 5.8.

Konačno, najbolja rješenja koja je ACO pronašao na definiranim problemima su:

$$X_{ACO}^{n32} = 793.377 \tag{5.13}$$

$$X_{ACO}^{n60} = 1391.141 \tag{5.14}$$

$$X_{ACO}^{n80} = 1870.925 \tag{5.15}$$



Graf 5.8: Kvaliteta rješenja ACO kroz iteracije

5.6. Algoritam umjetne kolonije pčela

ABC korišten u svim testovima odgovara algoritmu opisanom u odlomku 3.6. Maksimalni broj iteracija fiksiran je na $m = 100000$. Dugovječnost izvora hrane računa se kao $F = n_r$, dok je $n_r = n_p = n/2$ [11]. Za svaki od triju problema ispituje se utjecaj veličine populacije (n) na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja.

Konkretne veličine populacije za koje su provedeni testovi na sva tri problema su: 10, 20, 30, 50, 80 i 100.

Najbolja rješenja postignuta kroz deset testova za svaku od vrijednosti veličine populacije na sva tri problema navedena su u tablici 5.25, dok su prosječna vremena izvođenja navedena u tablici 5.26.

Instanca	Optimum	Veličina populacije					
		10	20	30	50	80	100
A-n32-k5	787.082	787.082	787.082	787.082	787.082	787.082	787.082
A-n60-k9	1355.799	1387.322	1364.500	1360.593	1360.593	1355.799	1355.799
A-n80-k10	1766.5	1827.010	1815.107	1814.582	1798.240	1790.322	1797.784

Tablica 5.25: Vrijednosti najboljih rješenja ABC u ovisnosti o veličini populacije

Kao optimalna veličina populacije kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema većih dimenzija postiže minimum za veličinu populacije $n = 80$.

Instanca	Veličina populacije					
	10	20	30	50	80	100
A-n32-k5	12250.8	18104.3	23082.9	33034.8	47905.6	58752.6
A-n60-k9	15845.0	23019.5	28546.0	40067.2	55718.1	66129.8
A-n80-k10	19274.4	25658.8	31835.1	44074.5	60369.5	71323.4

Tablica 5.26: Prosječna duljina izvođenja ABC (u ms) u ovisnosti o veličini populacije

Kod problema manjih dimenzija, minimum se postiže za *sve* veličine populacije, dok se kod problema srednjih dimenzija postiže za dvije veličine populacije. Kao optimalne među njima biraju se veličine populacija za koje je postignuto minimalno prosječno vrijeme izvođenja, a to su $n = 10$ za problem manjih dimenzija i $n = 80$ za problem srednjih dimenzija.

Nakon utvrđivanja optimalnih veličina populacija, za svaki od triju problema ispitan je utjecaj metode generiranja početne populacije na kvalitetu rješenja i prosječno vrijeme izvođenja, pri čemu su veličine populacija fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.27, dok su prosječna vremena izvođenja prikazana u tablici 5.28.

Instanca	Optimum	Početna populacija	
		NV	PP
A-n32-k5	787.082	787.082	787.082
A-n60-k9	1355.799	1355.799	1355.799
A-n80-k10	1766.5	1785.996	1778.320

Tablica 5.27: Vrijednosti najboljih rješenja ABC u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Instanca	Početna populacija	
	NV	PP
A-n32-k5	12993.4	13734.5
A-n60-k9	59295.4	59044.6
A-n80-k10	62692.1	64392.0

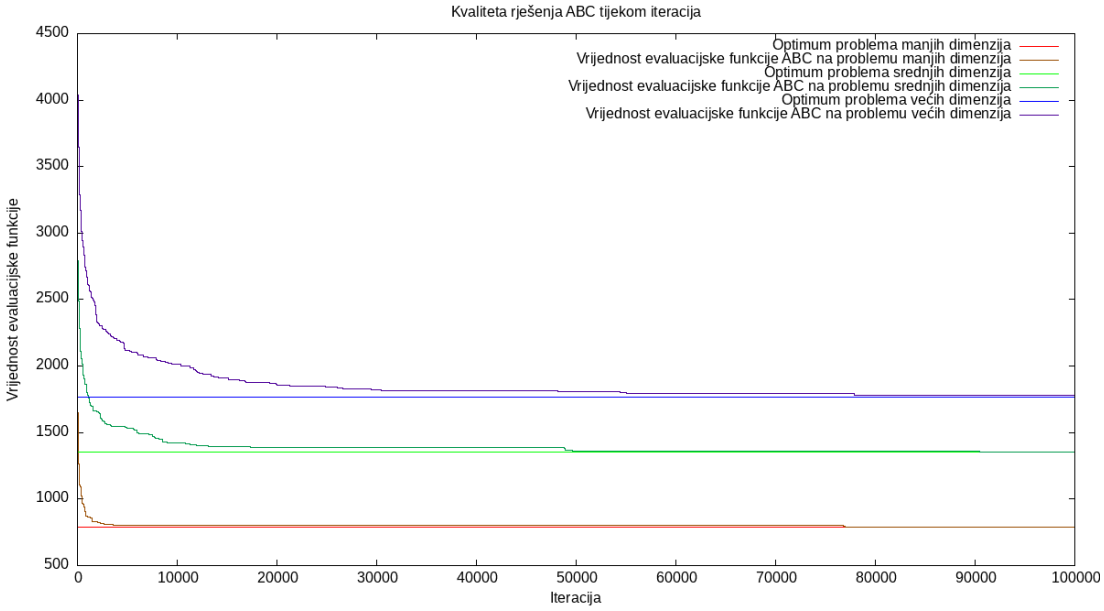
Tablica 5.28: Prosječna duljina izvođenja ABC (u ms) u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost

