UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO

Štefan Brest

REŠEVANJE PROBLEMA NESIMETRIČNEGA TRGOVSKEGA POTNIKA Z DIFERENCIALNO EVOLUCIJO IN HEVRISTIČNIMI ALGORITMI

Diplomsko delo

Maribor, januar 2009

UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO
2000 Maribor, Smetanova ul. 17

Diplomsko delo univerzitetnega študijskega programa

REŠEVANJE PROBLEMA NESIMETRIČNEGA TRGOVSKEGA POTNIKA Z DIFERENCIALNO EVOLUCIJO IN HEVRISTIČNIMI ALGORITMI

Študent: Štefan Brest
Študijski program: univerzitetni, Računalništvo in informatika
Smer: Programska oprema

Mentor: red. prof. dr. Viljem ŽUMER

Maribor, januar 2009

UNIVERZA V MARIBORU

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, 2000 Maribor

Številka: RI-469
Datum in kraj: 30. 10. 2008, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 32/2008)

SKLEP O DIPLOMSKEM DELU

1. **Štefanu Brestu,** študentu univerzitetnega študijskega programa Računalništvo in informatika, smer Programska oprema, se dovoljuje izdelati diplomsko delo pri predmetu Principi programskih jezikov.

2. **MENTOR:** **red. prof. dr. Viljem Žumer**

3. **Naslov diplomskega dela:**
   REŠEVANJE PROBLEMA NESIMETRIČNEGA TRGOVSKEGA POTNIKA Z DIFERENCIALNO EVOLUCIJO IN HEVRISTIČNIMI ALGORITMI

4. **Naslov diplomskega dela v angleškem jeziku:**
   SOLVING ASYMMETRIC TRAVELING SALESMAN PROBLEM WITH DIFFERENTIAL EVOLUTION AND HEURISTIC ALGORITHMS

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih ter en izvod elektronske verzije do 30. 10. 2009 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO RAČUNALNIŠTVO IN INFORMATIKO 8

Obvestiti:
- kandidata,
- mentorja,
- somentorja,
- odložiti arhiv.

Zahvala

Zahvaljujem se mentorju red. prof. dr. Viljemu Žumerju za pomoč in vodenje pri snovanju diplomskega dela.

Posebej se zahvaljujem izr. prof. dr. Janezu Brestu za vso pomoč, podporo in naklonjenost ter nasvete skozi celotni študijski čas, kakor tudi pri pripravi diplomskega dela.

Zahvalil bi se rad tudi družini, dekletu in prijateljem, ki so me podpirali in vzpodbujali ter mi stali ob strani na študijski poti.

Hvala Vam!

“Beatus, qui prodest, quibus potest“
(Srečen je tisti, ki lahko pomaga vsem, ki jim lahko pomaga)

REŠEVANJE PROBLEMA NESIMETRIČNEGA TRGOVSKEGA POTNIKA Z DIFERENCIALNO EVOLUCIJO IN HEVRISTIČNIMI ALGORITMI

Ključne besede: problem trgovskega potnika, diferencialna evolucija, hevristični algoritmi, optimizacija, iskanje globalnega optimuma

UDK: 004.43.021:004.89(043.2)

Povzetek

V diplomskem delu podamo zgodovino in razvoj evolucijskih algoritmov ter diferencialne evolucije. Opišemo tudi hevristične algoritme, ki bodo osnova za reševanje problema nesimetričnega trgovskega potnika.

Osrednji del diplomskega dela predstavlja načrtovanje in implementacija algoritmov DEATSP, kjer poiskušamo rešiti problem nesimetričnega trgovskega potnika z diferencialno evolucijo. Ti vključujejo različne kombinacije hevrističnih algoritmov, kjer skušamo ugotoviti, katera od teh daje najboljšo rešitev zastavljenega problema trgovskega potnika.

Sledijo še rezultati, ki smo jih dobili z algoritmi DEATSP na standardni knjižnici TSPLIB za probleme trgovskega potnika. Delo zaključimo s sklepom in ugotovitvami, kako reševati omenjen problem z diferencialno evolucijo in hevrističnimi algoritmi.

SOLVING ASYMMETRIC TRAVELING SALESMAN PROBLEM WITH DIFFERENTIAL EVOLUTION AND HEURISTIC ALGORITHMS

Keywords: traveling salesman problem, differential evolution, heuristic algorithms, optimization, searching global optimum

UDK: 004.43.021:004.89(043.2)

Abstract

The presented diploma work begins with the history and development of evolutionary algorithms and differential evolution. Heuristic algorithms are described as basis for the asymmetric traveling salesman problem solving.

The main part of the diploma work is designing and implementation of DEATSP algorithms, where we try to solve asymmetric salesman problem with differential evolution. These involve various combinations of heuristic algorithms and our aim is to search for the one that gives the best solution to the given traveling salesman problem.

Finally, the results acchieved with DEATSP algorithms on standard TSPLIB library data for the salesman problem are presented. The work is concluded with the summary of the best combination of differential evolution and heuristic algorithms for the given problem.

Kazalo

1 Uvod **1**

2 Sorodna dela **3**
2.1 Evolucijski algoritmi 3
2.2 Hevristični algoritmi 4

3 Diferencialna evolucija **6**
3.1 Uvod 6
3.2 Algoritem diferencialne evolucije 7
3.2.1 Mutacija 8
3.2.2 Križanje 8
3.2.3 Selekcija 8
3.2.4 Strategije 9

4 Problem trgovskega potnika **11**
4.1 Zgodovinski pregled razvoja problema trgovskega potnika 11
4.2 Opis problema trgovskega potnika 13
4.2.1 Simetrični problem trgovskega potnika 14
4.2.2 Nesimetrični problem trgovskega potnika 15
4.3 Algoritem RAI 16
4.3.1 Lokalna optimizacija 16
4.3.2 Opis algoritma *RAI* 17
4.3.3 Implementacija 18
4.3.4 Rezultati 19
4.4 Hevristični algoritem *RAI+OR* 21
4.4.1 Rezultati 23

5 Algoritmi DEATSP **25**
5.1 Predstavitev posameznikov in ocenitvena funkcija 25
5.1.1 Predstavitev posameznikov v *DE* 25

5.1.2 Krmilni parametri 28
5.1.3 Ocenitvena funkcija 28
5.1.4 Funkcija za povečanje disperzije 29
5.1.5 Funkcija za preverbo in popravo celovitosti cikla 30
5.2 Predstavitev strategij za algoritme *DEATSP* 31
5.2.1 Strategija *DE/rand/1/exp* za trgovskega potnika 31
5.2.2 Strategija *DE/rand/1/bin* za trgovskega potnika 32
5.2.3 Strategija *Xing* 33
5.3 Algoritmi *DEATSP* 37
5.3.1 Osnovni algoritem *DE* za trgovskega potnika 37
5.3.2 Algoritem *DE+RAI* 38
5.3.3 Algoritem *DE+OR-opt* 39
5.3.4 Algoritem *DE+RAI+OR-opt mod 3* 40
5.3.5 Algoritem *DE+RAI+OR-opt-V2* 41
5.3.6 Algoritem *DE+RAI+OR-opt-V3* 42

6 Rezultati **43**
6.1 Algoritem *DE+RAI* 45
6.1.1 Algoritem *DE+RAI* s strategijo *DE/rand/1/exp* 45
6.1.2 Algoritem *DE+RAI* s strategijo *DE/rand/1/bin* 46
6.1.3 Algoritem *DE+RAI* s strategijo *Xing* 48
6.2 Algoritem *DE+OR-opt* 50
6.2.1 Algoritem *DE+OR-opt* s strategijo *DE/rand/1/exp* 50
6.2.2 Algoritem *DE+OR-opt* s strategijo *DE/rand/1/bin* 50
6.2.3 Algoritem *DE+OR-opt* s strategijo *Xing* 52
6.3 Algoritem *DE+RAI+OR-opt mod 3* 55
6.3.1 Algoritem *DE+RAI+OR-opt mod 3* s strat. *DE/rand/1/exp* . 55
6.3.2 Algoritem *DE+RAI+OR-opt mod 3* s strat. *DE/rand/1/bin* . 56
6.3.3 Algoritem *DE+RAI+OR-opt mod 3* s strategijo *Xing* 58
6.4 Algoritem *DE+RAI+OR-opt-V2* 60
6.4.1 Algoritem *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/exp* . 60
6.4.2 Algoritem *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/bin* . 61
6.4.3 Algoritem *DE+RAI+OR-opt-V2* s strategijo *Xing* 62
6.5 Algoritem *DE+RAI+OR-opt-V3* 65
6.5.1 Algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/exp* . 65
6.5.2 Algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/bin* . 66
6.5.3 Algoritem *DE+RAI+OR-opt-V3* s strategijo *Xing* 68

6.5.4 Primerajva najboljših rešitev za algoritem *DE+RAI+OR-opt-V3* 69
6.6 Primerjava rezultatov algoritmov *DEATSP* 71
6.7 Praktičen primer iskanja obhoda slovenskih mest 73

7 Zaključek **74**

Literatura **75**

Življenjepis **77**

Slike

4.1 Grafična ponazoritev naraščanja števila mest v optimalnih rešitvah od začetkov raziskovanja problema trgovskega potnika do danes . . . 12
4.2 Primer grafa za problem simetričnega trgovskega potnika 15
4.3 Rešitev najkrajše poti TSP za primer iz slike 4.2 15
4.4 Primer grafa za problem nesimetričnega trgovskega potnika 15
4.5 Rešitev najkrajše poti ATSP za primer iz slike 4.4 16
4.6 Rezultat v odvisnosti od časa za graf *rbg443* 20
4.7 Povprečje rešitev v 100 poskusih za grafe od 10 do 50 vozlišč. 23

5.1 Obhodna pot (cikel) med osmimi slovenskimi mesti 26
5.2 Ilustracija križanja po metodi Wand in Wu 33
5.3 Ilustracija operatorja mutacije 36

6.1 Reševanje grafa *kro124p* z algoritmom *DE+RAI* 49
6.2 Reševanje grafa *ftv120* z algoritmom *DE+OR-opt* 54
6.3 Reševanje grafa *ftv70* z algoritmom *DE+RAI+OR-opt mod 3* 59
6.4 Reševanje grafa *ry48p* z algoritmom *DE+RAI+OR-opt-V2* 64
6.5 Reševanje grafa *ft70* z algoritmom *DE+RAI+OR-opt-V3* 69
6.6 Prikaz uspešnosti algoritmov *DEATSP* 72
6.7 Obhodna zračna pot med osmimi slovenskimi mesti 73

Tabele

4.1 Rezultati na nesimetričnih problemih trgovskega potnika 19

5.1 Matrika zračnih razdalj med osmimi slovenskimi mesti 26

6.1 Rezultati algoritma *DE+RAI* s strategijo *DE/rand/1/exp* 46
6.2 Rezultati algoritma *DE+RAI* s strategijo *DE/rand/1/bin* 47
6.3 Rezultati algoritma *DE+RAI* s strategijo *Xing* 48
6.4 Rezultati algoritma *DE+OR-opt* s strategijo *DE/rand/1/exp* 51
6.5 Rezultati algoritma *DE+OR-opt* s strategijo *DE/rand/1/bin* 52
6.6 Rezultati algoritma *DE+OR-opt* s strategijo *Xing* 53
6.7 Rezultati alg. *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp* . 56
6.8 Rezultati alg. *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/bin* . 57
6.9 Rezultati alg. *DE+RAI+OR-opt mod 3* s strategijo *Xing* 58
6.10 Rezultati alg. *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/exp* . . . 61
6.11 Rezultati alg. *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/bin* . . . 62
6.12 Rezultati alg. *DE+RAI+OR-opt-V2* s strategijo *Xing* 63
6.13 Povprečni rezultati 25-ih poskusov za algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/exp* 66
6.14 Povprečni rezultati 25-ih poskusov za algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/bin* 67
6.15 Povprečni rezultati 25-ih poskusov za algoritem *DE+RAI+OR-opt-V3* s strategijo *Xing* 68
6.16 Primerjava najboljših in povprečnih rezultatov za algoritem *DE+RAI+OR-opt-V3* 70
6.17 Primerjava rešitev za algoritme *DEATSP* 71

Seznam algoritmov

1 Evolucijski algoritem 4
2 Algoritem *RAI* 16
3 Funkcija *Tour2Position* 27
4 Funkcija *Position2Tour* 27
5 Ocenitvena funkcija za trgovskega potnika 28
6 Funkcija *check_same* za povečanje disperzije posameznikov 29
7 Funkcija *popravi_tmp*, ki preveri (popravi) celovitost poti (cikla) . . . 30
8 Strategija *DE/rand/1/exp* za trgovskega potnika 31
9 Strategija *DE/rand/1/bin* za trgovskega potnika 32
10 Funkcija križanja za trgovskega potnika 34
11 Funkcija mutacije za trgovskega potnika 35
12 Algoritem *DE+RAI* 38
13 Algoritem *DE+OR-opt* 39
14 Algoritem *DE+RAI+OR-opt mod 3* 40
15 Algoritem *DE+RAI+OR-opt-V2* 41
16 Algoritem *DE+RAI+OR-opt-V3* 42

Poglavje 1

Uvod

Diferencialna evolucija (DE - ang. *Differential Evolution*) je algoritem za numerično optimizacijo predvsem zveznih funkcij. Uporablja se v številnih praktičnih aplikacijah, predvsem zaradi njegove dobre konvergence. Prištevamo ga k evolucijskim algoritmom, ki za reševanje problema potrebujejo le funkcijo, katero želijo optimirati.

V diplomskem delu opisujemo reševanje naloge nesimetričnega trgovskega potnika z diferencialno evolucijo in hevrističnimi algoritmi. Trgovski potnik je eden najbolj obravnavanih primerov iz teorije grafov, za katerega je dokazana NP-polnost [18]. Skozi diplomsko delo bomo skovali nekaj različic algoritmov za iskanje (optimalne) rešitve problema trgovskega potnika. Vedeti je potrebno, da če velja P≠NP[1], potem za NP-težke probleme ne obstajajo polinomski algoritmi, kar nas privede k raziskovanju algoritmov na osnovi kombinacij starih in novih idej.

Za razumevanje delovanja algoritmov pri reševanju problema nesimetričnega trgovskega potnika, ki so predstavljeni v diplomskem delu, je potrebno poznati osnove diferencialne evolucije in osnove hevrističnih algoritmov *RAI* in *OPT*. Oba algoritma bomo natančneje spoznali do tolikšne mere, kot je potrebno za razvoj novih kombinacij algoritmov, ki bodo temeljili na diferencialni evoluciji. Tako bomo s kombinacijo hevrističnih algoritmov in diferencialne evolucije reševali nesimetrični problem trgovskega potnika na način, da se bomo optimalni rešitvi čimbolj približali, oziroma jo bomo morda celo našli.

Diplomsko delo obsega 7 poglavij. V uvodu je predstavljena vsebina diplomskega dela. Drugo poglavje predstavlja sorodna dela. Čeprav je sorodnih del za reševanje

[1]P≠NP je verjetno najbolj znan odprti problem teoretičnega računalništva (glej [14]). Prevladuje mnenje, da velja P≠NP.

nesimetričnega problema trgovskega potnika z diferencialno evolucijo malo, oziroma skoraj nič, smo se dotaknili osnovnih povezav evolucijskih in hevrističnih algoritmov. V tretjem poglavju opisujemo osnovne izraze diferencialne evolucije, njeno delovanje in algoritem, ki ga povezujejo operatorji mutacije, selekcije in križanja. Četrto poglavje obravnava problem trgovskega potnika. V tem poglavju je opisana zgodovina trgovskega potnika in opis problema, ki pa se v grobem deli na dva dela: simetrični in nesimetrični problem trgovskega potnika. Prav tako v istem poglavju opisujemo problem hevrističnega algoritma *RAI* ter algoritem *RAI+OR*, katerega soavtorstvo delim tudi sam.

Jedro diplomskega dela predstavlja peto poglavje. V tem poglavju najprej opišemo osnove naših algoritmov, ki smo jih poimenovali algoritmi *DEATSP*, strategije reševanja z diferencialno evolucijo ter kombinacije algoritmov za reševanja problemov nesimetričnega trgovskega potnika. Z rezultati pokažemo, da kombinacije algoritmov, ki smo jih uvedli, dosegajo dobre rezultate za reševanje nesimetričnega problema trgovskega potnika.

Diplomsko delo zaključimo s povzetkom obdelane tematike, z idejami za nadaljnje raziskovanje in predstavitvijo odprtih problemov in idej, ki bi nas vodile k še boljšemu reševanju zadanega problema.

Poglavje 2

Sorodna dela

2.1 Evolucijski algoritmi

Evolucijski algoritem (EA - ang. *Evolutionary Algorithm*) je skupni izraz za postopke reševanja problemov s pomočjo računalnikov, ki uporabljajo model in mehanizme biološke evolucije. Za vse te algoritme je značilno, da simulirajo evolucijo in njene mehanizme kot so selekcija, križanje in mutacija.

K evolucijskim algoritmom prištevamo (opis smo povzeli po delu [36]):

- **evolucijske strategije** (ang. *Evolutionary Strategies* - ES), ki poleg iskalnih parametrov v posamezni vektor kodirajo tudi krmilne parametre iskalnega algoritma,
- **genetske algoritme** (ang. *Genetic Algorithm – GA*) [15] rešitve kreira tako, da realno rešitev binarno kodira, nato pa z operatorji mutacije (ang. *mutation*) in križanja (*ang. crossover*) spreminja posamezne bite v binarni predstavitvi. Operator mutacije navadno deluje uniformno naključno nad vsakim bitom binarno kodiranega iskalnega parametra v posamezni rešitvi. Operator križanja pa združi posamezne iskalne parametre iz več posameznikov (staršev) v novega posameznika, tako da jemlje nekaj bitov iz vsakega od staršev,
- **genetsko programiranje** (ang. *Genetic Programming* [17]) gradi drevesno strukturo, program; uporabljeni genetski operatorji so posebej prirejeni za operacije nad drevesom,
- **evolucijsko programiranje** (ang. *Evolutionary Programming – EP*) je bilo predstavljeno kot pristop v umetni inteligenci [13], enega od sodobnejših algoritmov pa predstavlja delo [35], ki daje tudi primerjalno osnovo za številne

medsebojne primerjave ostalih algoritmov nad standardnim izbranim naborom testnih funkcij za enokriterijsko optimizacijo,

- **optimizacija z rojem "delcev"** (ang. *Particle Swarm Optimization – PSO*) poleg vsakega vektorja iskalnih parametrov (delca) hrani še njegov diferenčni vektor, ki modelira hitrost premikanja delca v geografskem (iskalnem) prostoru,
- **optimizacija s kolonijo mravelj** (ang. *Ant Colony Optimization – ACO*) modelira pomnenje zgodovine dobrih iskalnih parametrov po vzoru mravelj, ki v primeru najdene hrane za seboj puščajo feromonske sledi [16], in
- **diferencialna evolucija** (ang. *Differential Evolution* - DE), ki je opisana v naslednjem poglavju.