evaluacijske funkcije postiže minimum. Kod problema većih dimenzija, to je konfiguracija koja početnu populaciju popunjava *polu-pohlepnim* rješenjima.

U slučaju problema manjih i srednjih dimenzija, minimum se postiže za obje konfiguracije pa se kao optimalna bira ona konfiguracija za koju je postignuto minimalno prosječno vrijeme izvođenja. Kod problema manjih dimenzija, to je konfiguracija koja početnu populaciju popunjava *nasumičnim valjanim* rješenjima. S druge strane, kod problema srednjih dimenzija, to je konfiguracija koja početnu populaciju popunjava *polu-pohlepnim* rješenjima.

Napredak rješenja kroz iteracije uz navedene optimalne konfiguracije za svaki od triju problema vidljiv je na grafu 5.9.



Graf 5.9: Kvaliteta rješenja ABC kroz iteracije

Konačno, najbolja rješenja koja je ABC pronašao na definiranim problemima su:

$$X_{ABC}^{n32} = 787.082 \tag{5.16}$$

$$X_{ABC}^{n60} = 1355.799 \tag{5.17}$$

$$X_{ABC}^{n80} = 1778.320 \tag{5.18}$$

5.7. Tabu pretraživanje

TS korišteno u svim testovima odgovara algoritmu opisanom u odlomku 3.7. Maksimalni broj iteracija fiksiran je na $m = 1000$. Za svaki od triju problema ispituje se

utjecaj veličine tabu liste na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja. Konkretne veličine tabu liste za koje su provedeni testovi na sva tri problema su: 1, 2, 5, 10, 20 i 50.

Najbolja rješenja postignuta kroz deset testova za svaku od veličina tabu liste na sva tri problema navedena su u tablici 5.29, dok su prosječna vremena izvođenja navedena u tablici 5.30.

Instanca	Optimum	Veličina tabu liste					
		1	2	5	10	20	50
A-n32-k5	787.082	795.199	787.082	797.451	787.082	787.082	787.082
A-n60-k9	1355.799	1417.746	1419.485	1378.975	1402.098	1390.180	1385.862
A-n80-k10	1766.5	1825.425	1820.732	1869.268	1860.262	1826.859	1864.675

Tablica 5.29: Vrijednosti najboljih rješenja TS u ovisnosti o veličini tabu liste

Instanca	Veličina tabu liste					
	1	2	5	10	20	50
A-n32-k5	9355.4	9251.6	9303.7	9265.8	9241.5	9282.8
A-n60-k9	27347.0	27393.9	27375.7	27326.9	27411.0	27356.1
A-n80-k10	52022.9	52003.9	52053.7	52043.7	51969.1	52009.1

Tablica 5.30: Prosječna duljina izvođenja TS (u ms) u ovisnosti o veličini tabu liste

Kao optimalna veličina tabu liste kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema srednjih dimenzija postiže minimum za veličinu tabu liste $T = 5$, a kod problema većih dimenzija za veličinu tabu liste $T = 2$.

Kod problema manjih dimenzija, minimum se postiže za više veličina tabu liste pa se kao optimalna među njima bira ona za koju je postignuto minimalno prosječno vrijeme izvođenja, a to je $T = 20$.

Nakon utvrđivanja optimalnih veličina tabu lista, za svaki od triju problema ispitan je utjecaj metode generiranja početnog rješenja na kvalitetu konačnog rješenja i prosječno vrijeme izvođenja, pri čemu su veličine tabu lista fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.31, dok su prosječna vremena izvođenja prikazana u tablici 5.32.

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Kod problema srednjih i većih dimenzija, to je konfiguracija koja kao početno rješenje postavlja *nasumično valjano* rješenje.

Instanca	Optimum	Početno rješenje	
		NV	PP
A-n32-k5	787.082	787.082	787.082
A-n60-k9	1355.799	1389.184	1420.289
A-n80-k10	1766.5	1860.586	1883.021

Tablica 5.31: Vrijednosti najboljih rješenja TS u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Instanca	Početno rješenje	
	NV	PP
A-n32-k5	9699.6	9964.1
A-n60-k9	27464.5	27398.0
A-n80-k10	51818.4	51806.6

Tablica 5.32: Prosječna duljina izvođenja TS (u ms) u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

U slučaju problema manjih dimenzija, minimum se postiže za obje konfiguracije pa se kao optimalna bira ona konfiguracija za koju je postignuto minimalno prosječno vrijeme izvođenja, a to je konfiguracija koja kao početno rješenje postavlja *nasumično valjano* rješenje.

Napredak rješenja kroz iteracije uz navedene optimalne konfiguracije za svaki od triju problema vidljiv je na grafu 5.10

Konačno, najbolja rješenja koja je TS pronašlo na definiranim problemima su:

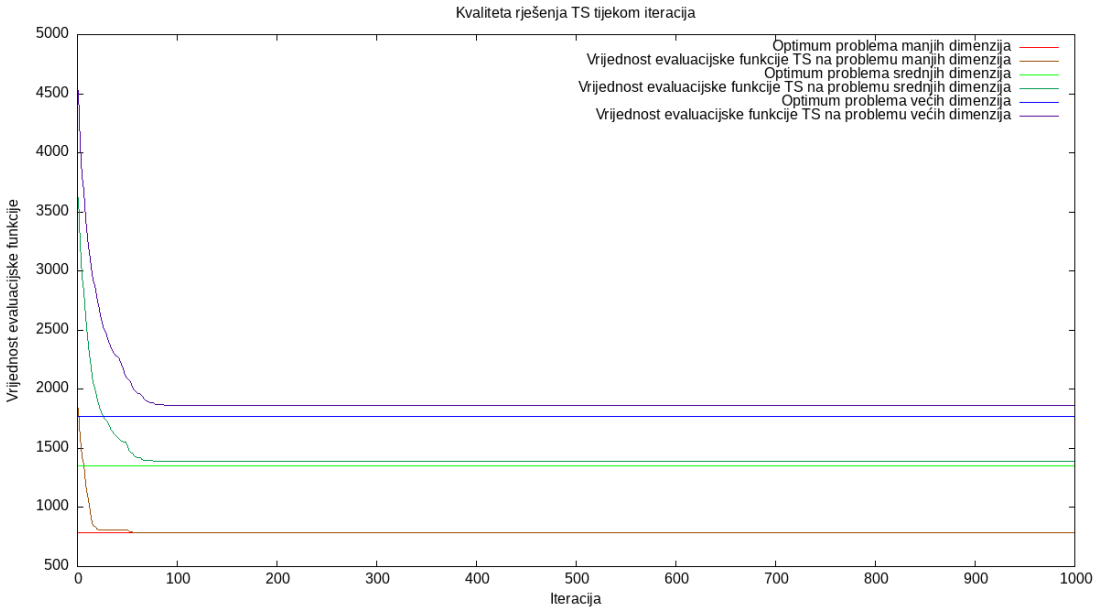
$$X_{TS}^{n32} = 787.082 \tag{5.19}$$

$$X_{TS}^{n60} = 1378.975 \tag{5.20}$$

$$X_{TS}^{n80} = 1820.732 \tag{5.21}$$

5.8. Simulirano kaljenje

SA korišteno u svim testovima odgovara algoritmu opisanom u odlomku 3.8. Broj paralelnih hlađenja fiksiran je na $n = 30$, početna temperatura na $T_s = 500$, a konačna temperatura na $T_f = 0.01$. Za svaki od triju problema ispituje se utjecaj funkcije dekrementa i odgovarajućeg parametra na kvalitetu rješenja, kao i na prosječno vrijeme izvođenja. Konkretne vrijednosti parametra B u slučaju linearnog dekrementa su: 0.005,



Graf 5.10: Kvaliteta rješenja TS kroz iteracije

0.01, 0.05 i 0.1; vrijednosti parametra α u slučaju geometrijskog dekrementa su: 0.99, 0.999, 0.9999 i 0.99995; a vrijednosti parametra β u slučaju vrlo sporog dekrementa su: 0.001, 0.005, 0.01 i 0.05.

Najbolja rješenja postignuta kroz deset testova za svaku od vrijednosti parametra svake od funkcija dekrementa na sva tri problema navedena su u tablicama 5.33, 5.35 i 5.37, dok su prosječna vremena izvođenja navedena u tablicama 5.34, 5.36 i 5.38.