Algoritem 1 Evolucijski algoritem

```
ustvari populacijo P(t);
ovrednoti P(t);
repeat
  P'(t) ⇐ križanje P(t);
  P"(t) ⇐ mutacija P'(t);
  ovrednoti P"(t);
  P(t + 1) ⇐ selekcija P"(t);
  t ⇐ t + 1;
until ustavitveni pogoj(t);
```

Psevdokod evolucijskega algoritma (Algoritem 1) je preprost. Ta osnovni evolucijski algoritem se v primeru evolucijskih strategij, genetskih algoritmov in evolucijskega programiranja le malenkostno razlikuje. Razlike nastanejo v predstavitvi posameznikov, vrsti selekcije, izbiri operatorjev in v vrstnem redu njihovega izvajanja.

2.2 Hevristični algoritmi

Metode, ki se jih loteva večina znanstvenikov, imenujemo *hevristike*. To so metode, ki ne zagotavljajo optimalne rešitve, vendar podajajo "relativno dobro" rešitev v sprejemljivem času. Pri problemu trgovskega potnika takšne metode delimo na hevristike, ki poskušajo "dobro" rešitev skonstruirati, medtem ko hevristike druge skupine poskušajo izboljšati neko dano rešitev z lokalnimi izboljšavami.

Raziskovalci se pri problemu trgovskega potnika ne ubadajo zgolj z iskanjem optimalne poti, temveč tudi s povezanimi problemi:

- kako preveriti optimalnost neke poti in
- kaj so spodnje meje optimalne dolžine poti?

Hevristični algoritem rešuje težki optimizacijski problem in se ne ozira na to, ali je trenutna rešitev optimalna oziroma ni.

Računalniška znanost teži k temu, da bi našla takšne algoritme, ki bi v zelo kratkem času znali poiskati dobro oziroma najboljšo rešitev. Hevristični algoritem lahko hitro opusti enega od teh dveh ciljev, na primer:

1. ponavadi najde precej dobre rešitve, vendar ni dokazov, da ne bi mogel priti samodejno do slabega rezultata,
2. običajno traja iskanje rešitve hitro, vendar ni nobenega zagotovila, da se bo to vedno zgodilo.

Številni komercialni proti–virusni programi uporabljajo prav hevristične metode za iskanje specifičnih lastnosti in značilnosti za odkrivanje virusov in drugih oblik škodljivih programskih paketov.

Pogosto je mogoče najti posebej oblikovane primere problemov, kjer hevristični algoritem najde zelo slabe rezultate ali pa je čas iskanja zelo dolg. Zato je pomembno, za kakšne probleme se uporablja hevristika v realnem svetu. Za mnogo praktičnih težav je hevristični algoritem prav tisti edini način, da se najdejo dobre rešitve v zadovoljivem času.

Poglavje 3

Diferencialna evolucija

3.1 Uvod

Algoritem diferencialne evolucije (DE) (ang. *Differential Evolution*) [28, 29, 23, 12] je algoritem za numerično optimizacijo predvsem zveznih funkcij. V zadnjih letih je postal zelo popularen in ga uporabljamo v številnih praktičnih aplikacijah, predvsem zato, ker algoritem odlikuje dobra konvergenca [28, 29, 26, 20, 21, 19, 10, 5, 7].

Za optimizacijo računsko težkih problemov se danes uporabljajo različne tehnike, kot so simulirano ohlajanje, optimizacija s pomočjo roja delcev in različne evolucijske tehnike.

Algoritem DE lahko prištevamo k evolucijskim algoritmom, ki imajo nekatere prednosti v primerjavi z drugimi metodami. Evolucijski algoritmi potrebujejo le funkcijo, ki jo želimo optimirati. Ponavadi govorimo o iskanju globalnega minimuma.

Izbira ustreznih krmilnih parametrov algoritma je ponavadi odvisna od problema, ki ga rešujemo, in od uporabnika zahteva predhodno znanje oziroma izkušnje. Čeprav je izbira parametrov zelo pomembna, ni nobene metodologije za določitev njihovih "optimalnih" vrednosti.

Pri optimizaciji funkcije je cilj poiskati vektor $\mathbf{x}_{min}$, za katerega velja: $\forall \mathbf{x}$, $f(\mathbf{x}_{min}) \leq f(\mathbf{x})$. Funkcija f ni treba, da je zvezna ali odvedljiva, mora pa biti navzdol omejena.

Algoritem DE ima tri krmilne parametre (F - faktor skaliranja, CR - krmilni

parameter križanja, NP - velikost populacije), ki se pri originalnem algoritmu DE med optimizacijskim postopkom ne spreminjajo [28, 21, 20].

Mehanizem samoprilagajanja (ang. *self-adaptation*) pa omogoča, da se evolucijska strategija (sama) prilagodi naravi problema, tako da se ustrezno rekonfigurira. Ta prilagoditev se opravi brez aktivne vloge uporabnika [2, 3, 11]. Eksperimenti v [7, 5, 6] so pokazali, da s spreminjanjem krmilnih parametrov med optimizacijskim postopkom lahko izboljšamo rezultate. V literaturi zasledimo, da avtorji ponavadi uporabljajo samoprilagajanje pri krmilnih parametrih F in CR [21, 20, 7].

3.2 Algoritem diferencialne evolucije

Algoritem diferencialne evolucije (DE) ustvari novega posameznika s kombinacijo starša in izbranih posameznikov iste populacije. Nov posameznik zamenja starša, če ima boljšo ocenitveno vrednost.

Pri algoritmu DE [28, 29, 26] populacija vsebuje NP D-dimenzionalnih vektorjev oziroma posameznikov:

$$\mathbf{x}_{i,G} = (x_{i,1,G}, x_{i,2,G}, ..., x_{i,D,G}),\ i = 1, 2, ..., NP.$$

Z G označimo generacijo. V eni generaciji algoritem DE uporabi mutacijo in križanje na vsakem vektorju $\mathbf{x}_{i,G}$ in ustvari nov vektor (novega posameznika):

$$\mathbf{u}_{i,G} = (u_{i,1,G}, u_{i,2,G}, ..., u_{i,D,G}),\ i = 1, 2, ..., NP.$$

V eni generaciji algoritem DE ustvari NP novih posameznikov. Nato uporabi selekcijo, da izbere vektorje za naslednjo generacijo $(G + 1)$.

Začetna populacija je izbrana naključno (po enakomerni porazdelitvi) med spodnjo ($x_{j,low}$) in zgornjo ($x_{j,upp}$) mejo, ki sta definirani za vsako spremenljivko x_j. Spodnjo in zgornjo mejo definira uporabnik glede na naravo problema, ki ga rešuje. Po inicializaciji algoritem DE izvede več transformacij (operacij) v postopku, ki ga imenujemo evolucija.

3.2.1 Mutacija

Mutacija za vsak vektor iz populacije ustvari *mutiran* vektor:

$$\mathbf{x}_{i,G} \Rightarrow \mathbf{v}_{i,G} = (v_{i,1,G}, v_{i,2,G}, ..., v_{i,D,G}),\ i = 1, 2, ..., NP.$$

Algoritem *DE* ustvari mutiran vektor z uporabo ene izmed strategij mutacije, ki so podrobneje opisane v poglavju 3.2.4.

3.2.2 Križanje

Po končani mutaciji sledi križanje, ki iz posameznika i in pripadajočega mutiranega vektorja ustvari novega posameznika i pri strategiji *DE/*/*/bin*:

$$u_{i,j,G} = \begin{cases} v_{i,j,G} & \text{če } rand(0,1) \leq CR \text{ ali } j = j_{rand}, \\ x_{i,j,G} & \text{sicer} \end{cases}$$

$$i = 1, 2, ..., NP \text{ in } j = 1, 2, ..., D.$$

CR je krmilni parameter križanja na intervalu $[0, 1)$ in pomeni verjetnost, da se komponenta vektorja pri novem posamezniku ustvari iz komponente mutiranega vektorja. Indeks j_{rand} je naključno izbrano naravno število na intervalu $[1, NP]$. Njegova naloga je, da je vsaj ena komponenta vektorja pri novem posamezniku izbrana iz mutiranega vektorja. Obstaja tudi drug tip križanja (npr.*exp*).

3.2.3 Selekcija

Glede na ocenitveno vrednost posameznika iz populacije in ocenitveno vrednost pripadajočega novega pozameznika z operacijo selekcije izberemo, kateri od obeh omenjenih posemeznikov bo preživel in bo uvrščen v naslednjo generacijo. Če na primer iščemo globalni minimum, uporabimo za selekcijo naslednje pravilo:

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G} & \text{če } f(\mathbf{u}_{i,G}) \leq f(\mathbf{x}_{i,G}), \\ \mathbf{x}_{i,G} & \text{sicer.} \end{cases}$$

Algoritem *DE* ima preprosto selekcijo v primerjavi z ostalimi evolucijskimi algoritmi.

3.2.4 Strategije

Price in Storn [29] sta uvedla algoritem diferencialne evolucije (*DE*). Predlagala sta tudi nekaj osnovnih pravil glede izbire ključnih parametrov, kot so *NP* - število populacij, *CR* - parameter križanja in *F* - faktor skaliranja, ki so uporabni za naključno diferencialno evolucijo pri reševanju kakršnega koli problema. Na njihovi spletni strani [30] sta objavila različne primere aplikacij *DE* skupaj s programsko kodo za različne programske jezike. Prav tako sta tudi predlagala deset različnih strategij za reševanje problemov z diferencialno evolucijo in nekaj smernic, kako uporabiti te strategije za različne probleme. V *DE* algoritem lahko vključimo različne strategije odvisno od vrste problema, ki ga rešujemo. Strategije se razlikujejo glede na vektor, ki ga mutiramo, število razlik vektorjev, ki se mutirajo, in tip križanja. Predlagala sta naslednjih deset strategij:

1. *DE/best/1/exp*
2. *DE/rand/1/exp*
3. *DE/rand-to-best/1/exp*
4. *DE/best/2/exp*
5. *DE/rand/2/exp*
6. *DE/best/1/bin*
7. *DE/rand/1/bin*
8. *DE/rand-to-best/1/bin*
9. *DE/best/2/bin*
10. *DE/rand/2/bin*

V splošnem so strategije zapisane v formatu *DE/x/y/z*. *DE* označuje, da gre za uporabo diferencialne evolucije, *x* predstavlja vektor, ki ga mutiramo in se izbira naključno (*rand*), ali kot najboljši posameznik v trenutni populaciji (*best*). Oznaka *y* je število vektorjev, ki nastopajo v razliki pri mutaciji, *z* pa označuje tip križanja: binominalno ali eksponentno (*bin*, *exp*).

Matematično opišemo omenjene strategije mutacij na naslednje načine:

1. "rand/1": $\mathbf{v}_{i,G} = \mathbf{x}_{r_1,G} + F \cdot (\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G})$

2. "best/1": $\mathbf{v}_{i,G} = \mathbf{x}_{best,G} + F \cdot (\mathbf{x}_{r_1,G} - \mathbf{x}_{r_2,G})$

3. "current to best/1":
   $\mathbf{v}_{i,G} = \mathbf{x}_{i,G} + F \cdot (\mathbf{x}_{best,G} - \mathbf{x}_{i,G}) + F \cdot (\mathbf{x}_{r_1,G} - \mathbf{x}_{r_2,G})$

4. "best/2":
   $\mathbf{v}_{i,G} = \mathbf{x}_{best,G} + F \cdot (\mathbf{x}_{r_1,G} - \mathbf{x}_{r_2,G}) + F \cdot (\mathbf{x}_{r_3,G} - \mathbf{x}_{r_4,G})$

5. "rand/2":
   $\mathbf{v}_{i,G} = \mathbf{x}_{r_1,G} + F \cdot (\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) + F \cdot (\mathbf{x}_{r_4,G} - \mathbf{x}_{r_5,G})$

Indeksi r_1, r_2, r_3, r_4, r_5 predstavljajo naključna in paroma različna naravna števila na intervalu $[1, NP]$. Indeksi so različni tudi od indeksa i. F je mutacijski skalirni faktor (realno število) na intervalu $(0, 2]$, ponavadi manjši kot 1. $\mathbf{x}_{best,G}$ je vektor z najboljšo ocenitveno vrednostjo oziroma najboljši posameznik generacije G.

Poglavje 4

Problem trgovskega potnika

Problem, s katerim se verjetno sreča vsak trgovski potnik, je, da mora obiskati vsa mesta na danem seznamu, začenši z domačim (ki obenem predstavlja tudi končno mesto njegove poti) tako, da bo dolžina poti, ki jo opravi, čim krajša. Pri tem mora obiskati vsako mesto natanko enkrat, izjema je le začetno mesto.

Problem trgovskega potnika (TSP - *ang. traveling salesman problem*) velja za enega od najpomembnejših optimizacijskih problemov in je zanimiv za mnoge raziskovalce, ki skušajo zanj razviti eksaktne ali hevristične algoritme.

Reševanje naloge trgovskega potnika sodi v skupino zelo zahtevnih *NP*-težkih problemov [18, 4]. Če velja P≠NP[1], potem za NP-težke probleme ne obstajajo polinomski algoritmi.

Matematično problem TSP preprosto formuliramo takole: V uteženem polnem grafu je treba poiskati minimalni Hamiltonov cikel. Hamiltonov cikel je sprehod v grafu, na katerem je vsaka točka, razen prve, ki je hkrati tudi zadnja, natanko enkrat. Utež sprehoda je vsota uteži njegovih povezav (osnovne pojme teorije grafov glej npr. [33]).

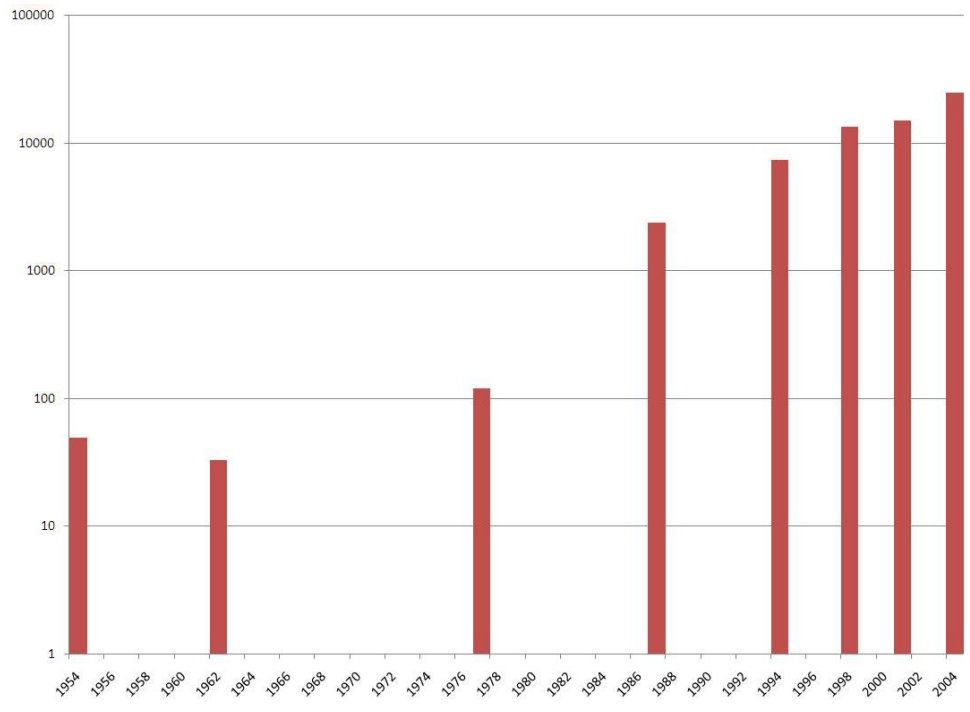
4.1 Zgodovinski pregled razvoja problema trgovskega potnika

Podatki o raziskovanju problemov, podobnih problemu trgovskega potnika, izhajajo iz 19. stoletja, ko sta se s podobnimi problemi ukvarjala irski matematik sir William

[1] P≠NP je verjetno najbolj znan odprti problem teoretičnega računalništva (glej [14]). Prevladuje mnenje, da velja P≠NP.

Rowan Hamilton in britanski matematik Thomas Penyngton Kirkman. Hamilton je celo izdelal igro, kjer je bilo potrebno na polju z mrežo nad 20 točkami poiskati obhode z uporabo le prikazanih povezav. Prvo omembo problema trgovskega potnika najdemo že leta 1832 v nemški knjigi z naslovom *Der Handlungsreisende*, ki pa se bistva problema trgovskega potnika loti le na kratko v svojem zadnjem poglavju.

Kot pravi avtorica diplomskega dela z naslovom *Christofidesov algoritem* [24], zaznamki kažejo, da bi se naj s splošno obliko problema trgovskega potnika v tridesetih letih 20. stoletja prvi ukvarjal matematik Karl Menger na Dunaju in Harvardu. Brez dvoma lahko kot 'krivca' za prenos problema trgovskega potnika v matematične kroge določimo Merrilla Flooda, ki je v poznih štiridesetih letih 20. stoletja prvi omenil to ime v svojih zapiskih in je bil v tistem času pravi glasnik problema trgovskega potnika. Njegovi zapisi so temeljili na ustnem izročilu Hasslerja Whitneya. Tedaj se je začela doba problema trgovskega potnika v matematičnih krogih, ki se je obdržala vse do danes. Raziskovalci širom sveta poizkušajo najti načine za rešitev problema trgovskega potnika na čim večjem številu mest. Na sliki 4.1 je grafično ponazorjen trend naraščanja najdenih optimalnih rešitev problema trgovskega potnika na n mestih od začetkov sredi 20. stoletja do danes. Za lažjo ponazoritev je prikaz v logaritemski skali.



Slika 4.1: Grafična ponazoritev naraščanja števila mest v optimalnih rešitvah od začetkov raziskovanja problema trgovskega potnika do danes [24]

Glede na to, da se razvoj računalnikov razvija po Moorovem zakonu, po katerem se hitrost pomnilniških zmogljivosti podvoji vsakih 24 mesecev, lahko v prihodnosti sicer pričakujemo rešitve za višje število mest, vendar vzrok za to ne bodo boljši algoritmi, temveč zmogljivejši računalniki.