Instanca	Optimum	B			
		0.005	0.01	0.05	0.1
A-n32-k5	787.082	787.082	787.082	791.079	798.377
A-n60-k9	1355.799	1367.357	1411.752	1563.973	1700.560
A-n80-k10	1766.5	1875.438	1944.877	2287.873	2511.054

Tablica 5.33: Vrijednosti najboljih rješenja SA u ovisnosti o vrijednosti parametra B uz linearni dekrement

Kao optimalna kombinacija funkcije dekrementa i vrijednosti odgovarajućeg parametra bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Evaluacijska funkcija kod problema srednjih dimenzija postiže minimum za *geometrijski* dekrement s vrijednošću parametra $\alpha = 0.9999$, a kod problema većih dimenzija za *geometrijski* dekrement s vrijednošću parametra $\alpha = 0.99995$.

Kod problema manjih dimenzija, minimum se postiže za više kombinacija funkcije

Instanca	B			
	0.005	0.01	0.05	0.1
A-n32-k5	19750.1	9894.3	1978.0	997.4
A-n60-k9	21208.4	10368.9	2105.6	1071.1
A-n80-k10	21614.9	10868.1	2243.5	1163.7

Tablica 5.34: Prosječna duljina izvođenja SA (u ms) u ovisnosti o vrijednosti parametra B uz linearni dekrement

Instanca	Optimum	α			
		0.99	0.999	0.9999	0.99995
A-n32-k5	787.082	848.980	787.082	787.082	787.082
A-n60-k9	1355.799	1997.173	1472.283	1365.202	1379.586
A-n80-k10	1766.5	2953.029	2146.190	1872.385	1815.205

Tablica 5.35: Vrijednosti najboljih rješenja SA u ovisnosti o vrijednosti parametra α uz geometrijski dekrement

Instanca	α			
	0.99	0.999	0.9999	0.99995
A-n32-k5	207.8	2070.6	21411.5	42621.4
A-n60-k9	243.9	2288.5	22417.3	44719.3
A-n80-k10	294.7	2438.1	23444.4	46794.3

Tablica 5.36: Prosječna duljina izvođenja SA (u ms) u ovisnosti o vrijednosti parametra α uz geometrijski dekrement

Instanca	Optimum	β			
		0.001	0.005	0.01	0.05
A-n32-k5	787.082	787.082	787.082	787.082	853.302
A-n60-k9	1355.799	1381.630	1407.318	1487.901	1812.052
A-n80-k10	1766.5	1843.386	1988.423	2155.665	2597.973

Tablica 5.37: Vrijednosti najboljih rješenja SA u ovisnosti o vrijednosti parametra β uz vrlo spor dekrement

dekrementa i vrijednosti odgovarajućeg parametra pa se kao optimalna među njima bira ona za koju je postignuto minimalno prosječno vrijeme izvođenja, a to je *vrlo spor* dekrement s vrijednošću parametra $\beta = 0.01$.

Instanca	β			
	0.001	0.005	0.01	0.05
A-n32-k5	19683.6	3954.3	1977.1	397.7
A-n60-k9	20594.0	4167.7	2106.4	440.0
A-n80-k10	23388.9	4395.3	2250.6	502.1

Tablica 5.38: Prosječna duljina izvođenja SA (u ms) u ovisnosti o vrijednosti parametra β uz vrlo spor dekrement

Nakon utvrđivanja optimalnih kombinacija funkcije dekrementa i vrijednosti odgovarajućeg parametra, za svaki od triju problema ispitan je utjecaj metode generiranja početnog rješenja na kvalitetu konačnog rješenja i prosječno vrijeme izvođenja, pri čemu su funkcije dekremenata i vrijednosti odgovarajućih parametara fiksirane na pronađene optimalne vrijednosti. Najbolja rješenja za navedene testove prikazana su u tablici 5.39, dok su prosječna vremena izvođenja prikazana u tablici 5.40.

Instanca	Optimum	Početno rješenje	
		NV	PP
A-n32-k5	787.082	787.082	787.082
A-n60-k9	1355.799	1380.278	1372.760
A-n80-k10	1766.5	1837.589	1832.575

Tablica 5.39: Vrijednosti najboljih rješenja SA u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Instanca	Početno rješenje	
	NV	PP
A-n32-k5	2025.7	2140.9
A-n60-k9	26477.1	25260.9
A-n80-k10	51181.3	51802.5