4.2 Opis problema trgovskega potnika

Definicija: Dan je graf z n vozlišči. Za vsak par vozlišč c_i in c_j je podana razdalja $d(c_i, c_j)$, ki jo krajše označimo z d_{ij}. Cilj pri reševanju TSP je poiskati permutacijo vozlišč, označimo jo s π, ki minimizira:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

Probleme trgovskega potnika lahko delimo na dva dela:

- *simetrične* in
- *nesimetrične*.

Pri simetričnih problemih za vsako vozlišče velja $d_{ij} = d_{ji}$, kar pomeni, da je razdalja od vozlišča i do vozlišča j enaka razdalji od j od i. Pri nesimetričnih problemih je lahko $d_{ij} \neq d_{ji}$.

Poleg teh dveh glavnih razdelitev problemov obstaja še mnogo različic problema trgovskega potnika:

- TSP s ponavljanjem mest, *kjer pogoj, da mora biti vsako mesto obiskano* natanko *enkrat, spremenimo v milejši pogoj, da mora biti vsako mesto obiskano* vsaj *enkrat*.
- TSP z več potniki, *kjer imamo m potnikov, ki morajo obiskati določena mesta, vsako natanko enkrat. Vsako od mest mora obiskati vsaj en potnik*.
- TSP z najkrajšo medmestno razdaljo, *kjer želimo namesto skupne razdalje minimizirati razdaljo, ki jo potnik prevozi (ali prehodi) med dvema zaporednima obiskoma pri svojem obhodu*.

Poseben primer simetričnega trgovskega potnika so evklidski problemi trgovskega potnika, kjer so vozlišča grafa razporejena v dvodimenzionalnem (tridimezionalnem)

prostoru, razdalje med njimi pa so definirane z evklidsko razdaljo. Problem trgovskega potnika si lahko predstavljamo takole: vozlišča grafa so kraji, ki jih mora trgovski potnik obiskati, povezave pa poti, po katerih hodi (v današnjem času bi raje rekli, da se vozi) iz kraja v kraj. Povezave med kraji so različno dolge in imajo še dodatne omejitve (slabo vozna cesta, gost promet itd.). Vsaki povezavi priredimo utež, ki pomeni ceno, potrebno, da trgovski potnik opravi to pot. Trgovski potnik mora obiskati vsak kraj natanko enkrat in se vrniti v izhodiščni kraj ter pri tem narediti najkrajšo pot, oziroma stroški potovanja morajo biti minimalni.

Omenili smo že, da sodi problem trgovskega potnika v skupino NP težkih problemov (ne moremo jih rešiti v polinomskem času). Naivno iskanje optimalne rešitve zahteva pregled vseh mogočih Hamiltonovih ciklov danega grafa. Vseh Hamiltonovih ciklov v polnem grafu je $(n-1)!$, kjer je n število vozlišč grafa. Zato naivni pristop (požrešna metoda) odpade že pri zelo majhnih n.

Ker je problem TSP NP-težak, uporabljamo hevristične algoritme [27, 18, 6], ki imajo manjše časovne zahtevnosti, s katerimi pa ne moremo vedno poiskati optimalne rešitve. Optimalni rešitvi se bolj ali manj približamo.

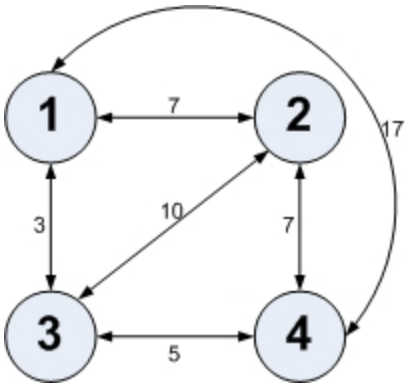
V nadaljevanju poglavja opišemo simetrični in nesimetrični problem trgovskega potnika ter spregovorimo o njuni predstavitvi.

4.2.1 Simetrični problem trgovskega potnika

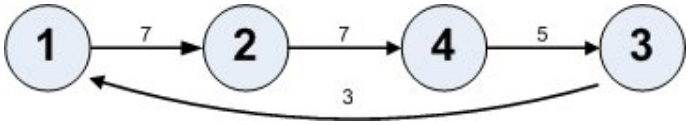
Optimalno rešitev lahko dobimo z algoritmi, ki dajo točno rešitev (dinamično programiranje, sestopanje, razveji in omeji...).

$$C = \begin{bmatrix} 0 & 7 & 3 & 17 \\ 7 & 0 & 10 & 7 \\ 3 & 10 & 0 & 5 \\ 17 & 7 & 5 & 0 \end{bmatrix}$$

Problem algoritmov, ki dajo točno rešitev, je ta, da so algoritmi počasni in za izračun pri večjem številu vozlišč odpovejo zaradi prevelike časovne zahtevnosti.



Slika 4.2: Primer grafa za problem simetričnega trgovskega potnika

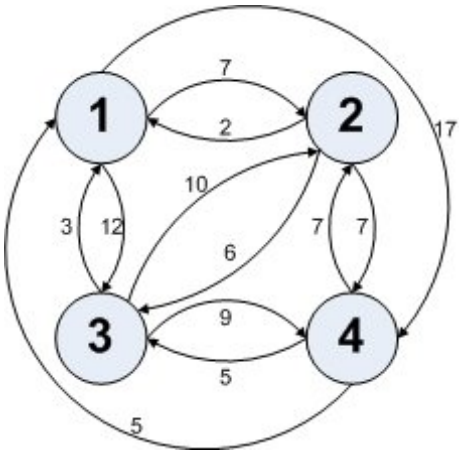


Slika 4.3: Rešitev najkrajše poti TSP za primer iz slike 4.2. Rešitev najkrajše poti za ta primer je $g(1,[2,3,4]) = C[1,2] + g[2,[3,4]) = 7 + 15 = 22$ (*Bellmanova* enačba)

4.2.2 Nesimetrični problem trgovskega potnika

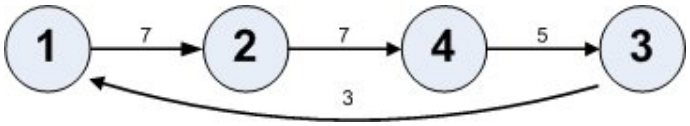
Nesimetrični problem trgovskega potnika (ATSP - *ang. asymmetric traveling salesman problem*) pomeni, da razdalja med vozlišči $d_{ij} \neq d_{ji}$.

$$C = \begin{bmatrix} 0 & 7 & 12 & 17 \\ 2 & 0 & 6 & 7 \\ 3 & 10 & 0 & 9 \\ 5 & 7 & 5 & 0 \end{bmatrix}$$



Slika 4.4: Primer grafa za problem nesimetričnega trgovskega potnika

Enako kot pri reševanju problemov simetričnega trgovskega potnika, so 'naivni' algoritmi, ki rešujejo probleme nesimetričnega trgovskega potnika počasni, oziroma še počasnejši, in prav tako odpovejo zaradi prostorske omejitve, ali pa je trajanje, da izračunajo rešitev, zelo dolgo.



Slika 4.5: Rešitev najkrajše poti ATSP za primer iz slike 4.4. Rešitev najkrajše poti za ta primer je $g(1,[2,3,4]) = C[1,2] + g[2,[3,4]) = 7 + 15 = 22$ (*Bellmanova* enačba)

4.3 Algoritem RAI

V tem poglavju opisujemo hevristični algoritem, ki temelji na gradnji cikla in uporablja lokalno optimizacijo. Avtorji v članku [9] prikazujejo, kako se da z dokaj preprostimi hevrističnimi metodami priti do dobrih rezultatov. Algoritem, ki ga predstavljajo ima majhno prostorsko in časovno zahtevnost.

4.3.1 Lokalna optimizacija

Lokalna optimizacija je zelo dobro poznana in velikokrat uporabljena splošno namenska hevristika. Osnovni korak pri lokalni optmimizaciji prikazuje algoritem 2 [1]. Korak 1 je *inicializacijski del*, ki zgradi začetno rešitev S. Korak 2 predstavlja *optimizacijski del*, ki izboljšuje trenutno rešitev s pomočjo algoritma lokalnega iskanja.

Začetno rešitev lahko poiščemo naključno ali s pomočjo kakega algoritma. Transformacija v koraku 3 mora biti poceni v primerjavi z iskanjem začetne rešitve v koraku 1.

Algoritem 2 Algoritem *RAI*

1: poišči začetno rešitev S
2: **while** zaključni pogoj ni izpolnjen **do**
3: transformiraj S v S'
4: **if** S' boljša rešitev kot S **then**
5: $S = S'$
6: **end if**
7: **end while**
8: Izhod: S

Pri problemu TSP je najpogosteje uporabljena lokalna optimizacija tipa *k-opt*. Pri lokalni optimizaciji *2-opt* sosede danega obhoda dobimo na primer tako, da izberemo dve nesosednji povezavi, recimo $e = c_{\pi(i)}c_{\pi(i+1)}$ in $f = c_{\pi(j)}c_{\pi(j+1)}$, ki ju nadomestimo z $e' = c_{\pi(i)}c_{\pi(j)}$ in $f' = c_{\pi(j+1)}c_{\pi(i+1)}$ [4, 18].

V primeru, ki so ga avtorji opisali [9], dobimo začetno rešitev s pomočjo hevristike, ki gradi cikel. Hevristični algoritem je predstavljen v poglavju 4.3.2. Lokalna optimizacija je zasnovana na isti način kot gradnja začetnega cikla (začetne rešitve).

Iteracijski korak pri optimizacijski metodi je naslednji: iz cilka, ki je trenutna rešitev, odstranimo množico vozlišč in jih nato ponovno vstavimo v cikel glede na optimizacijsko kriterijsko funkcijo.

4.3.2 Opis algoritma *RAI*

Predstavljen hevristični algoritem je preprost in ima polinomsko časovno zahtevnost [6, 8, 9].

1. Začetni cikel naj sestoji iz poljubnega vozlišča in zanke.
2. Naključno izberi vozlišče, ki ga še ni v ciklu.
3. Izbrano vozlišče vstavi med dve sosednji vozlišči v ciklu na najcenejši način. Če cikel ne vsebuje vseh vozlišč, pojdi na korak 2.
4. Pomni trenutno rešitev, poimenujmo jo S.
5. Korake od 6 do 10 ponovi P-krat.
6. Naključno izberi i in j $(i, j \in N = \{1, \ldots, n\}, 1 \leq i \leq j \leq n)$.
7. Iz S odstrani del poti, ki se začne z i in konča z j, in poveži vozlišče $i - 1$ z vozliščem $j + 1$.
8. Naključno izberi vozlišče na odstranjeni poti.
9. Izbrano vozlišče vstavi med dve sosednji vozlišči v ciklu na najcenejši način. Če cikel ne vsebuje vseh vozlišč, pojdi na korak 8.
10. Trenutno novo dobljeno rešitev primerjaj z rešitvijo S in boljšo obdrži.

Prvi štirje koraki zgradijo začetno rešitev. Lokalna optimizacija se izvaja v zanki, ki zajema korake od 6 do 10. Iz cikla, ki pomeni trenutno rešitev, odstranimo nekaj sosednjih vozlišč, ki jih nato drugo za drugo ponovno vstavimo v cikel na najcenejši način.

Ko cikel iz koraka 1 vsebuje eno vozlišče, izbrano vozlišče vstavimo v cikel (obstaja le ena možnost vstavitve) in rezultat je cikel z dvema vozliščema.

Zaključni pogoj lokalne optimizacije je, "ko ni več mogoče izboljšanje rešitve". Ta pogoj ni enostavno testirati, saj bi morali vsakič pregledati vse dopustne rešitve iz okolice trenutno iskane rešitve, kar je seveda časovno zelo potratno. Pogosto izbrana možnost je, da zaključni pogoj uporablja parameter P. Če je več kot P zaporednih izbir neuspešnih, zaključimo iskanje. V članku [9] so avtorji, predvsem zaradi časovne zahtevnosti, izbrali fiksno število korakov lokalne optimizacije ($P = n^2$, $2n^2$ in n^3, kjer je n število vozlišč grafa).

V vsaki ponovitvi je število odstranjenih in ponovno vstavljenih vozlišč največ n in pri vsakem vstavljanju vozlišča v cikel je mogočih največ n različnih mest vstavitve, torej primerjav. Od tod izvira ideja za n^2 ponovitev optimizacijske zanke.

V najslabšem primeru je časovna zahtevnost algoritma RAI: $T_\omega(n) = O(n^4)(O(n^5)$ pri $P = n^3$).

4.3.3 Implementacija

Za rešitev, ki so jo avtorji [9] v algoritmu RAI poimenovali S, so uporabili enodimenzionalno polje. Korak 6 so izvedli tako, da so z uporabo generatorja naključnih števil določili vrednosti i in j ($i, j \in N = \{1, 2, \ldots, n\}$). Če pogoj $1 \leq i \leq j \leq n$ ni bil izpoljnjen, so vrednosti i in j zamenjali in tako dosegli izpolnitev pogoja.

Z namenom, da so vozlišča na poti, ki so jih odstranili iz cikla (korak 7), izbrana naključno, so po končanem koraku 10 izvedli rotacijo polja S (za eno mesto v desno). Tako so dosegli implementacijsko neodvisnost predstavitve cikla z uporabo enodimenzionalnega polja; in drugič, kar je še pomembnejše, rezultat rotacije in zamenjave vrednosti i in j v primeru, ko je $j > i$, je naslednji: povprečno število vozlišč na odstranjeni poti je reda $n/3$.

4.3.4 Rezultati

Algoritem so avtorji preiskusili na grafih iz knjižnjice TSPLIB [25], ki jo najdemo na internetni spletni strani *http://softlib.rice.edu/softlib/tsplib*. Za grafe ATSP v tej knjižnjici so znane optimalne rešitve, ki predstavljajo dobro oceno za delovanje tega hevrističnega algoritma.

Rezultati, ki so jih avtorji dobili pri poskusu [9], so prikazani v tabeli 4.1. Drugi in tretji stolpec prikazujeta ime grafa ATSP in število vozlišč. Četrti stoplec pa prikazuje najboljšo znano oziroma optimalno rešitev.

Čas, v katerem je algoritem našel najboljšo rešitev, je prikazan v petem stolpcu in je izmerjen na osebnem računalniku (PC-i586/Linux, 800Mhz).

Tabela 4.1: Rezultati na nesimetričnih problemih trgovskega potnika [9]

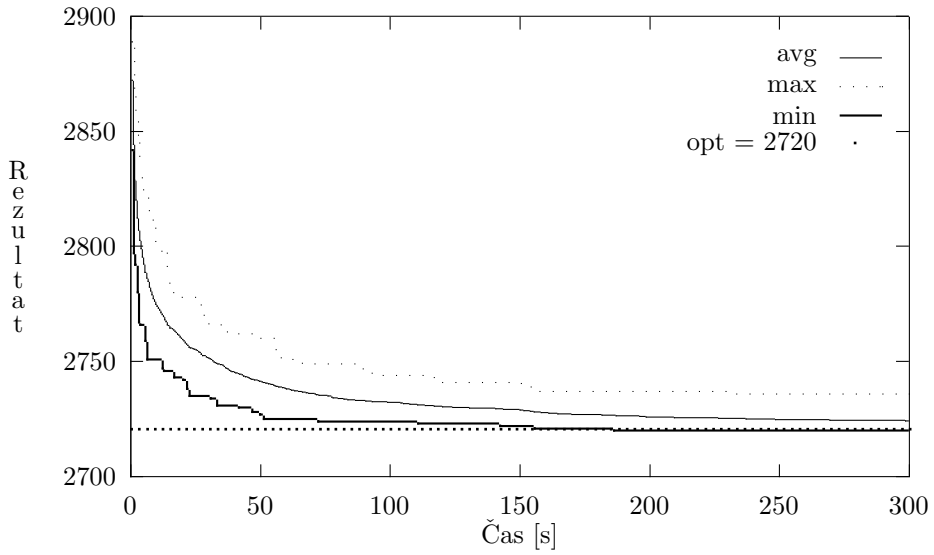
	Graf	n	Optimum	$t[s]$	min.	%	avg.	%	(n^2)	$(2n^2)$
1	br17	17	39	0,001	39	0,00	39,00	0,00	100	100
2	ftv33	34	1286	0,003	1286	0,00	1287,08	0,08	95	98
3	ftv35	36	1473	0,030	1473	0,00	1479,35	0,43	19	17
4	ftv38	39	1530	0,039	1530	0,00	1537,42	0,48	11	21
5	p43	43	5620	0,056	5620	0,00	5620,71	0,01	24	30
6	ftv44	45	1613	0,071	1613	0,00	1636,49	1,45	26	16
7	ftv47	48	1776	0,085	1776	0,00	1780,18	0,24	26	34
8	ry48p	48	14422	0,122	14422	0,00	14512,50	0,63	3	2
9	ft53	53	6905	0,091	6905	0,00	6929,55	0,36	44	51
10	ftv55	56	1608	0,127	1608	0,00	1617,16	0,57	27	42
11	ftv64	65	1839	0,324	1839	0,00	1849,94	0,60	17	32
12	ft70	70	38673	0,463	38762	0,23	39095,30	1,09	0	0
13	ftv70	71	1950	0,463	1950	0,00	1959,15	0,47	7	28
14	ftv90	91	1579	0,636	1579	0,00	1581,56	0,16	11	14
15	kro124p	100	36230	0,891	36241	0,03	37088,90	2,37	0	0
16	ftv100	101	1788	0,834	1788	0,00	1790,78	0,16	5	21
17	ftv110	111	1958	1,220	1958	0,00	1961,65	0,19	16	15
18	ftv120	121	2166	1,966	2166	0,00	2175,34	0,43	5	5
19	ftv130	131	2307	2,479	2307	0,00	2319,88	0,56	4	6
20	ftv140	141	2420	3,426	2420	0,00	2430,95	0,45	5	7
21	ftv150	151	2611	4,545	2611	0,00	2649,08	1,46	17	7
22	ftv160	161	2683	5,231	2683	0,00	2719,46	1,36	18	21
23	ftv170	171	2755	7,157	2755	0,00	2796,82	1,52	3	5
24	rbg323	323	1326	220,2	1339	0,98	1354,54	2,15	0	0
25	rbg358	358	1163	162,3	1163	0,00	1172,36	0,80	1	0
26	rbg403	403	2465	182,3	2465	0,00	2465,57	0,02	91	85
27	rbg443	443	2720	270,7	2720	0,00	2720,43	0,02	72	80

V naslednjih štirih stolpcih so prikazani naslednji podatki: najboljša rešitev (oznaka 'min.') in povprečna rešitev (oznaka 'avg.') ter vrednosti, ki povesta, koliko odstotkov sta rešitvi (najboljša in povprečna) večji od optimalne rešitve. Za vsak graf so izvedli 100 meritev. Pri vsaki meritvi so zgradili n^2 Hamiltonovih ciklov.