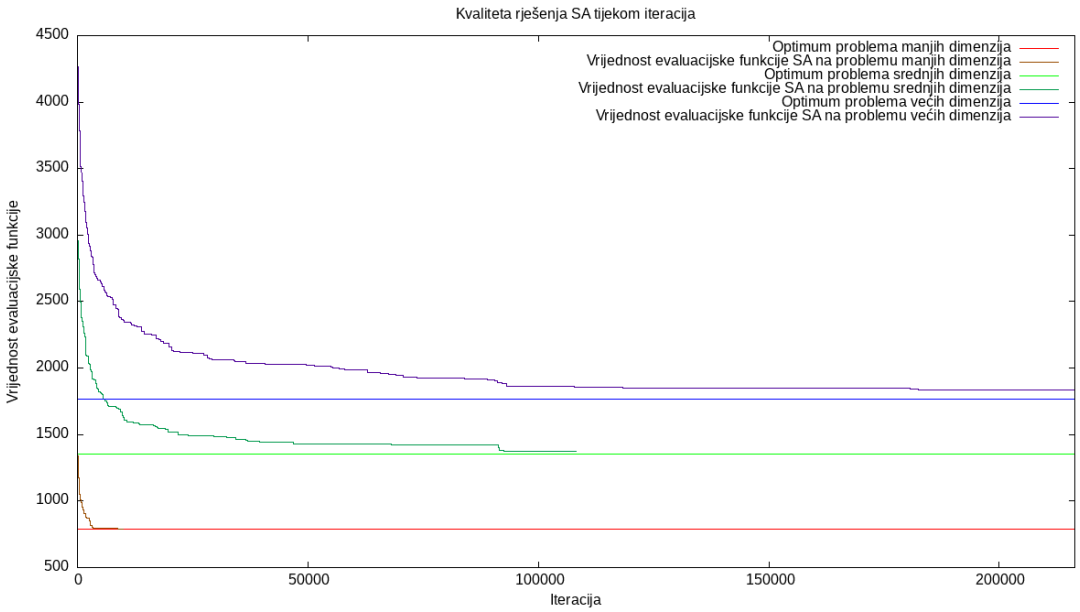
Tablica 5.40: Prosječna duljina izvođenja SA (u ms) u ovisnosti o lokalnom pretraživanju i metodi generiranja početne populacije

Kao optimalna konfiguracija kod svakog problema bira se ona za koju vrijednost evaluacijske funkcije postiže minimum. Kod problema srednjih i većih dimenzija, to je konfiguracija koja kao početno rješenje postavlja *polu-pohlepno* rješenje.

U slučaju problema manjih dimenzija, minimum se postiže za obje konfiguracije

pa se kao optimalna bira ona konfiguracija za koju je postignuto minimalno prosječno vrijeme izvođenja, a to je konfiguracija koja kao početno rješenje postavlja *nasumično valjano* rješenje.

Napredak rješenja kroz iteracije uz navedene optimalne konfiguracije za svaki od triju problema vidljiv je na grafu 5.11.



Graf 5.11: Kvaliteta rješenja SA kroz iteracije

Konačno, najbolja rješenja koja je SA pronašlo na definiranim problemima su:

$$X_{SA}^{n32} = 787.082 \tag{5.22}$$

$$X_{SA}^{n60} = 1365.202 \tag{5.23}$$

$$X_{SA}^{n80} = 1815.205 \tag{5.24}$$

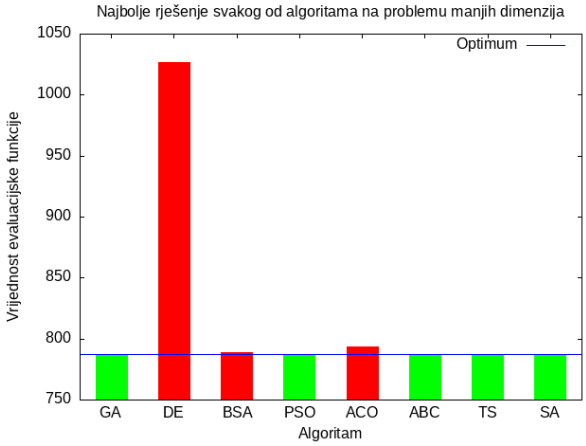
5.9. Zajednička analiza rezultata svih algoritama

Najbolja rješenja koja su svi algoritmi pronašli na svakom od problema navedena su u tablici 5.41. Najbolja rješenja za problem manjih dimenzija vizualizirana su na grafu 5.12, za problem srednjih dimenzija na grafu 5.13, a za problem većih dimenzija na grafu 5.14.