Ugotovimo lahko, da je hevristični algoritem *RAI* uspel najti rešitev pri 24 grafih, ki je bila enaka optimalni rešitvi. Pri treh grafih mu to ni uspelo, in sicer pri grafih: *ft70*, *kro124p* in *rbg323*. Opazimo še lahko, da je algoritem vedno našel rešitev, ki se od optimalne rešitve razlikuje za manj kot en odstotek.

Zanimivo je pogledati tudi povprečno vrednost rešitve. Rezultati v tabeli 4.1 kažejo, da se povprečna vrednost rešitve razlikuje od optimalne rešitve največ za 2.37%. Le še na enem grafu (*rbg323*) ta razlika preseže 2%, v petih primerih je omenjena razlika med 1% in 2% in kar v 20 primerih je ta razlika manjša od 1%.

V tabeli 4.1 so v desnih dveh stolpcih prikazani rezultati, ki povedo, kolikokrat je hevristični algoritem *RAI* našel rešitev, ki je bila enaka optimalni. Oznaka '(n^2)' pomeni, kolikokrat je algoritem našel optimalno rešitev pri ponovitvi lokalne optimizacije n^2 (korak 5), oziroma '$(2n^2)$' pri $2n^2$ ponovitvah.



Slika 4.6: Rezultat v odvisnosti od časa za graf *rbg443* [9]

Slika 4.6 prikazuje, kako se minimalna, maksimalna in povprečna vrednost približajo optimalni rešitvi glede na čas izvajanja algoritma za izbran graf.

Uporabnost algoritma lahko ocenjujemo tudi tako, da ga primerjamo z algoritmi istega tipa. Tako bomo v tej diplomski nalogi primerjali ta algoritem tudi z drugimi hevrističnimi algoritmi.

4.4 Hevristični algoritem *RAI+OR*

Hevristični algoritem *RAI+OR* [6] smo razvili na način, da smo združili dva že znana algoritma. Prvi je algoritem *RAI* [9, 8], ki je opisan v poglavju 4.3, drugi pa je v literaturi znan algoritem *OR-opt* [18], ki ga bomo na kratko opisali na koncu tega poglavja.

Tako dobljen nov hevristični algoritem *RAI+OR* želimo uporabiti za reševanje naloge predvsem nesimetričnega trgovskega potnika. Predstavljeni algoritem je preprost in ima polinomsko časovno zahtevnost.

Ideja algoritma je naslednja: iz cikla, ki je trenutna rešitev, odstranimo množico vozlišč in jih nato ponovno vstavimo v cikel glede na optimizacijsko kriterijsko funkcijo ter ob tem na trenutno dobljenem ciklu uporabimo algoritem *OR*.

Algoritem *RAI+OR*:

1. Začetni cikel naj sestoji iz poljubnega vozlišča in zanke.
2. Naključno izberi vozlišče, ki ga še ni v ciklu.
3. Izbrano vozlišče vstavi med dve sosednji vozlišči v ciklu na najcenejši način. Če cikel ne vsebuje vseh vozlišč, pojdi na korak 2.
4. *Nad dobljenim polnim ciklom izvedi algoritem OR-opt.*
5. Pomni trenutno rešitev, poimenujmo jo S.
6. Korake od 7 do 13 ponovi P-krat.
7. Naključno izberi i in j ($i, j \in N = \{1, \ldots, n\}$, $1 \leq i \leq j \leq n$).
8. Iz S odstrani del poti, ki se začne z i in konča z j, in poveži vozlišče $i-1$ z vozliščem $j+1$.
9. *Nad dobljenim ciklom, ki ne vsebuje v prejšni točki odstranjene poti, izvedi algoritem OR-opt.*

10. Naključno izberi vozlišče na odstranjeni poti.

11. Izbrano vozlišče vstavi med dve sosednji vozlišči v ciklu na najcenejši način. Če cikel ne vsebuje vseh vozlišč, pojdi na korak 10.

12. *Nad dobljenim polnim ciklom izvedi algoritem OR-opt.*

13. Trenutno novo dobljeno rešitev primerjaj z rešitvijo S in boljšo obdrži.

Prvih pet korakov zgradi začetno rešitev. Lokalna optimizacija se izvaja v glavni zanki, ki zajema korake od 7 do 13. Iz cikla, ki pomeni trenutno rešitev, odstranimo nekaj sosednjih vozlišč, izvedemo optimizacijo *OR-opt*, nato odstranjena vozlišča drugo za drugo ponovno vstavimo v cikel na najcenejši način. Ko cikel vsebuje vsa vozlišča, izvedemo še optimizacijo OR-opt.

Algoritem RAI+OR smo dobili tako, da smo algoritmu RAI [9, 8] dodali korake 4, 9 in 12, ki so napisane poševno.

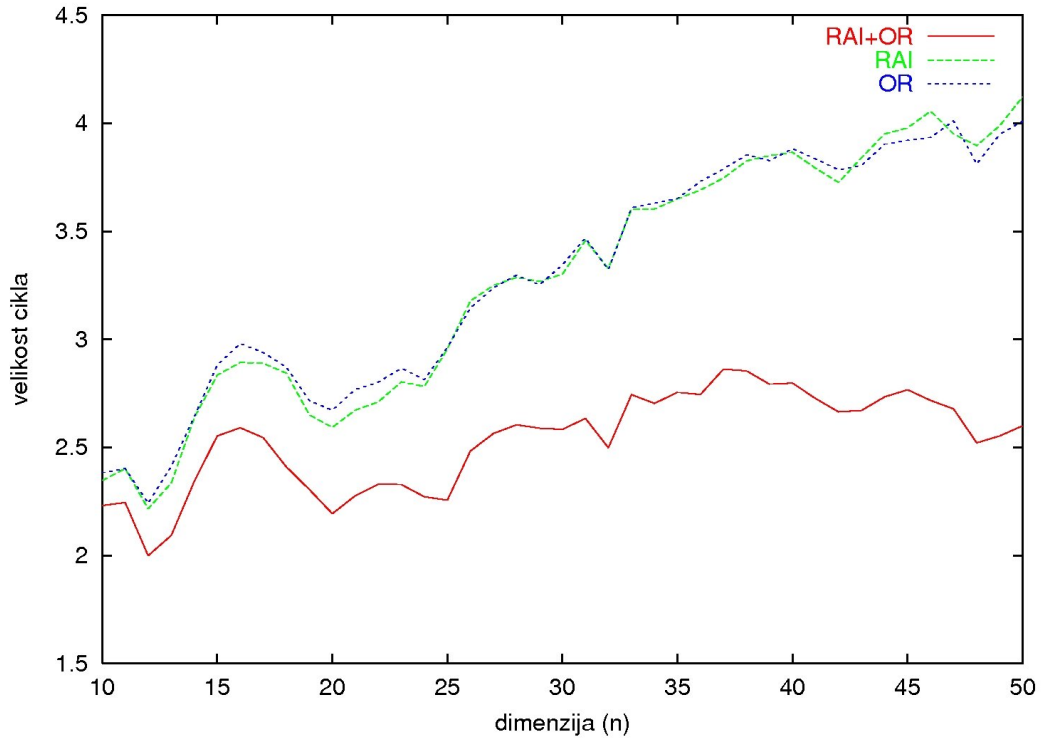
Predvsem zaradi časovne zahtevnosti, smo izbrali fiksno število korakov lokalne optimizacije ($P = n^2$, kjer je n število vozlišč grafa).

OR-opt [18] (str. 220) je algoritem lokalne optimizacije in deluje takole. Za vsako povezano pot iz s vozlišč (najprej je s enako 3, zatem 2 in na koncu 1), preverimo, ali lahko vstavimo pot med poljubni dve vozlišči, da dobimo boljšo rešitev. Če to lahko storimo, pot vstavimo. Ko smo izčrpali vse poti dolžine 3, nadaljujejo s potmi dolžine 2, ter na koncu še s potmi z enim vozliščem. Ko s takim vstavljenjem ni več izboljšanj rešitve, se algoritem zaključi.

Algoritem OR, ki ga bomo v nadaljevanju primerjali z algoritmoma RAI+OR in RAI, smo implementirali takole:
naključno smo generirali celotno rešitev, nato pa smo izvedli optimizacijo OR-opt in celotno zadevo ponovili $P = n^2$ krat. Med n^2 rešitvami smo poiskali najboljšo dobljeno rešitev.

4.4.1 Rezultati

V tem poglavju prikažemo dobljene rezultate delovanja algoritmov *RAI*, *RAI+OR* in *OR* na grafih, ki smo jih generirali na naslednji način. Za dano dimenzijo n smo naključno (enakomerna porazdelitev na $[0, 1]$) generirali povezave polnega (simetričnega) grafa $d \times d$. Dobili smo grafe velikosti od 10 do 50 vozlišč.



Slika 4.7: Povprečje rešitev v 100 poskusih za grafe od 10 do 50 vozlišč.

Rezultati, ki smo jih dobili pri poskusu, so prikazani na sliki 4.7. Prikazane so rešitve, ki so dobljene kot povprečje v 100 poskusih, za grafe velikosti od 10 do 50 vozlišč. Manjša velikost cikla pomeni boljše rezultate. Opazimo, da za algoritma RAI in OR-opt ne moremo ugotoviti, kateri od njiju daje boljše rezultate. Algoritem RAI+OR pa daje boljše ali enake rezultate kot algoritma RAI in OR. Za grafe, ki imajo več kot 23 vozlišč, pa so rezultati algoritma vedno boljši v primerjavi z rezultati ostalih dveh algoritmov.

Izmerili smo tudi porabljen čas osebnem računalniku pri grafu s 50 vozlišči: RAI+OR je potreboval $0.65s$, OR $0.95s$ in RAI $0.03s$.

Pri testnih grafih smo za grafe z majhno dimenzijo izračunali optimalno rešitev s pomočjo algoritma sestopanja, in sicer za dimenzije od 10 do vključno 17. Za graf

s 17 vozlišči smo potrebovali približno 14 ur, za ostale, večje grafe pa bi izvajanja algoritma trajalo preveč časa. Za grafe od 10 do 17 vozlišč so vsi algoritmi (izjema je le algoritem RAI, ki ni našel optimalne rešitve pri grafu s 17 vozlišči) našli rešitev, ki je optimalna, kar smo preverili oziroma potrdili z algoritmom sestopanja.

Tako lahko vidimo, da kombinacija algoritmov *RAI* in *OR-opt* daje vidno boljše rezultate. Ocenjujemo, da sta algoritma primerna za reševanje problema nesimetričnega trgovskega potnika.

Poglavje 5

Algoritmi DEATSP

V tem poglavju, ki je jedro diplomskega dela, opisujemo reševanje problema nesimetričnega trgovskega potnika s pomočjo diferencialne evolucije. Načrtovali in implementirali smo več algoritmov, ki smo jih s skupnim imenom poimenovali '***Algoritmi DEATSP***' (*ang.* ***D****ifferential* ***E****volution -* ***A****symmetric* ***T****raveling* ***S****alesman* ***P****roblem*). Algoritmi, zgrajeni na takšni osnovi, nam bodo pomagali reševati probleme nesimetričnega trgovskega potnika, kar pomeni, da bodo poskušali poiskati minimalni Hamiltonov cikel v uteženem polnem grafu.

Osnova algoritmom, ki smo jih načrtovali in razvili, je algoritem diferencialne evolucije (*DE*), ki služi iskanju optimumov (največkrat minimumov) zveznih funkcij, kjer je potrebno (kot že omenjeno v poglavju 3.1) poiskati vektor $\mathbf{x}_{min}$, za katerega velja: $\forall \mathbf{x},\ f(\mathbf{x}_{min}) \leq f(\mathbf{x})$.

5.1 Predstavitev posameznikov in ocenitvena funkcija

Tako kot pri algoritmu diferencialne evolucije, smo tudi pri naših algoritmih *DEATSP* tvorili začetno populacijo posameznikov, ki so bili izbrani naključno (po enakomerni porazdelitvi). Zaradi mutacije pri diferencialni evoluciji smo vpeljali drugačen način predstavitve posameznika (cikla), ki ga bomo natančneje opisali v nadaljevanju.

5.1.1 Predstavitev posameznikov v *DE*

Predstavljajmo si, da vsako mesto opisuje število med 0 in 7 (slika 5.1). Geografski položaj je določen s pomočjo koordinat X in Y (tabela 5.1). Rešitev problema trgov-

skega potnika je cikel, ki smo ga dobili iz poti tako, da smo povezali zadnje in prvo mesto. Tako npr. pot 1-2-3-4 predstavlja cikel 1-2-3-4-1. Če takšno pot shranimo v enodimenzionalno polje, se naslenje mesto nahaja na naslednjem indeksu. To predstavitev poimenujmo osnovna predstavitev.

Slika 5.1: Obhodna pot (cikel) med osmimi slovenskimi mesti [31]

Tabela 5.1: Matrika zračnih razdalj med osmimi slovenskimi mesti

#	Mesto	X	Y	1	2	3	4	5	6	7	8
1	Murska Sobota	590	168	0	41	83	122	144	146	208	225
2	Maribor	550	156	41	0	45	91	103	105	168	185
3	Celje	521	121	83	45	0	49	62	70	129	142
4	Novo Mesto	517	69	122	91	49	0	59	80	120	116
5	Ljubljana	468	995	144	103	62	59	0	24	68	83
6	Kranj	450	122	146	105	70	80	24	0	63	91
7	Noga Gorica	403	95	208	168	129	120	68	63	0	45
8	Koper	409	43	225	185	142	116	83	91	45	0

Za lažjo predstavitev posameznikov v algoritmu diferencialne evolucije, smo načrtovali in implementirali funkciji *Tour2Position* (Algoritem 3) in *Position2Tour* (Algoritem 4). Prva pri danem posamezniku preoblikuje njegovo osnovno predstavitev. Nova dobljena predstavitev ima na i-tem indeksu zapisano vozlišče, v katerega gre trgovski potnik (iz i gre v $P[i]$).

Algoritem 3 Funkcija *Tour2Position*

```
1  // T[] ... vhodni posameznik
2  // P[] ... izhodni posameznik
3  // n ... velikost posameznika
4  void Tour2position(int P[], int T[], int n)
5  {
6      int i;
7      for (i = 0; i < n; i++)
8          P[T[i]] = T[ ( i + 1 ) % n];
9  }
```

Funkcija *Position2Tour* pa preoblikuje posameznika nazaj v osnovno predstavitev. Uporabimo jo, preden nadaljujemo postopek izvajanja algoritma diferencialne evolucije, kjer posameznika ocenimo z ocenitveno funkcijo, ter ga v primeru boljšega rezultata ohranimo oziroma nasprotno, zavržemo.

Algoritem 4 Funkcija *Position2Tour*

```
1  // T[] ... vhodni posameznik
2  // P[] ... izhodni posameznik
3  // n ... velikost posameznika
4  void Position2tour(int T[], int P[], int n)
5  {
6      int i;
7      int j;
8      int m;
9      int mnozica[MAX_N];
10     for (m=0;   m<n; m++)
11         mnozica[m] = 0;
12     j   = 0;
13     for (i=0;   i<n; i++) {
14         m = P[j];
15         if (mnozica[m] == 0) {
16             mnozica[m] =    1;   /* ga zasedemo */
17             T[(i+1)%n] =    m;
18             j = P[j];
19         }
20         else
21         {
22             m = (int)(mRandom() * n);
23             while (mnozica[m] ==    1)  /* preskocim ze zasedene */
24                 m   = (m+1)%n;
25             mnozica[m] =    1;   /* ga zasedemo */
26
27             T[(i+1)%n] =    m;
28             j = P[j];
29         }
30     }
31 }
```

5.1.2 Krmilni parametri

Osnovni (krmilni) parametri, ki jih nastavimo ob začetku algoritma diferencialne evolucije, so:

- *NP* - število posameznikov v populaciji,
- *gen* - število generacij,
- *F* - faktor skaliranja in
- *CR* - krmilni parameter križanja.

Začetno množico posameznikov v populaciji inicializiramo naključno, tako da najprej tvorimo množico po vrsti, katero nato naključno premešamo.

V našem delu smo za reševanje problema nesimetričnega trgovskega potnika uporabili dve od desetih strategij algoritma *DE* ter razvili novo strategijo *Xing*. Uporabljene strategije opisujemo v poglavjih 5.2.1, 5.2.2 in 5.2.3.

5.1.3 Ocenitvena funkcija

Ocenitvena funkcija za dan cikel izračuna dolžino poti, ki jo mora potnik prehoditi, da se vrne v svoje začetno mesto. V našem primeru, za reševanje problema nesimetričnega trgovskega potnika, ocenitvena funkcija vedno vrne pozitiven rezultat. Če je ta rezultat večji od prejšnjega (dolžina cikla večja od prejšnjega cikla), pomeni, da nismo našli boljše rešitve. V primeru, ko funkcija vrne manjši rezultat v primerjavi s prejšnjo rešitvijo, pomeni, da smo našli cikel, kjer ima trgovski potnik na voljo krajšo razdaljo, da obišče vsa mesta in se vrne v začetno mesto.

Algoritem 5 Ocenitvena funkcija za trgovskega potnika

```
1  /* k ... number of cities in current tour */
2  int Cena(int G[MAXDIM][MAXDIM], int V[MAXDIM], int k)
3  {
4      int i;
5      int c=0;
6      for (i=1; i<k; i++)
7          c += G[ V[i-1] ][ V[i] ];
8      c += G[V[k-1]] [V[0]];          /* to make a cycle */
9      return c;
10 }
```

Ocenitvena funkcija (Algoritem 5) torej sešteje razdalje med kraji (vozlišči), kjer še poveže zadnje in prvo mesto. To pomeni, da smo ocenili celotno povezano pot med vsemi kraji in končali v začetnem kraju.

5.1.4 Funkcija za povečanje disperzije

Preden predstavimo strategije, s katerimi rešujemo probleme nesimetričnega trgovskega potnika, predstavimo še funkcijo, ki smo jo poimenovali *check_same* (Algoritem 6). Ta funkcija poskrbi, da se po kreiranju novih posameznikov le-ti ne bi ponavljali. To pomeni, da pregleda vse posameznike, ki so v trenutni generaciji in jih primerja s trenutno kreiranim posameznikom. Če je le-ta enak kateremu izmed ostalih, ga na novo naključno generiramo. Verjetnost, da je enak enemu izmed prejšnjih, je zelo majhna. Funkcija nas tako privede k večji disperziji, oziroma k raznolikosti posameznikov in pripomore k učinkovitejšemu iskanju rešitve.