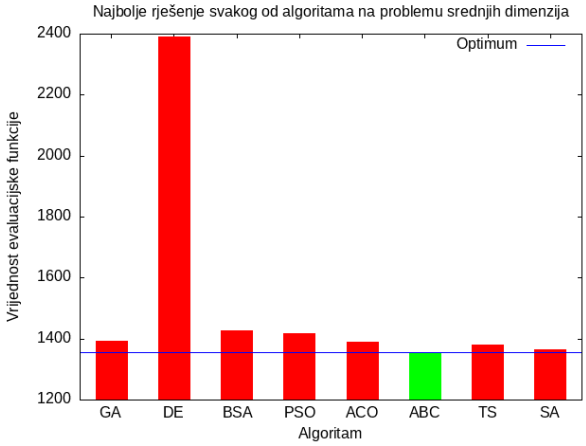
Na svakom od grafova je jedno ili više najboljih rješenja za konkretni problem prikazano zelenom bojom, dok su preostala (lošija) rješenja prikazana crvenom bojom. Optimalno rješenje za svaki od problema prikazano je plavom linijom.

Instanca	Optimum	Algoritam							
		GA	DE	BSA	PSO	ACO	ABC	TS	SA
A-n32-k5	787.082	787.082	1026.73	789.170	787.082	793.377	787.082	787.082	787.082
A-n60-k9	1355.79	1393.368	2391.500	1427.780	1417.630	1391.141	1355.799	1378.975	1365.202
A-n80-k10	1766.5	1838.116	4174.94	1893.750	1872.790	1870.925	1778.320	1820.732	1815.205

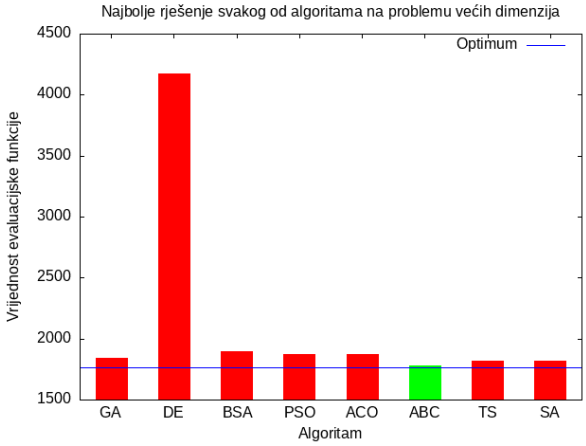
Tablica 5.41: Najbolje vrijednosti evaluacijske funkcije PSO na instanci A-n80-k10.vrp.



Graf 5.12: Kvaliteta rješenja po algoritmima na problemu manjih dimenzija



Graf 5.13: Kvaliteta rješenja po algoritmima na problemu srednjih dimenzija



Graf 5.14: Kvaliteta rješenja po algoritmima na problemu većih dimenzija

Radi jednostavnosti problema manjih dimenzija, pet algoritama je pronašlo njegovo optimalno rješenje — GA, PSO, ABC, TS i SA; dok je DE pronašao uvjerljivo najlošije rješenje. Na problemima srednjih i većih dimenzija ABC je pronašao najbolja rješenja (pri čemu je na problemu srednjih dimenzija pronašao optimum), dok je DE i dalje pronašao uvjerljivo najlošije rješenje.

Iz rezultata prikazanih u tablicama i grafovima vidljivo je da su algoritmi koji u svojem osnovnom obliku koriste lokalnu pretragu (ABC, TS i SA) pronašli rješenja koja su uvjerljivo bolja od rješenja ostalih algoritama. S druge strane, algoritmi koji koriste prikaz rješenja realnim brojevima (DE, BSA i PSO, a posebice DE) pronašli su lošija rješenja od ostalih algoritama, pogotovo u slučaju kada nije korišteno pretraživanje susjedstva.

Kao uvjerljivo najbolji algoritam pokazuje se ABC, za što je najvjerojatnije zaslužna učinkovitost lokalne pretrage na više fronti (rješenja pčela radnica) uz intenzivniju lokalnu pretragu u susjedstvu boljih rješenja. S druge strane, kao uvjerljivo najlošiji algoritam pokazuje se DE, za što je najvjerojatnije zaslužna nekompatibilnost algoritma s prikazom rješenja realnim brojevima koji koristi RK kodiranje.

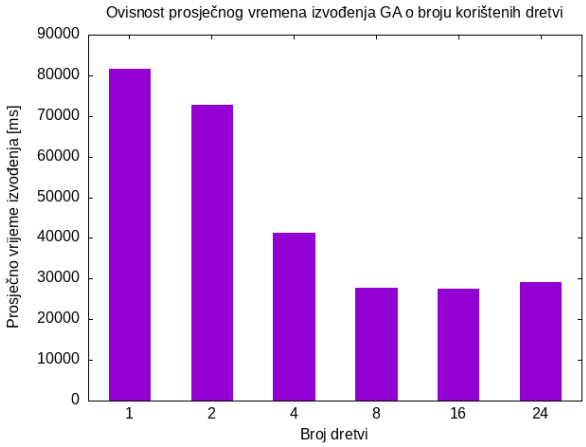
Korištenje lokalnog pretraživanja kod većine algoritama rezultira boljim rješenjima. Ova pojava može se primijetiti i u baznom obliku GA, budući da se mutacija u suštini može promatrati kao lokalno pretraživanje — veći iznos vjerojatnosti mutacije rezultira boljim rješenjima. S druge strane, metoda generiranja početnih rješenja nema velik utjecaj na konačno rješenje.