Algoritem 6 Funkcija *check_same* za povečanje disperzije posameznikov

```
1  void check_same(int trenutni[], int D, int NP)
2  {
3      int i, j;
4      bool isti=false;
5      for (i=0; i<NP; i++)
6      {
7          isti=true;
8          for (j=0; j<D; j++)
9          {
10             if (trenutni[j] != ((*pnew)[i])[j])
11             {
12                 isti=false;
13                 break;
14             }
15         }
16         if (isti==true)
17         {
18             Init_P1(trenutni,D);   // NAKLJUCNO
19             Init_NAK(trenutni,D);   // NAKLJUCNO
20             break;
21         }
22     }
23 }
```

5.1.5 Funkcija za preverbo in popravo celovitosti cikla

Funkcija, ki je opisana v algoritmu 7, po vsaki mutaciji najprej preveri celovitost cikla, in če je potrebno, ga tudi popravi.

Algoritem 7 Funkcija *popravi_tmp*, ki preveri (popravi) celovitost poti (cikla)

```
1  /*
2  * TMP[] ... vhodni posameznik
3  * mnozica[] ... vhodno polje
4  * n ... trenutno vozlisce
5  * D ... velikost posameznika
6  */
7  void popravi_tmp(int TMP[], int mnozica[], int n, int D)
8  {
9      int m;
10     if (TMP[n] < 0) // spravi med 0 ... (D-1)
11     {
12         TMP[n] = -TMP[n];
13     }
14
15     TMP[n] = TMP[n]%D;
16     m = TMP[n];
17
18     if (mnozica[m] == 0)
19     {
20         mnozica[m] = 1;    // ga zasedemo
21         TMP[n] = m;
22     }
23     else
24     {
25         m = (int)(mRandom()*D);
26
27         while (mnozica[m] == 1)  // preskočim že zasedene
28             m = (m+1)%D;
29
30         mnozica[m] = 1;    // ga zasedemo
31         TMP[n] = m;
32     }
33 }
```

5.2 Predstavitev strategij za algoritme *DEATSP*

5.2.1 Strategija *DE/rand/1/exp* za trgovskega potnika

Glavna naloga strategije *DE/rand/1/exp* je, da tvori novo pot (cikel) iz treh naključno izbranih poti: *R1*, *R2* in *R3*, po matematični formuli: $\omega = x_1 + F * (x_2 - x_3)$ [28].

Strategija deluje na način, da najprej pretvori trenutnega in naključno izbrane posameznike z uporabo funkcije *Tour2Position* (Algoritem 3). Zatem sledi ustvarjanje novega posameznika (cikla), po funkciji $\omega = x_1 + F * (x_2 - x_3)$. V primeru, ko je funkcija vrnila negativno vrednost, jo moramo negirati, torej kot rezultat dobimo pozitivno vrednost, sicer vozlišča ne moremo predstaviti v naši poti. To delo opravi funkcija *popravi_tmp()* (Algoritem 7), ki prav tako preveri celovitost poti ter jo v primeru napačne predstavitve tudi popravi.

Algoritem 8 Strategija *DE/rand/1/exp* za trgovskega potnika

```
1  F = 0.1+mRandom()*0.9; // nastavitev krmilnega parametra F
2  CR = 0.1+mRandom()*0.8; // nastavitev krmilnega parametra CR
3  Tour2position(TMP, (*pold)[i], D);
4  Tour2position(R1, (*pold)[r1], D);
5  Tour2position(R2, (*pold)[r2], D);
6  Tour2position(R3, (*pold)[r3], D);
7  Init_mnozica(mnozica, D);
8  n = (int)(rnd_uni(&rnd_uni_init)*D);
9  L = 0;
10 do
11 {
12     TMP[n] = (int) ( R1[n] + F*(R2[n] - R3[n]) );
13
14     popravi_tmp(TMP, mnozica, n, D); // popravi trenutnega posameznika
15     n = (n+1)%D;
16     L++;
17 }
18 while((rnd_uni(&rnd_uni_init) < CR) && (L < D));
19 Position2tour(tmp, TMP, D); // DELUJE S POPRAVLJANJEM CIKLA
20
21 if (mRandom() < 0.5)
22     check_same(tmp, D, NP); // pogleda, ali se razlikuje od vseh
       posameznikov
```

Strategija uporablja eksponentno križanje (*exp*) pri algoritmu *DE*. Nato še cikel pretvori nazaj v osnovno predstavitev posameznika in opravi preverjanje raznolikosti posameznika v populaciji (Algoritem 6).

5.2.2 Strategija *DE/rand/1/bin* za trgovskega potnika

Naloga strategije *DE/rand/1/bin* je, da tvori novo pot (cikel) iz treh naključno izbranih poti: *R1*, *R2* in *R3*, po matematični formuli: $\omega = x_1 + F * (x_2 - x_3)$ in pri tem uporablja binominalno križanje (glej zanko *for* v algoritmu 9).

Algoritem 9 Strategija *DE/rand/1/bin* za trgovskega potnika

```
1  F = 0.1+mRandom()*0.9;
2  CR = 0.1+mRandom()*0.8;
3  F2 = 0.1+mRandom()*0.9;
4  Tour2position(TMP, (*pold)[i], D);
5  Tour2position(TMP1, (*pold)[i], D);
6  Tour2position(R1, (*pold)[r1], D);
7  Tour2position(R2, (*pold)[r2], D);
8  Tour2position(R3, (*pold)[r3], D);
9
10 Init_mnozica(mnozica, D);
11 n = (int)(rnd_uni(&rnd_uni_init)*D);  // eden izmed v intervalu D
12 for (L=0; L<D; L++)
13 {
14     if ((rnd_uni(&rnd_uni_init) < CR) || L == (D-1))
15     {
16         TMP[n] = R1[n];
17         TMP1[n] = R1[n];
18
19         TMP[n] = (int) ( R1[n] + F*(R2[n] - R3[n]) );
20
21         popravi_tmp(TMP, mnozica, n, D); // popravi trenutnega
               posameznika
22     }
23     n = (n+1)%D;
24 }
25 Position2tour(tmp, TMP, D); // DELUJE S POPRAVLJANJEM CIKLA
26
27 if (mRandom() < 0.5)
28     check_same(tmp, D, NP); // pogleda, ali se razlikuje od vseh
           posameznikov
```

Strategijo zaključimo s klicem funkcije *Position2Tour* (Algoritem 4), ki cikel pretvori nazaj v osnovno predstavitev posameznika ter s preverjanjem raznolikosti posameznika v populaciji (Algoritem 6).

5.2.3 Strategija *Xing*

Strategijo, ki smo jo poimenovali *Xing*, smo delno izvzeli iz članka [34]. Avtorji v tem članku rešujejo genetske algoritme za optimizacijske probleme. V članku smo našli takšni operaciji križanja in mutiranja, ob katerih smo predvidevali, da bi lahko izboljšali reševanje našega problema, zato smo se odločili, da ju bomo implementirali tudi v našem algoritmu kot eno izmed možnih kombinacij za reševanje nesimetričnega trgovskega potnika.

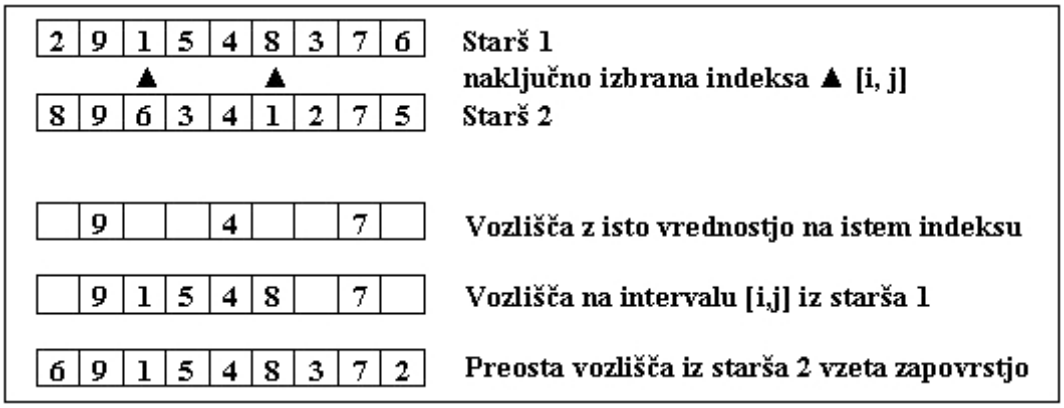
Strategijo smo implemenetirali tako, da smo nad posamezniki izvajali operacije križanja, mutacije in selekcije. Pred selekcijo smo posameznike popravili s hevrističnima algoritmoma *RAI* in/ali *OR-opt*.

Križanje

Operator križanja kombinira gene dveh staršev in ustvari potomca, tako da le-ta podeduje nabor gradnikov iz obeh staršev.

Postopek križanja, ki ga prikazuje slika 5.2, sta predlagala Wand in Wu [32]:

1. Naključno izberemo indeska i in j v ciklu (velja $i < j$).
2. V nov prazen cikel (v sina) prekopiramo vsa vozlišča, ki imajo isto vrednost na istem indeksu pri obeh starših.
3. Na prazna mesta v sinu prekopiramo vsa vozlišča iz prvega starša, ki se nahajajo na intervalu $[i, j]$.
4. Preostala prazna vozlišča prekopiramo iz drugega starša po vrstnem redu.



Slika 5.2: Ilustracija križanja po metodi Wand in Wu [32]

Algoritem 10 Funkcija križanja za trgovskega potnika

```
1  /*
2  * T[] ... izhodni posameznik (sin)
3  * R1[] ... 1. vhodni posameznik (1. stars)
4  * R2[] ... 2. vhodni posamezni (2. stars)
5  * n ... velikost posameznika
6  */
7  void Crossover(int T[], int R1[], int R2[], int n)
8  {
9      int mnozica[MAX_N];
10     int i=0;
11     int f1, f2;
12     /* najdemo dva indeksa */
13     do
14     {
15         f1 = rand()%n;
16         f2 = rand()%n;
17     } while (f1==f2);
18
19     int s_meja, z_meja;
20     if (f1 < f2)
21     {
22         s_meja = f1;
23         z_meja = f2;
24     }
25     else
26     {
27         z_meja = f1;
28         s_meja = f2;
29     }
30     for (i=0; i<n; i++)
31     {
32         T[i]=-1;
33         mnozica[i]=0;
34         if (R1[i] == R2[i] || (i>=s_meja && i<=z_meja))
35         {
36             mnozica[i]=1;
37             T[i] = R1[i];
38         }
39     }
40     int j, k;
41     int nasel=0;
42     for (i=0; i<n; i++)
43     {
44         nasel=0;
45         for (j=0; j<n; j++)
46         {
47             if (R2[i] == T[j])
48             {
49                 nasel=1;
50                 break;
51             }
52         }
53         if (nasel==0)
54         {
55             for (k=0; k<n; k++)
56             {
57                 if (mnozica[k]==0)
58                 {
59                     mnozica[k]=1;
60                     T[k]=R2[i];
61                     break;
62                 }
63             }
64         }
65     }
66 }
```

Mutacija

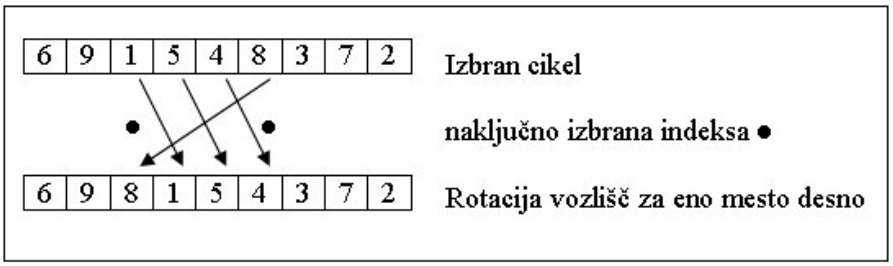
Mutacija vnaša v populacijo raznolikost in s tem omogoča algoritmom, da uspešneje preiskujejo prostor rešitve. Operator mutacije je potrebno definirati za vsak problem posebej. Prav to smo storili tudi mi, saj smo iz članka [34] prevzeli idejo operatorja mutacije ter ga priredili za naš problem.

Algoritem 11 Funkcija mutacije za trgovskega potnika

```
1  /*
2  * T[] ... izhodni posameznik (sin)
3  * R1[] ... vhodni posameznik (stars)
4  * n ... velikost posameznika
5  */
6  void Mutation(int T[], int R1[], int n)
7  {
8      int i=0;
9      int f1, f2;
10     do
11     {
12         f1 = rand()%n;
13         f2 = rand()%n;
14     } while (f1==f2);
15
16     int s_meja, z_meja;
17     if (f1 < f2)
18     {
19         s_meja = f1;
20         z_meja = f2;
21     }
22     else
23     {
24         z_meja = f1;
25         s_meja = f2;
26     }
27     for (i=0; i<n; i++)
28     {
29         if (i>s_meja && i<=z_meja)
30             T[i]=R1[i-1];
31         else if (i==s_meja)
32             T[i]=R1[z_meja];
33         else
34             T[i]=R1[i];
35     }
36 }
```

Z mutacijo smo se predvsem želeli približati optimalni rešitvi problema. Uporabili smo tako imenovano "rotirajočo" mutacijo (ang. *shift-change*) [22]. Njen postopek, ki je prikazan na sliki 5.3, je sledeč:

1. Naključno izberemo indeksa poti izmed vseh vozlišč v ciklu.
2. Rotiramo vsa vozlišča za en indeks (začenši pri levem izbranem) proti desni.
3. Zadnjega kopiramo na prvo mesto.



Slika 5.3: Ilustracija operatorja mutacije [22]

5.3 Algoritmi *DEATSP*

V diplomskem delu smo raziskovali, katere strategije izbrati za iskanje dobrih rešitev. Razvili smo več različnih algoritmov, ki nam na različne načine pomagajo k dobrim rešitvam. Algoritme smo razvili tako, da smo kombinirali hevristična algoritma *RAI* in *OR-opt* ter algoritem diferencialne evolucije.

V nadaljevanju pri opisu algoritmov prikazujemo le jedro algoritmov, ki smo jih kombinirali.

5.3.1 Osnovni algoritem *DE* za trgovskega potnika

Reševanje problema nesimetričnega trgovskega potnika z diferencialno evolucijo po literaturi skorajda ne zasledimo. Zato smo se odločili idejo razviti po svoje, z znanjem osnov reševanja optimizacijskih problemov. Funkcionalnost algoritma smo pripravili tako, da smo priredili osnovni algoritem *DE* [28]. Najprej smo naključno izbrali tri posameznike. Iz njih smo tvorili novega posameznika po principu ene izmed strategij (5.2.1, 5.2.2 ali 5.2.3). Nato smo s klicem funkcije *Cena* (glej Algoritem 5) ocenili posameznika (izračunali smo ceno cikla) in če je posameznik bil boljši, smo ga vključili v populacijo, sicer pa smo ga zavrgli.

5.3.2 Algoritem *DE+RAI*

Algoritem, ki smo ga poimenovali *DE+RAI*, je sestavljen iz dela diferencialne evolucije in hevrističnega algoritma *RAI*.

Algoritem 12 Algoritem *DE+RAI*

```
1  /*
2       mRandom() ... nakljucno stevilo na intervalu (0, 1]
3       param_m_random ... doloceno stevilo na intervalu (0, 1]
4       tmp ... novo generiran posameznik (alg. DE)
5  */
6  if (mRandom() < param_m_random) // ce izraz pravilen => RAI, sicer DE
7  {
8       assignd(D,tmp,(*pold)[i]); // prepisi tmp s starim
9       trial_cost = Try_Cena(G, tmp, D);  // alg. RAI
10 }
11 else
12 {
13      trial_cost = Cena(G, tmp, D);    // samo oceni (alg. DE)
14 }
```

Kot lahko vidimo iz programske kode algoritma 12, se za vsakega posameznika ob določeni verjetnosti (parameter *param_m_random*) požene algoritem *RAI* oziroma algoritem *DE*.

5.3.3 Algoritem *DE+OR-opt*

Kombinacija (algoritem 13) med algoritmom diferencialne evolucije in hevrističnega algoritma *OR-opt*, ki predvsem izboljša rešitev, kadar smo že blizu rešitve, se izraža v tem, da smo diferencialno evolucijo prisilili, da opravi večino dela, medtem ko algoritem *OR-opt* nastopi šele takrat, kadar smo predelali že 70% generacij in kadar je naključno izbrana vrednost pod mejo vrednosti, ki jo določimo s parametrom *param_m_random*.

Algoritem 13 Algoritem *DE+OR-opt*

```
1  /*
2       mRandom() ... nakljucno stevilo na intervalu (0, 1]
3       param_m_random ... doloceno stevilo na intervalu (0, 1]
4       gen ... trenutna generacija
5       genmax ... stevilo vseh generacij
6  */
7  if (gen > 0.7*genmax && mRandom() < param_m_random) // ce izraz
       pravilen => or-OPT, sicer DE
8  {
9       assignd(D,tmp,(*pold)[i]); // prepisi tmp s starim
10      trial_cost = or_opt(G, tmp, D);
11 }
12 else
13 {
14      trial_cost = Cena(G, tmp, D);    // samo oceni
15 }
```

5.3.4 Algoritem *DE+RAI+OR-opt mod 3*

Med algoritmi *DEATSP* smo ustvarili tudi takšnega, ki mu določimo enakomerno porazdelitev dela pri kreiranju posameznikov. Sestavljajo ga trije algoritmi:

- osnovni algoritem *DE*,
- hevristični algoritem *RAI* in
- hevristični algoritem *OR-opt*.

Algoritem 14 Algoritem *DE+RAI+OR-opt mod 3*

```
1  /*
2       Algoritem deluje po 1/3 porazdelitvi dela (mod 3)
3  */
4  if (i%3==2)
5  {
6       assignd(D,tmp,(*pold)[i]); // prepisi posameznika s starim
7       trial_cost = or_opt(G, tmp, D); // OR-opt
8  }
9  else if (i%3==1)
10 {
11      assignd(D,tmp,(*pold)[i]); // prepisi posameznika s starim
12      trial_cost = Try_Cena(G, tmp, D);  // RAI
13 }
14 else
15 {
16      trial_cost = Cena(G, tmp, D);    // osnovni DE
17 }
```

5.3.5 Algoritem *DE+RAI+OR-opt-V2*

Algoritem *DE+RAI+OR-opt-V2* je predvsem zanimiv zaradi svoje strukture. Zgrajen je tako, da prvi tretjini vseh generacij delujeta le algoritem *DE* in *RAI*, ki se izmenjujeta po verjetnosti, določeni s parametrom *param_m_random*. V drugi tretjini vseh generacij nastopi le algoritem *DE*. Zadnjo tretjino pa prevzame algoritem *OR-opt*, ki je znan po tem, da zna zelo dobro konvergirati k optimalni rešitev, če je že blizu le-te.

Algoritem 15 Algoritem *DE+RAI+OR-opt-V2*

```
1  /*
2      mRandom() ... nakljucno stevilo na intervalu [0, 1)
3      param_m_random ... doloceno stevilo na intervalu [0, 1)
4      gen ... trenutna generacija
5      genmax ... stevilo vseh generacij
6  */
7  if (mRandom() < param_m_random && gen<0.3*genmax) // ce izraz pravilen
       => RAI
8  {
9      assignd(D,tmp,(*pold)[i]); // prepisi posameznika s starim
10     trial_cost = Try_Cena(G, tmp, D);
11 }
12 else if (gen>0.7*genmax)
13 {
14     assignd(D,tmp,(*pold)[i]); // prepisi posameznika s starim
15     trial_cost = or_opt(G, tmp, D); // alg. OR-opt
16 }
17 else
18 {
19     trial_cost = Cena(G, tmp, D);  // alg. DE
20 }
```

5.3.6 Algoritem *DE+RAI+OR-opt-V3*

Algoritem, ki smo ga poimenovali *DE+RAI+OR-opt-V3*, je načrtovan in implementiran tako, da poiskušamo boljše posameznike še izboljševati z izmenjavo algoritmov *DE*, *RAI* in *OR-opt*. Odločamo se med naslednjimi koraki:

- če je trenutni posameznik najboljši v prejšnji generaciji, nad njim izvedemo algoritem *RAI* (Algoritem 4.3); v primeru, da smo izvedli polovico generacij, še nad njim izvedemo algoritem *OR-opt*,
- če je smo v evoluciji izvedli že polovico generacij in je verjetnost izbire manjša od 10%, potem nad posameznikom izvedemo algoritem *OR-opt*,
- sicer izvedemo osnovni algoritem *DE*.

Algoritem 16 Algoritem *DE+RAI+OR-opt-V3*

```
1  /*
2      mRandom() ... nakljucno stevilo na intervalu [0, 1)
3      gen ... trenutna generacija
4      genmax ... stevilo vseh generacij
5      imin ... indeks najboljsega posameznika
6  */
7  if (i==imin || mRandom() < 0.10)
8  {
9      assignd(D,tmp,(*pold)[i]);
10     trial_cost = Try_Cena(G, tmp, D); // alg. RAI
11     if (i==imin && gen > 0.5*genmax)
12     {
13         trial_cost = or_opt(G, tmp, D); // alg. OR-opt
14     }
15 }
16 else if (gen > 0.5*genmax && mRandom() < 0.10)
17 {
18     trial_cost = or_opt(G, tmp, D); // alg. OR-opt
19 }
20 else
21 {
22     trial_cost = Cena(G, tmp, D);  // alg. DE
23 }
```

Poglavje 6

Rezultati

V tem poglavju so opisani in prikazani rezultati, ki smo jih dobili pri iskanju rešitve za problem nesimetričnega trgovskega potnika. Algoritme smo izvajali na osebnem računalniku in preiskušali na grafih iz knjižnice TSPLIB [25]. V tej knjižjnici so za grafe, ki predstavljajo probleme nesimetričnega trgovskega potnika, znane optimalne rešitve, katere pa predstavljajo dobro oceno za delovanje naših algoritmov *DEATSP*.

Rešitve so bile zelo različne glede na kombinacije strategij in algoritmov, ki so opisani v poglavju 5. Raziskovali smo jih tako, da smo se želeli čim bolj približali optimalnim rešitvam.

Parametre za reševanje ATSP z diferencialno evolucijo in s hevrističnimi algoritmi smo nastavili na naslednji način:

- faktor skaliranja: $F = 0.1 + mRandom() * 0.9$, kjer funkcija $mRandom$ vrne naključno število na intervalu $[0, 1)$,
- krmilni parameter križanja: $CR = 0.1 + mRandom() * 0.8$,
- krmilni parameter izmenjave algoritmov: $param_m_random = 0.5$,
- število generacij $genmax = D * D$, kjer je D (oziroma n) dimenzija grafa,
- velikost populacije $NP = 100$.

Ti parametri so veljali skozi vse evaluacije nad vsemi kombinacijami algoritmov in jih nismo spreminjali pri nobeni od različic algoritmov.

Rezultati, ki smo jih dobili pri poiskusih, so prikazani v tabelah v tem poglavju. Drugi in tretji stolpec prikazujeta ime grafa ATSP ter število vozlišč. Četrti stolpec, ki je označen z besedo '*opt.*', pa prikazuje najboljšo znano oziroma optimalno rešitev.

Pri naših raziskavah smo si kot glavni cilj zastavili, da algoritem naj poišče čim boljšo rešitev za dani graf. Tako nas najbolj zanima stolpec, označen z '*min*'. Opozorimo, da stolpec '*povp.*' v tabelah 6.1–6.12 (v poglavjih 6.1–6.4) prikazuje povprečno vrednost posameznih rešitev znotraj populacije v zadnji generaciji, stolpec '*std.dev*' pa pripadajočo standardno deviacijo.

Naj še omenimo, da smo pri zadnjih štirih grafih: rbg323, rbg358, rbg358 in rbg358, poiskuse iskanja rešitve zaganjali le n-krat (n - število vseh generacij), medtem ko smo pri vseh ostalih rešitev poskušali poiskati n^2-krat.

6.1 Algoritem *DE+RAI*

V tem poglavju opisujemo rezultate, ki smo jih dobili z algoritmom *DE+RAI* v kombinaciji s strategijami:

- *DE+RAI* s strategijo *DE/rand/1/exp*,
- *DE+RAI* s strategijo *DE/rand/1/bin* in
- *DE+RAI* s strategijo *Xing*.

Strategije so opisane v poglavju 5.

6.1.1 Algoritem *DE+RAI* strategijo *DE/rand/1/exp*

Algoritem *DE+RAI* smo povezali s strategijo *DE/rand/1/exp*, kjer je imel algoritem *RAI* 5% verjetnosti, da je reševal problem, ostalo delo pa je opravil algoritem *DE*.

Rešitve algoritma *DE+RAI* s strategijo *DE/rand/1/exp* nam prikazuje tabela 6.1. Kot lahko iz nje razberemo, je algoritem našel nekatere optimalne rešitve, pri nekaterih grafih pa je prišlo do manjših odstopanj. Če izpustimo zadnje štiri grafe, lahko vidimo, da je največje odstopanje pri grafu *ft70*, in sicer približno 1%.

Algoritem je uspel najti optimalno rešitev v 10-ih primerih, medtem ko mu v 13-ih primerih to ni uspelo. Ker so v tabeli grafi urejeni po številu vozlišč v grafu, vidimo, da je algoritem uspešno rešil grafe do velikosti 65 vozlišč, z izjemo grafa *ry48p* z 48 vozlišči. Tukaj je algoritem našel rešitev z vrednostjo cikla 14429, optimalna rešitev pa je 14422, kar predstavlja 0,05% odstopanje od optimalne rešitve.

Pri *rgb* grafih sta grafa *rgb323* in *rgb358* predstavljala algoritmu težje iskanje, saj se rešitev od optimalne razlikuje za več ko 12%.

Tako lahko sklepamo, da smo že v začetku samega razvoja algoritmov prišli do dokaj dobrih rezultatov, kjer so nekateri bili celo optimalni, nekateri pa so odstopali z zanemarljivo majhno vrednostjo.

Tabela 6.1: Rezultati algoritma *DE+RAI* s strategijo *DE/rand/1/exp*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1334,00	3,73	35,00
3	ftv35	36	1473	1473	0,00	1508,00	2,38	25,00
4	ftv38	39	1530	1530	0,00	1562,00	2,09	23,00
5	p43	43	5620	5620	0,00	5622,00	0,04	2,00
6	ftv44	45	1613	1613	0,00	1663,00	3,10	19,00
7	ftv47	48	1776	1776	0,00	1791,00	0,84	15,00
8	ry48p	48	14422	14429	0,05	14707,00	1,98	135,00
9	ft53	53	6905	6905	0,00	7069,00	2,38	141,00
10	ftv55	56	1608	1608	0,00	1648,00	2,49	21,00
11	ftv64	65	1839	1839	0,00	1893,00	2,94	25,00
12	ft70	70	38673	39027	0,92	39609,00	2,42	229,00
13	ftv70	71	1950	1962	0,62	1997,00	2,41	20,00
14	ftv90	91	1579	1581	0,13	1603,00	1,52	15,00
15	kro124p	100	36230	36241	0,03	36983,00	2,08	643,00
16	ftv100	101	1788	1790	0,11	1817,00	1,62	18,00
17	ftv110	111	1958	1960	0,10	1996,00	1,94	31,00
18	ftv120	121	2166	2168	0,09	2220,00	2,49	29,00
19	ftv130	131	2307	2311	0,17	2371,00	2,77	34,00
20	ftv140	141	2420	2422	0,08	2486,00	2,73	48,00
21	ftv150	151	2611	2613	0,08	2709,00	3,75	57,00
22	ftv160	161	2683	2686	0,11	2782,00	3,69	49,00
23	ftv170	171	2755	2770	0,54	2866,00	4,03	48,00
24	rbg323	323	1326	1488	12,22	1550,00	16,89	28,00
25	rbg358	358	1163	1344	15,56	1417,00	21,84	34,00
26	rbg403	403	2465	2528	2,56	2566,00	4,10	18,00
27	rbg443	443	2720	2801	2,98	2851,00	4,82	22,00

6.1.2 Algoritem *DE+RAI* s strategijo *DE/rand/1/bin*

Algoritem *DE+RAI* smo prav tako povezali s strategijo *DE/rand/1/bin*. Rezultate, ki smo jih dobili z izvajanjem algoritma, prikazuje tabela 6.2.

Algoritem je našel (če zopet izvzamemo zadnje štiri grafe) optimalno rešitev v 13-ih primerih, v 10-ih primerih pa optimalne rešitve ni našel. Največje odstopanje od optimalne rešitve je bilo pri grafu *ft70* in sicer 0,66%. V primerjavi s tabelo 6.1 opazimo, da algoritem *DE+RAI* s strategijo *DE/rand/1/bin* zopet ni našel optimalne rešitve pri grafu *ry48p*.

Algoritem ni našel rešitve pri grafu *ftv64*, uspešno pa je rešil grafe *ftv90*, *ftv100*, *ftv110* in *ftv120*, kjer ima zadnje omenjeni graf 121 vozlišč.

Pri grafih *rgb* je algoritem *DE+RAI* s strategijo *DE/rand/1/bin* dosegel podobne rezutate v primerjavi z algoritmom *DE+RAI* s strategijo *DE/rand/1/exp* (tabela 6.1).

Tabela 6.2: Rezultati algoritma *DE+RAI* s strategijo *DE/rand/1/bin*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1324,00	2,95	38,00
3	ftv35	36	1473	1473	0,00	1500,00	1,83	23,00
4	ftv38	39	1530	1530	0,00	1566,00	2,35	22,00
5	p43	43	5620	5620	0,00	5623,00	0,05	2,00
6	ftv44	45	1613	1613	0,00	1659,00	2,85	22,00
7	ftv47	48	1776	1776	0,00	1791,00	0,84	15,00
8	ry48p	48	14422	14495	0,51	14692,00	1,87	129,00
9	ft53	53	6905	6905	0,00	7078,00	2,51	148,00
10	ftv55	56	1608	1608	0,00	1648,00	2,49	19,00
11	ftv64	65	1839	1850	0,60	1893,00	2,94	27,00
12	ft70	70	38673	38930	0,66	39596,00	2,39	268,00
13	ftv70	71	1950	1954	0,21	1996,00	2,36	21,00
14	ftv90	91	1579	1579	0,00	1600,00	1,33	20,00
15	kro124p	100	36230	36241	0,03	36821,00	1,63	652,00
16	ftv100	101	1788	1788	0,00	1810,00	1,23	16,00
17	ftv110	111	1958	1958	0,00	1996,00	1,94	33,00
18	ftv120	121	2166	2166	0,00	2223,00	2,63	33,00
19	ftv130	131	2307	2310	0,13	2364,00	2,47	36,00
20	ftv140	141	2420	2422	0,08	2488,00	2,81	48,00
21	ftv150	151	2611	2613	0,08	2712,00	3,87	54,00
22	ftv160	161	2683	2688	0,19	2775,00	3,43	41,00
23	ftv170	171	2755	2767	0,44	2869,00	4,14	47,00
24	rbg323	323	1326	1486	12,07	1545,00	16,52	29,00
25	rbg358	358	1163	1340	15,22	1421,00	22,18	37,00
26	rbg403	403	2465	2527	2,52	2568,00	4,18	18,00
27	rbg443	443	2720	2796	2,79	2853,00	4,89	21,00

6.1.3 Algoritem *DE+RAI* s strategijo *Xing*

Rezultate algoritma *DE+RAI*, ki smo ga kombinirali s strategijo *Xing*, prikazuje tabela 6.3. Algoritem je našel optimalno rešitev v 9-ih primerih izmed 23-ih. Če to primerjamo s predhodnima algoritmoma (isti algoritem z uporabo strategij *DE/rand/1/exp* in *DE/rand/1/bin*), opazimo, da sta predhodna algoritma večkrat našla optimalno rešitev.

Algoritem *DE+RAI* s strategijo *Xing* je optimalno rešil grafe do velikosti 45 vozlišč ter grafe *ftv70*, *ftv90* in *ftv160*. Graf *ftv160* ima 161 vozlišč. Pri grafih *ftv70* in *ftv160* pa predhodna algoritma nista našla optimalih rešitev.

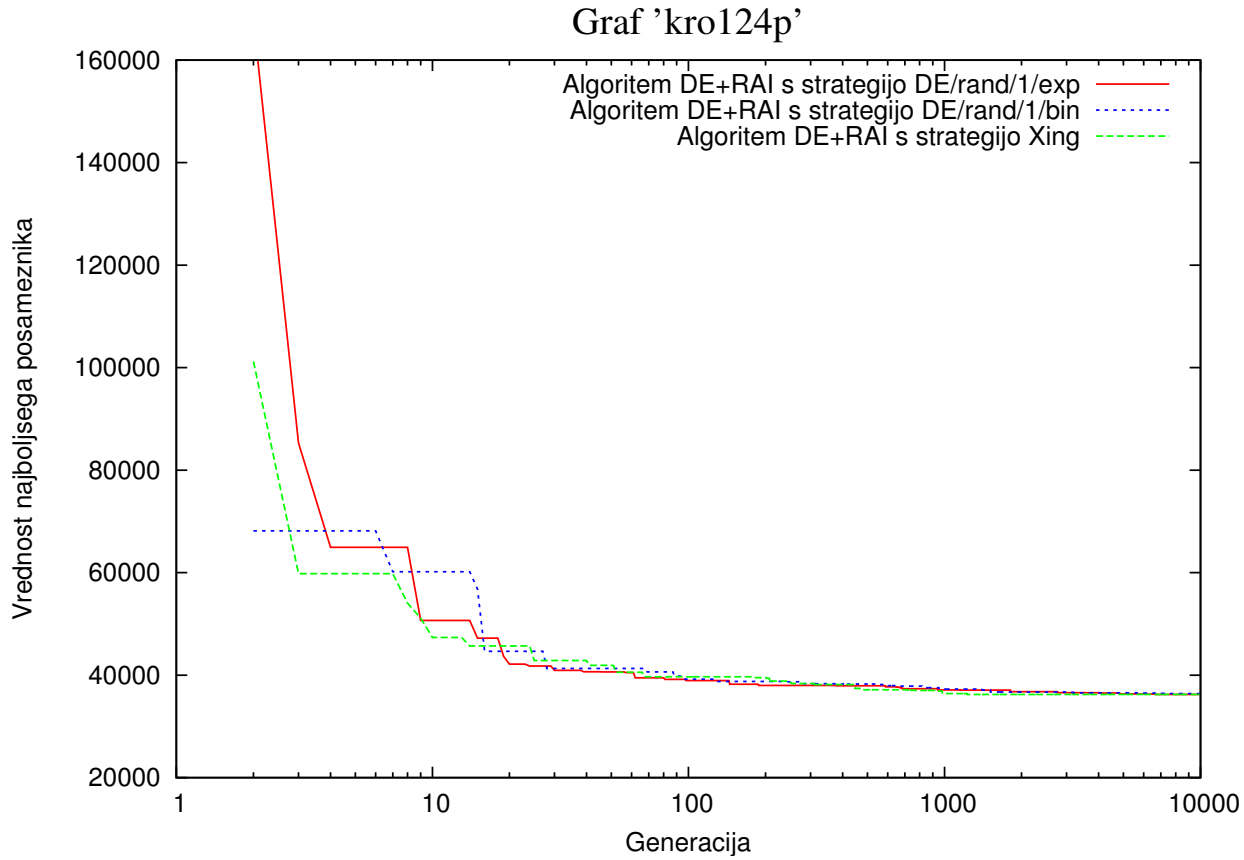
Tabela 6.3: Rezultati algoritma *DE+RAI* s strategijo *Xing*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1286,00	0,00	1,00
3	ftv35	36	1473	1473	0,00	1483,00	0,68	12,00
4	ftv38	39	1530	1530	0,00	1534,00	0,26	9,00
5	p43	43	5620	5620	0,00	5620,00	0,00	0,00
6	ftv44	45	1613	1613	0,00	1617,00	0,25	12,00
7	ftv47	48	1776	1777	0,06	1777,00	0,06	0,00
8	ry48p	48	14422	14507	0,59	14596,00	1,21	53,00
9	ft53	53	6905	6915	0,14	6915,00	0,14	0,00
10	ftv55	56	1608	1623	0,93	1623,00	0,93	0,00
11	ftv64	65	1839	1855	0,87	1859,00	1,09	3,00
12	ft70	70	38673	38929	0,66	38929,00	0,66	0,00
13	ftv70	71	1950	1950	0,00	1952,00	0,10	4,00
14	ftv90	91	1579	1579	0,00	1579,00	0,00	0,00
15	kro124p	100	36230	36241	0,03	36241,00	0,03	0,00
16	ftv100	101	1788	1790	0,11	1791,00	0,17	1,00
17	ftv110	111	1958	1960	0,10	1960,00	0,10	0,00
18	ftv120	121	2166	2168	0,09	2168,00	0,09	0,00
19	ftv130	131	2307	2309	0,09	2309,00	0,09	0,00
20	ftv140	141	2420	2426	0,25	2426,00	0,25	0,00
21	ftv150	151	2611	2613	0,08	2613,00	0,08	0,00
22	ftv160	161	2683	2683	0,00	2683,00	0,00	0,00
23	ftv170	171	2755	2780	0,91	2780,00	0,91	0,00
24	rbg323	323	1326	1511	13,95	1562,00	17,80	25,00
25	rbg358	358	1163	1342	15,39	1433,00	23,22	32,00
26	rbg403	403	2465	2521	2,27	2578,00	4,58	21,00
27	rbg443	443	2720	2800	2,94	2857,00	5,04	22,00

Pri grafih *rgb* je algoritem je našel podobne rezultate kot prehodna algoritma.