5.10. Utjecaj paralelizacije na vrijeme izvođenja algoritma

Kako su svi testovi provedeni na procesoru s osam fizičkih jezgri koji se zbog „grupirane višedretvenosti“ (engl. *clustered multithreading*, AMD-ov ekvivalent Intelovog „*hyperthreadinga*“) operacijskom sustavu predstavlja kao procesor sa šesnaest jezgri, u svim testovima iz prethodnih odlomaka je prilikom paralelizacije korišten bazen dretvi veličine 16. Međutim, provedeno je i ispitivanje utjecaja veličine bazena dretvi na prosječno vrijeme izvođenja algoritma.

Testiranje je provedeno za GA nad problemom manjih dimenzija, a konkretne veličine bazena dretvi za koje su provedeni testovi su: 1, 2, 4, 8, 16 i 24. Za svaku od ispitanih veličina bazena dretvi rezultati su dobiveni uprosječivanjem vremena izvođenja dobivenih iz deset pokretanja algoritma.

Rezultati testiranja vidljivi su na grafu 5.15 i navedeni u tablici 5.42 zbog male razlike između određenih rezultata.



Graf 5.15: Utjecaj veličine bazena dretvi na prosječno vrijeme izvođenja algoritma

	Broj dretvi					
	1	2	4	8	16	32
duljina izvođenja [ms]	81556.5	72743.3	41248.5	27641.2	27607.3	29193.9

Tablica 5.42: Prosječna duljina izvođenja GA u ovisnosti o veličini bazena dretvi

Kao što je i očekivano, vrijeme izvođenja algoritma smanjuje se povećanjem veličine bazena dretvi sve do određene mjere gdje kreće stagnirati, a daljnjim povećanjem veličine bazena dretvi i rasti. Ovisnost vremena izvođenja o broju dretvi nije linearna iz tri razloga:

- procesor ima fizičko ograničenje broja instrukcija koje može izvoditi paralelno;
- stvaranje i evaluacija novih jedinki provode se paralelno, dok se ostatak algoritma izvodi sekvencijski;
- povećanjem broja dretvi u paraleli povećava se vrijeme koje dretve gube na komunikaciju.

Prosječno vrijeme izvođenja algoritma je najkraće za bazen dretvi veličine 16, ali ono se tek minorno razlikuje od prosječnog vremena izvođenja koje postiže algoritam koji koristi bazen dretvi veličine 8 — „grupirana višedretvenost“ ne igra veliku ulogu kod paralelizacije algoritama. Daljnjim povećanjem broja dretvi algoritam se usporava jer se izvođenje algoritma ne ubrzava, a vrijeme koje se troši na komunikaciju raste.

6. Zaključak

Metaheuristike se pokazuju kao vrlo moćno sredstvo pri aproksimaciji rješenja optimizacijskih problema, a među njima i problema usmjeravanja vozila ograničenog kapaciteta. Iako rješenja koja metaheuristike daju pri rješavanju problema CVRP najčešće nisu optimalna, u velikom broju slučajeva su dovoljno dobra da bi se mogla primijeniti za usmjeravanje vozila u stvarnom životu.

U okviru ovog rada ostvaren je programski sustav koji služi za rješavanje problema CVRP uz pomoć metaheuristika. Konkretan problem usmjeravanja vozila ograničenog kapaciteta može se zadati proizvoljno, pod uvjetom da je zadan na način specificiran u odlomku 4.2. Kao metoda rješavanja može se odabrati bilo koji od osam opisanih metaheurističkih algoritama uz mogućnost regulacije parametara, metode generiranje početne populacije te korištenja lokalne pretrage.

Bitno je primijetiti da postoje metaheuristike koje su pogodne, ali isto tako i metaheuristike koje nisu pogodne za rješavanje kako problema CVRP, tako ni ostalih optimizacijskih problema, što se može vidjeti iz eksperimentalnih rezultata ovog rada. Metaheuristike koje u svom osnovnom obliku koriste lokalno pretraživanje dominiraju nad ostalima, dok su metaheuristike u čijoj je implementaciji korišten prikaz rješenja realnim brojevima vidljivo lošije od ostalih.