Če pogledamo stolpec '*povp.*', ki prikazuje povprečno vrednost posameznika v populaciji, opazimo, da so pri algoritmu *DE+RAI* s strategijo *Xing* posamezniki v povprečju boljši. Hkrati pa to pomeni, da imajo posamezniki manjšo standardno deviacijo ob zaključku procesa optimizacije.

Delovanje algoritmov *DE+RAI* v kombinaciji s strategijami *DE/rand/1/exp*, *DE/rand/1/bin* in *Xing* za graf *kro124p* prikazuje slika 6.1. Na sliki, kjer smo za x-os uporabili logaritemsko lestvico, lahko vidimo, kako so se funkcijske vrednosti najboljšega posameznika algoritmov spreminjale med evolucijskim procesom. Vsi algoritmi so našli enako rešitev, ki je od optimalne rešitve odstopala za 0,03%.



Slika 6.1: Reševanje grafa *kro124p* z algoritmom *DE+RAI*

6.2 Algoritem *DE+OR-opt*

Tukaj opisujemo rezultate algoritma *DE+OR-opt* v kombinaciji s strategijami:

- *DE+OR-opt* s strategijo *DE/rand/1/exp*,
- *DE+OR-opt* s strategijo *DE/rand/1/bin* in
- *DE+OR-opt* s strategijo *Xing*.

V primerjavi s poglavjem 6.1 tukaj uporabljamo iste strategije, razlika je v algoritmih.

6.2.1 Algoritem *DE+OR-opt* s strategijo *DE/rand/1/exp*

Tabela 6.4 prikazuje rezultate algoritma *DE+OR-opt* s strategijo *DE/rand/1/exp*. Le-ta je našel optimalno rešitev 11-krat, 12-krat pa optimalne rešitve ni našel.

Najdena rešitev se je od optimalne rešitve razlikovala za več kot 1% pri sedmih grafih (od *ftv110* do *ftv170*). Kot zanimivost lakho ugotovimo, da je algoritem optimalno rešil grafa *ry48p* in *kro124p*, katerih optimalnih rešitev ni našel noben od algoritmov iz poglavja 6.1.

Poglejmo še rezultate na grafih *rgb*. Algoritem je najboljšo rešitev našel na grafu *rgb403*, ki znaša 2467, kar pomeni, da se od optimalne (2465) razlikuje za 0.08%. Najslabšo rešitev pa je našel pri grafu *rgb358*, kjer je dolžina cilka za trgovskega potnika znašala 1227, to je 5,5% slabše od optimalne rešitve.

6.2.2 Algoritem *DE+OR-opt* s strategijo *DE/rand/1/bin*

Rezultati rešitev grafov z algoritmom *DE+OR-opt* v kombinaciji s strategijo *DE/rand/1/bin* so prikazani v tabeli 6.5. Algoritem je optimalno rešitev našel v 7-ih primerih, v 16-ih primerih pa mu optimalne rešitve ni uspelo najti.

Algoritem se je predvsem slabo odrazil pri grafih, ki imajo več kot 70 vozlišč. To se kaže pri grafih od *ft70* do *ftv170*, kjer se je najdena rešitev od optimalne razlikovala za več kot 1,13%. Pri grafih od *ftv120* do *ftv170* je ta razlika bila celo več kot 10%.

Tabela 6.4: Rezultati algoritma *DE+OR-opt* s strategijo *DE/rand/1/exp*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	1,00
2	ftv33	34	1286	1286	0,00	1341,00	4,28	30,00
3	ftv35	36	1473	1473	0,00	1499,00	1,77	20,00
4	ftv38	39	1530	1530	0,00	1557,00	1,76	22,00
5	p43	43	5620	5620	0,00	5625,00	0,09	3,00
6	ftv44	45	1613	1613	0,00	1665,00	3,22	23,00
7	ftv47	48	1776	1776	0,00	1801,00	1,41	18,00
8	ry48p	48	14422	14422	0,00	14629,00	1,44	88,00
9	ft53	53	6905	6915	0,14	7055,00	2,17	87,00
10	ftv55	56	1608	1608	0,00	1659,00	3,17	27,00
11	ftv64	65	1839	1842	0,16	1890,00	2,77	29,00
12	ft70	70	38673	38878	0,53	39305,00	1,63	229,00
13	ftv70	71	1950	1950	0,00	2006,00	2,87	37,00
14	ftv90	91	1579	1585	0,38	1676,00	6,14	46,00
15	kro124p	100	36230	36230	0,00	37022,00	2,19	572,00
16	ftv100	101	1788	1799	0,62	1900,00	6,26	56,00
17	ftv110	111	1958	1983	1,28	2118,00	8,17	59,00
18	ftv120	121	2166	2199	1,52	2346,00	8,31	74,00
19	ftv130	131	2307	2379	3,12	2499,00	8,32	65,00
20	ftv140	141	2420	2502	3,39	2646,00	9,34	72,00
21	ftv150	151	2611	2692	3,10	2905,00	11,26	88,00
22	ftv160	161	2683	2819	5,07	2995,00	11,63	76,00
23	ftv170	171	2755	2831	2,76	3097,00	12,41	100,00
24	rbg323	323	1326	1362	2,71	1405,00	5,96	17,00
25	rbg358	358	1163	1227	5,50	1316,00	13,16	504,00
26	rbg403	403	2465	2467	0,08	2530,00	2,64	451,00
27	rbg443	443	2720	2727	0,26	2749,00	1,07	10,00

V primerjavi z algoritmom *DE+OR-opt* v kombinaciji s strategijo *DE/rand/1/exp* iz prejšnjega poglavja, je ta algoritem deloval precej slabše, podobne rešitve lahko najdemo le pri grafih, ki imajo število vozlišč manjše kot 55.

Pri grafih *rbg* lahko iz tabele 6.5 razberemo, da algoritem ni uspel najti optimalne rešitve. Algoritem je najslabše rešil graf *rgb358*, kjer je njegova rešitev znašala 1238, kar pomeni 6,45% razlike od optimalne rešitve, ki je 1163. Najboljše izmed *rgb* grafov, pa je algoritem rešil graf *rgb443*, ki ima 443 vozlišč. Rešitev, ki jo je algoritem našel, znaša 2726, kar je za 0,22% slabše kot optimalna rešitev, kjer je velikost cilka enaka 2720.

Tabela 6.5: Rezultati algoritma *DE+OR-opt* s strategijo *DE/rand/1/bin*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1337,00	3,97	28,00
3	ftv35	36	1473	1473	0,00	1498,00	1,70	18,00
4	ftv38	39	1530	1530	0,00	1560,00	1,96	16,00
5	p43	43	5620	5622	0,04	5627,00	0,12	3,00
6	ftv44	45	1613	1623	0,62	1670,00	3,53	21,00
7	ftv47	48	1776	1776	0,00	1810,00	1,91	22,00
8	ry48p	48	14422	14429	0,05	14659,00	1,64	98,00
9	ft53	53	6905	6905	0,00	7112,00	3,00	105,00
10	ftv55	56	1608	1608	0,00	1671,00	3,92	27,00
11	ftv64	65	1839	1854	0,82	1909,00	3,81	28,00
12	ft70	70	38673	39111	1,13	39574,00	2,33	179,00
13	ftv70	71	1950	1975	1,28	2045,00	4,87	34,00
14	ftv90	91	1579	1644	4,12	1762,00	11,59	39,00
15	kro124p	100	36230	36845	1,70	38278,00	5,65	534,00
16	ftv100	101	1788	1919	7,33	2016,00	12,75	40,00
17	ftv110	111	1958	2105	7,51	2258,00	15,32	50,00
18	ftv120	121	2166	2386	10,16	2523,00	16,48	51,00
19	ftv130	131	2307	2558	10,88	2724,00	18,08	60,00
20	ftv140	141	2420	2712	12,07	2898,00	19,75	62,00
21	ftv150	151	2611	2989	14,48	3184,00	21,95	68,00
22	ftv160	161	2683	3109	15,88	3305,00	23,18	64,00
23	ftv170	171	2755	3286	19,27	3463,00	25,70	56,00
24	rbg323	323	1326	1385	4,45	1414,00	6,64	13,00
25	rbg358	358	1163	1238	6,45	1277,00	9,80	17,00
26	rbg403	403	2465	2471	0,24	2488,00	0,93	8,00
27	rbg443	443	2720	2726	0,22	2756,00	1,32	10,00

6.2.3 Algoritem *DE+OR-opt* s strategijo *Xing*

V tem poglavju prikazujemo in opisujemo rezultate reševanja problema nesimetričnega trgovskega potnika, ki ga je reševal algoritem *DE+OR-opt* s strategijo *Xing*. Rezultati so prikazani v tabeli 6.6. Algoritem je uspešno rešil 13 grafov iz knjižnice *TSPLIB*, kar pomeni, da je našel njihovo optimalno rešitev, medtem ko mu za 10 grafov to ni uspelo.

Za razliko od rezultatov algoritma *DE+OR-opt* s strategijo *DE/rand/1/bin*, ki so prikazani v tabeli 6.5, je algoritem pri vseh grafih poiskal rešitve, ki so odstopale od optimalnih za manj kot 1%. To mu ni uspelo le pri grafu *ftv170*, kjer se je od optimalne rešitve (2755) najdena rešitev razlikovala za 2,54%, kar pomeni, da je

velikost cilka v grafu znašala 2825.

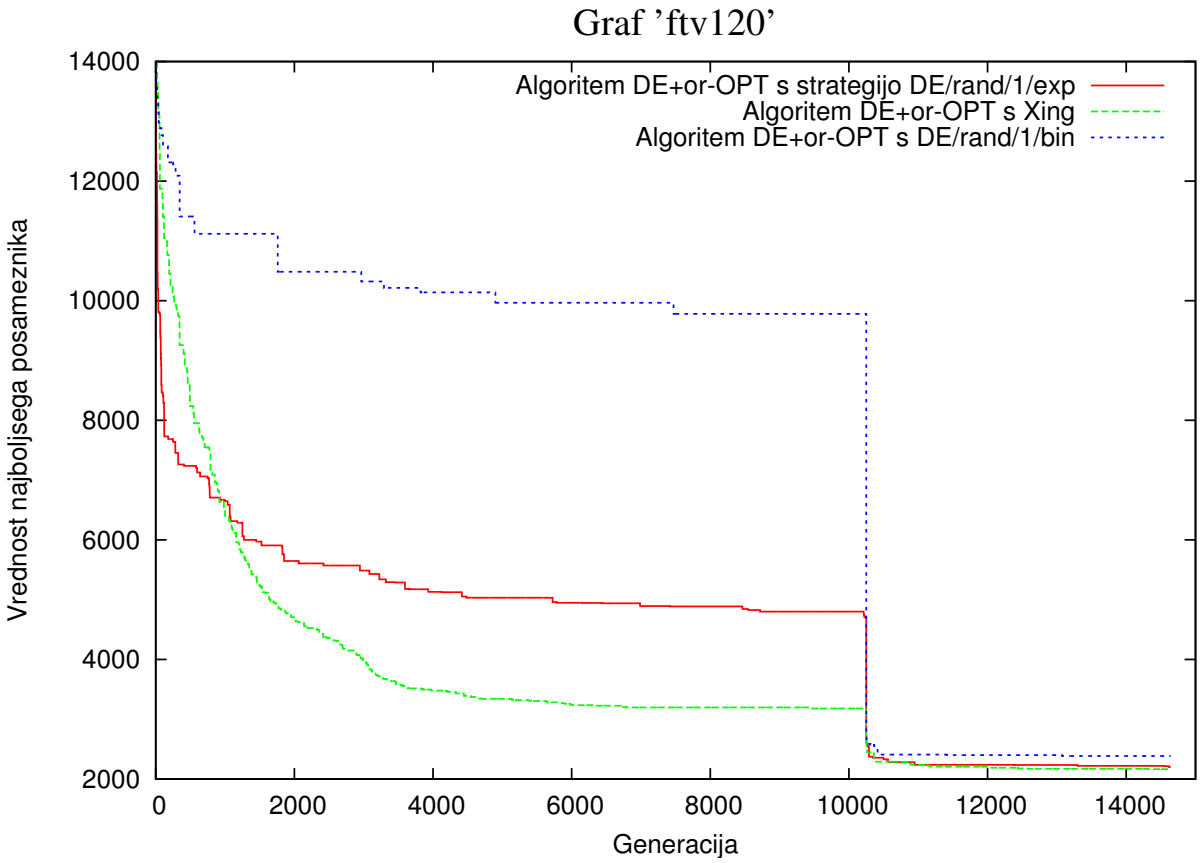
Glede na to, da je algoritem našel optimalno rešitev za graf *ftv120*, ki ima 120 vozlišč, je zanimiv rezultat pri grafu *ftv38*, ki ima le 38 vozlišč. Algoritem pri tem grafu ni uspel najti optimalne rešitve. Rešitev, ki jo je poiskal, je enaka 1532, kar pa znaša 0,13% odstopanja od optimalne.

Tabela 6.6: Rezultati algoritma *DE+OR-opt* s strategijo *Xing*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1293,00	0,54	20,00
3	ftv35	36	1473	1473	0,00	1474,00	0,07	5,00
4	ftv38	39	1530	1532	0,13	1536,00	0,39	6,00
5	p43	43	5620	5620	0,00	5620,00	0,00	0,00
6	ftv44	45	1613	1613	0,00	1622,00	0,56	6,00
7	ftv47	48	1776	1776	0,00	1777,00	0,06	3,00
8	ry48p	48	14422	14507	0,59	14563,00	0,98	43,00
9	ft53	53	6905	6905	0,00	6921,00	0,23	28,00
10	ftv55	56	1608	1608	0,00	1608,00	0,00	0,00
11	ftv64	65	1839	1839	0,00	1839,00	0,00	0,00
12	ft70	70	38673	38677	0,01	38875,00	0,52	96,00
13	ftv70	71	1950	1950	0,00	1950,00	0,00	1,00
14	ftv90	91	1579	1585	0,38	1585,00	0,38	0,00
15	kro124p	100	36230	36230	0,00	36230,00	0,00	0,00
16	ftv100	101	1788	1788	0,00	1788,00	0,00	1,00
17	ftv110	111	1958	1964	0,31	1964,00	0,31	0,00
18	ftv120	121	2166	2166	0,00	2176,00	0,46	7,00
19	ftv130	131	2307	2313	0,26	2322,00	0,65	11,00
20	ftv140	141	2420	2426	0,25	2453,00	1,36	12,00
21	ftv150	151	2611	2635	0,92	2636,00	0,96	3,00
22	ftv160	161	2683	2707	0,89	2707,00	0,89	1,00
23	ftv170	171	2755	2825	2,54	2834,00	2,87	4,00
24	rbg323	323	1326	1367	3,09	1394,00	5,13	15,00
25	rbg358	358	1163	1202	3,35	1240,00	6,62	17,00
26	rbg403	403	2465	2465	0,00	2476,00	0,45	7,00
27	rbg443	443	2720	2722	0,07	2736,00	0,59	7,00

Za *rgb* grafe lahko povemo, da je algoritem pri grafu *rbg403* našel optimalno rešitev, kjer velikost cilka trgovskega potnika znašala 2465. Za grafa *rbg358* in *rbg323* pa se je rešitev od optimale razlikovala za več kot 3%, natančneje za graf *rbg358* 3,35%, za graf *rbg323* pa 3,09%.

V primerjavi z rezultati algoritma *DE+OR-opt* s kombinacijo ene od strategij *DE/rand/1/exp* in *DE/rand/1/bin* iz prejšnjih poglavij, vidimo, da so posamezniki v stolpcu '*povp.*' v povprečju boljši, glede na najdene rešitve, ki jih je poiskal algoritem *DE+OR-opt* s strategijo *Xing*.



Slika 6.2: Reševanje grafa *ftv120* z algoritmom *DE+OR-opt*

Slika 6.2 prikazuje reševanje problema nesimetričnega trgovskega potnika z algoritmom *DE+OR-opt*, kjer smo uporabili vse tri kombinacije že omenjenih strategij. Dobro se da razbrati, da algoritem s strategijo *Xing* hitreje konvergira k rešitvi v primerjavi z algoritmom, ki uporablja strategijo *DE/rand/1/exp* oziroma *DE/rand/1/bin*.

Naj še omenimo, da je le algoritem *DE+OR-opt* s strategijo *Xing* pri grafu, ki je prikazan na sliki 6.2, našel optimalno rešitev. Algoritem *DE+OR-opt* s strategijo *DE/rand/1/bin* je pri reševanju tega problema imel največje odstopanje od optimalne rešitve, kar se na grafičnem prikazu tudi lepo vidi.

6.3 Algoritem *DE+RAI+OR-opt mod 3*

V tem poglavju opisujemo rezultate reševanja problema nesimetričnega trgovskega potnika z algoritmom *DE+RAI+OR-opt mod 3*, ki delujejo po pravilu enakomerne izmenjave algoritmov *DE*, *RAI* in *OR-opt*. Kombinacije algoritma *DE+RAI+OR-opt mod 3* smo tvorili na naslednje načine:

- *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp*,
- *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/bin* in
- *DE+RAI+OR-opt mod 3* s strategijo *Xing*.

6.3.1 Algoritem *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp*

Iz rezultatov, ki so prikazani v tabeli 6.7, lahko vidimo, da je algoritem *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp* uspešno našel optimalno rešitev v 11-ih primerih. V 12-ih primerih algoritmu to ni uspelo. Njegova 100% uspešnost iskanja optimalne rešitve je bila pri grafih, ki imajo do 47 vozlišč.