Nedeterministički algoritmi izvode se neusporedivo brže od deterministčkih. Međutim, njihova brzina izvođenja može se dodatno povećati paralelizacijom određenih dijelova algoritma, sve do mjere koju dozvoljava računalno sklopovlje.

LITERATURA

[1] D. Abrahams i suradnici. Radni okvir Boost. URL https://www.boost.org/. [Online; pristupljeno 6.11.2021.].

[2] James Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal of Computing*, 6, 02 2006.

[3] Tonci Caric i Hrvoje Gold. *Vehicle Routing Problem*. IntechOpen, Rijeka, 2008. doi: 10.5772/64. URL https://doi.org/10.5772/64.

[4] Patryk Filipiak i Piotr Lipinski. Parallel chc algorithm for solving dynamic traveling salesman problem using many-core gpu. 09 2012. ISBN 978-3-642-33184-8. doi: 10.1007/978-3-642-33185-5_34.

[5] Ahmed Fouad. Backtracking search optimization algorithm (bsa), 04 2015.

[6] F. Glover i K. Sörensen. Metaheuristics. *Scholarpedia*, 10(4):6532, 2015. doi: 10.4249/scholarpedia.6532. revision #149834.

[7] Marin Golub. Predavanja iz kolegija neizrazito, evolucijsko i neuroračunarstvo, 2020./2021.

[8] D. Karaboga. Artificial bee colony algorithm. *Scholarpedia*, 5(3):6915, 2010. doi: 10.4249/scholarpedia.6915. revision #91003.

[9] Korisnici Wikipedije. Travelling salesman problem — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1086332106, 2022. [Online; pristupljeno 19.5.2022.].

[10] Korisnici Wikipedije. Vehicle routing problem — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Vehicle_routing_problem&oldid=1086710722, 2022. [Online; pristupljeno 5.6.2022.].

[11] Sandeep Kumar, Rajani Poonia, i Vivek Sharma. Improved onlooker bee phase in artificial bee colony algorithm. *International Journal of Computer Applications*, 90:31–39, 03 2014. doi: 10.5120/15579-4304.

[12] Vjeko Kužina. Rješavanje višekriterijskog problema usmjeravanja vozila evolucijskim algoritmima, 2020.

[13] Wang Lijin, Yilong Yin, Wenting Zhao, Binqing Wang, i Yulong Xu. A hybrid backtracking search optimization algorithm with differential evolution. *Mathematical Problems in Engineering*, 2015:1–16, 04 2015. doi: 10.1155/2015/769245.

[14] I. Lima i ostali. Capacitated vehicle routing problem library. URL `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`. [Online; pristupljeno 6.11.2021.].

[15] Lea Skorin-Kapov i Nina Skorin-Kapov. Prezentacije u sklopu predavanja iz predmeta heurističke metode optimizacija, 2021.

[16] Rainer Storn i Kenneth Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 01 1997. doi: 10.1023/A:1008202821328.

[17] W.F. Tan, Lai Soon Lee, Zanariah Abdul Majid, i Hsin-Vonn Seow. Ant colony optimization for capacitated vehicle routing problem. *Journal of Computer Science*, 8:846–852, 01 2012.

[18] Shuzhu Zhang i C. Lee. An improved artificial bee colony algorithm for the capacitated vehicle routing problem. stranice 2124–2128, 10 2015. doi: 10.1109/SMC.2015.371.

Rješavanje problema usmjeravanja vozila ograničenog kapaciteta uz pomoć metaheuristika

Sažetak

U ovom je radu opisan problem usmjeravanja vozila ograničenog kapaciteta, kao i nekoliko metaheuristika koje se koriste u njegovom rješavanju. Prikazan je i način korištenja programskog rješenja ostvarenog u svrhu rješavanja problema usmjeravanja vozila ograničenog kapaciteta uz pomoć metaheuristika. Grafički i tablično prikazana je statistika dobivenih rješenja, vremena izvođenja algoritama, utjecaj lokalne pretrage i metode generiranja početnih rješenja na kvalitetu konačnih rješenja te utjecaj veličine bazena dretvi na vrijeme izvođenja algoritma.

Ključne riječi: metaheuristike, problem usmjeravanja vozila ograničenog kapaciteta, paralelizacija, bazen dretvi, evolucijsko računarstvo

Solving the capacitated vehicle routing problem using metaheuristics

Abstract

This thesis describes the capacitated vehicle routing problem, as well as few metaheuristics used to solve it. The way of using the software solution developed for solving the capacitated vehicle routing problem is also shown. Results and execution times statistics, the influence of local search and initial population generation method on the solution quality, as well as the influence of thread pool size on the algorithm execution time are shown in the graphs and tables.

Keywords: metaheuristics, capacitated vehicle routing problem, parallelization, thread pool, evolutionary computation