Algoritem ni uspel poiskati optimalne rešitve pri grafu *ft53* s 53-imi vozliči. Tukaj je našel rešitev cikla dolžine 6973, medtem ko je znana optimalna rešitev tega grafa enaka 6905, kar znaša 0,98% odstopanja. Največje odstopanje med grafi od optimalne rešitve je prišlo pri grafu *ftv170*, kjer je algoritem našel rešitev 2792, ki je za 1,34% slabša od optimalne.

Pri *rbg* grafih algoritem *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp* ni uspel posikati optimalne rešitve. Najboljše se je odrezal pri grafu *rgb403*, kjer je njegova razlika med dobljeno in optimalno rešitvijo znašala 0,61%. Najslabši rezultat izmed *rbg* grafov, pa je algoritem izračunal pri grafu *rbg358*, kjer je rešitev od optimalne odstopala 6,71%. To pomeni, da je rešitev cikla znašala 1241, medtem ko je optimalna rešitev 1163.

Iz stolpca '*povp.*' lahko vidimo, da so za grafe od *kro124p* naprej, posamezniki v populaciji bili povprečju za vsaj 60% slabši od najboljše rešitve, kar pomeni, da je bila konvergenca proti rešitvi počasna, to pa pomeni počasnejše iskanje rešitev za probleme nesimetričnega trgovskega potnika.

Tabela 6.7: Rezultati algoritma *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	42,00	7,69	6,00
2	ftv33	34	1286	1286	0,00	1562,00	21,46	286,00
3	ftv35	36	1473	1473	0,00	1781,00	20,91	340,00
4	ftv38	39	1530	1530	0,00	1849,00	20,85	352,00
5	p43	43	5620	5620	0,00	5709,00	1,58	111,00
6	ftv44	45	1613	1613	0,00	2006,00	24,36	426,00
7	ftv47	48	1776	1776	0,00	2423,00	36,43	797,00
8	ry48p	48	14422	14422	0,00	19450,00	34,86	6523,00
9	ft53	53	6905	6973	0,98	9339,00	35,25	2767,00
10	ftv55	56	1608	1608	0,00	2356,00	46,52	879,00
11	ftv64	65	1839	1839	0,00	2736,00	48,78	1070,00
12	ft70	70	38673	38975	0,78	44351,00	14,68	6516,00
13	ftv70	71	1950	1950	0,00	2786,00	42,87	958,00
14	ftv90	91	1579	1581	0,13	2600,00	64,66	1114,00
15	kro124p	100	36230	36241	0,03	58362,00	61,09	27274,00
16	ftv100	101	1788	1790	0,11	2950,00	64,99	1261,00
17	ftv110	111	1958	1964	0,31	3303,00	68,69	1407,00
18	ftv120	121	2166	2168	0,09	3665,00	69,21	1588,00
19	ftv130	131	2307	2309	0,09	4096,00	77,55	1918,00
20	ftv140	141	2420	2422	0,08	4365,00	80,37	2094,00
21	ftv150	151	2611	2636	0,96	4797,00	83,72	2312,00
22	ftv160	161	2683	2695	0,45	5261,00	96,09	2825,00
23	ftv170	171	2755	2792	1,34	5610,00	103,63	3132,00
24	rbg323	323	1326	1373	3,54	2836,00	113,88	1957,00
25	rbg358	358	1163	1241	6,71	2954,00	154,00	2326,00
26	rbg403	403	2465	2480	0,61	4009,00	62,64	2108,00
27	rbg443	443	2720	2746	0,96	4380,00	61,03	2247,00

6.3.2 Algoritem *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/bin*

Rezultati, ki smo jih dobili pri reševanju nesimetričnega trgovskega potnika z algoritmom *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/bin*, so prikazani v tabeli 6.8. Iz njih lahko izberemo, da je algoritem 11-krat našel optimalno rešitev, kar pomeni, da je v grafih poiskal optimalni cikel, ki ga mora prehoditi trgovski potnik, da se vrne v začetni kraj, 12-krat pa algoritmu ni uspelo poiskati optimalne rešitve.

Algoritem je enako kot njegov predhodnik (algoritem *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp*) med grafi našel optimalno rešitev pri grafu *ftv70*, med-

tem ko mu je prav tako to uspelo tudi pri grafu *ftv110*, kjer pa njegov predhodnik ni uspel.

Največje odstopanje od optimalne rešitve je algoritem *DE+RAI+OR-opt mod 3* v kombinaciji s strategijo *DE/rand/1/bin* izračunal pri grafu *ftv170*, ki ima 171 vozlišč. Rešitev, ki jo je našel, je za 2,21% slabša od optimalne, ki znaša 2755.

Algoritem se prav tako ni izkazal pri *rgb* grafih, saj pri nobenem od njih ni našel optimalne rešitve. Rešitve so približno enake kot pri njegovem predhodniku. Največje odstopanje je algoritem *DE+RAI+OR-opt mod 3* izračunal pri grafu *rbg358*.

Tabela 6.8: Rezultati algoritma *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/bin*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	45,00	15,38	10,00
2	ftv33	34	1286	1286	0,00	1604,00	24,73	356,00
3	ftv35	36	1473	1473	0,00	1820,00	23,56	408,00
4	ftv38	39	1530	1530	0,00	1936,00	26,54	489,00
5	p43	43	5620	5620	0,00	5773,00	2,72	200,00
6	ftv44	45	1613	1613	0,00	2182,00	35,28	671,00
7	ftv47	48	1776	1776	0,00	2533,00	42,62	951,00
8	ry48p	48	14422	14429	0,05	20730,00	43,74	8258,00
9	ft53	53	6905	6905	0,00	9339,00	35,25	2741,00
10	ftv55	56	1608	1608	0,00	2521,00	56,78	1130,00
11	ftv64	65	1839	1842	0,16	3030,00	64,76	1470,00
12	ft70	70	38673	38906	0,60	44803,00	15,85	6918,00
13	ftv70	71	1950	1950	0,00	3135,00	60,77	1465,00
14	ftv90	91	1579	1581	0,13	2917,00	84,74	1615,00
15	kro124p	100	36230	36241	0,03	66173,00	82,65	38742,00
16	ftv100	101	1788	1794	0,34	3718,00	107,94	2320,00
17	ftv110	111	1958	1958	0,00	4213,00	115,17	2716,00
18	ftv120	121	2166	2174	0,37	4907,00	126,55	3299,00
19	ftv130	131	2307	2309	0,09	5434,00	135,54	3774,00
20	ftv140	141	2420	2426	0,25	5980,00	147,11	4343,00
21	ftv150	151	2611	2640	1,11	6545,00	150,67	4794,00
22	ftv160	161	2683	2686	0,11	7204,00	168,51	5515,00
23	ftv170	171	2755	2816	2,21	7883,00	186,13	6300,00
24	rbg323	323	1326	1385	4,45	2917,00	119,98	2071,00
25	rbg358	358	1163	1244	6,96	3048,00	162,08	2467,00
26	rbg403	403	2465	2472	0,28	4098,00	66,25	2237,00
27	rbg443	443	2720	2745	0,92	4477,00	64,60	2386,00

6.3.3 Algoritem *DE+RAI+OR-opt mod 3* s strategijo *Xing*

Algoritem *DE+RAI+OR-opt mod 3* smo prav tako, kot vse predhodnike, preiskusili tudi s strategijo *Xing*. Njegove rezultate lahko vidimo v tabeli 6.9, od kod razberemo, da je omenjeni algoritem našel optimalno rešitev pri 15-ih grafih, medtem ko mu pri 8-ih grafih to ni uspelo.

Razen pri grafih *ry48p*, *ft70* in *kro124p*, mu je uspelo najti vse optimalne rešitve pri grafih, ki imajo do 101 vozlišč. Tako je pri grafu *ry48p* optimalno rešitev zgrešil za 0,15%, pri grafu *ft70* za 0,71% ter pri grafu *kro124p* za 0,03%.

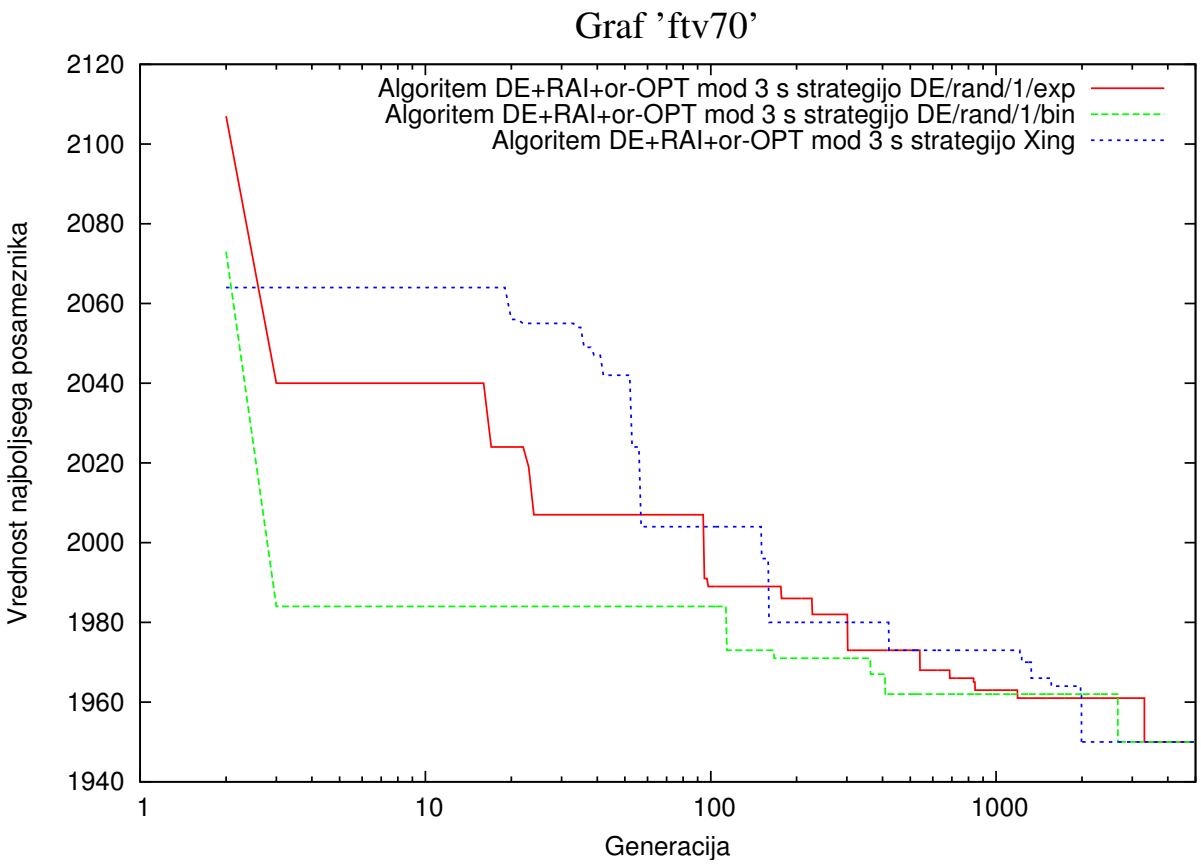
Tabela 6.9: Rezultati algoritma *DE+RAI+OR-opt mod 3* s strategijo *Xing*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1339,00	4,12	62,00
3	ftv35	36	1473	1473	0,00	1515,00	2,85	51,00
4	ftv38	39	1530	1530	0,00	1591,00	3,99	71,00
5	p43	43	5620	5620	0,00	5629,00	0,16	12,00
6	ftv44	45	1613	1613	0,00	1698,00	5,27	84,00
7	ftv47	48	1776	1776	0,00	1822,00	2,59	62,00
8	ry48p	48	14422	14444	0,15	14843,00	2,92	304,00
9	ft53	53	6905	6905	0,00	7260,00	5,14	435,00
10	ftv55	56	1608	1608	0,00	1684,00	4,73	93,00
11	ftv64	65	1839	1839	0,00	1927,00	4,79	103,00
12	ft70	70	38673	38946	0,71	39613,00	2,43	553,00
13	ftv70	71	1950	1950	0,00	2064,00	5,85	138,00
14	ftv90	91	1579	1579	0,00	1746,00	10,58	221,00
15	kro124p	100	36230	36241	0,03	38077,00	5,10	2155,00
16	ftv100	101	1788	1788	0,00	1976,00	10,51	239,00
17	ftv110	111	1958	1960	0,10	2208,00	12,77	320,00
18	ftv120	121	2166	2168	0,09	2439,00	12,60	326,00
19	ftv130	131	2307	2313	0,26	2613,00	13,26	385,00
20	ftv140	141	2420	2422	0,08	2736,00	13,06	395,00
21	ftv150	151	2611	2611	0,00	2988,00	14,44	457,00
22	ftv160	161	2683	2683	0,00	3092,00	15,24	503,00
23	ftv170	171	2755	2764	0,33	3195,00	15,97	537,00
24	rbg323	323	1326	1379	4,00	1510,00	13,88	123,00
25	rbg358	358	1163	1250	7,48	1365,00	17,37	116,00
26	rbg403	403	2465	2482	0,69	2561,00	3,89	82,00
27	rbg443	443	2720	2742	0,81	2829,00	4,01	93,00

Algoritem je še optimalno rešitev našel pri grafih *ftv150* in *ftv160*, ki imata 150 oziroma 160 vozlišč. S tem podatkom se je algoritem *DE+RAI+OR-opt mod 3* s

strategijo *Xing* izkazal kot najboljši v sklopu algoritmov, ki so opisani v tem podpoglavju, kjer smo opisali algoritem *DE+RAI+OR-opt mod 3*.

Pri *rbg* grafih algoritem ni blestel, saj ni našel optimalne rešitve za nobenega od njih. Najboljši rezultat smo z algoritmom dobili pri grafu *rbg403*, kjer je njegova rešitev znašala 2482, optimalna pa 2465. To pomeni, da je odstopanje med optimalno in dobljeno rešitvijo znašalo 0,69%. Pri grafu *rgb358* pa se je algoritem *DE+RAI+OR-opt mod 3* s strategijo *Xing* izkazal kot najslabši izmed algoritmov *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/exp* in *DE+RAI+OR-opt mod 3* s strategijo *DE/rand/1/bin*. Razlika med najboljšo najdeno in optimalno rešitvijo je znašala 7,48%.



Slika 6.3: Reševanje grafa *ftv70* z algoritmom *DE+RAI+OR-opt mod 3*

Na sliki 6.3 lahko vidimo konvergenčnost reševanja problema nesimetričnega trgovskega potnika, kjer je algoritem *DE+RAI+OR-opt mod 3* z vsemi kombinacijami strategij našel optimalno rešitev.

6.4 Algoritem *DE+RAI+OR-opt-V2*

Algoritem *DE+RAI+OR-opt-V2*, ki je opisan v poglavju 5.3.5, smo kombinirali na naslednje načine:

- *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/exp*,
- *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/bin* in
- *DE+RAI+OR-opt-V2* s strategijo *Xing*.

Tako smo dobili zopet nove tri kombinacije algoritma, katerih rezultati so prikazani v tem poglavju.

6.4.1 Algoritem *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/exp*

Po izvedbi algoritma *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/exp* smo ugotovili, da je algoritem izmed 23-ih primerov grafov iz knjižnice *TSPLIB* uspešno rešil le 8 grafov. To pomeni, da pri ostalih 15-ih ni našel optimalne rešitve. Te rezultate lahko vidimo v tabeli 6.10.

Največje odstopanje od optimalne rešitve je prišlo pri grafu *ft70*, kjer je algoritem izračunal obhod poti za 1,10% slabše kot je optimalni, ki znaša 38673. Zgrešek, ki ga je algoritem dobil pri grafih, kjer mu ni uspelo najti optimalne rešitve, je pri grafih znašal manj kot 0,3%, razen pri grafih *ftv44*, *ftv64*, *ft70* in *ftv170*.

Tako lahko sklepamo, da je algoritem sicer bil v večini primerov blizu optimalne rešitve, ampak se ni dokazal v optimizaciji same rešitve, ali pa bi poteboval večje število generacij, da bi problem ATSP rešil optimalno.

Naj še povemo, da za grafe *rbg*, algoritem ni uspel poiskati nobene optimalne rešitve. Najboljše je rešil graf *rbg403*, kjer je dobil rešitev 2473, kar je za 0,32% več kot optimalna rešitev, ki znaša 2465.

Tabela 6.10: Rezultati algoritma *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/exp*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1366,00	6,22	34,00
3	ftv35	36	1473	1473	0,00	1543,00	4,75	47,00
4	ftv38	39	1530	1530	0,00	1595,00	4,25	37,00
5	p43	43	5620	5620	0,00	5629,00	0,16	5,00
6	ftv44	45	1613	1623	0,62	1694,00	5,02	32,00
7	ftv47	48	1776	1777	0,06	1824,00	2,70	40,00
8	ry48p	48	14422	14446	0,17	14875,00	3,14	230,00
9	ft53	53	6905	6905	0,00	7311,00	5,88	247,00
10	ftv55	56	1608	1608	0,00	1687,00	4,91	43,00
11	ftv64	65	1839	1857	0,98	1931,00	5,00	45,00
12	ft70	70	38673	39099	1,10	39652,00	2,53	289,00
13	ftv70	71	1950	1955	0,26	2028,00	4,00	37,00
14	ftv90	91	1579	1579	0,00	1643,00	4,05	51,00
15	kro124p	100	36230	36241	0,03	37509,00	3,53	726,00
16	ftv100	101	1788	1791	0,17	1875,00	4,87	58,00
17	ftv110	111	1958	1960	0,10	2042,00	4,29	58,00
18	ftv120	121	2166	2172	0,28	2273,00	4,94	56,00
19	ftv130	131	2307	2313	0,26	2420,00	4,90	53,00
20	ftv140	141	2420	2426	0,25	2547,00	5,25	77,00
21	ftv150	151	2611	2613	0,08	2764,00	5,86	74,00
22	ftv160	161	2683	2691	0,30	2839,00	5,81	67,00
23	ftv170	171	2755	2785	1,09	2934,00	6,50	66,00
24	rbg323	323	1326	1371	3,39	1414,00	6,64	20,00
25	rbg358	358	1163	1224	5,25	1284,00	10,40	25,00
26	rbg403	403	2465	2473	0,32	2504,00	1,58	14,00
27	rbg443	443	2720	2735	0,55	2776,00	2,06	15,00

6.4.2 Algoritem *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/bin*

Rezultati rešitev grafov iz knjižnice *TSPLIB*, ki jih je reševal algoritem *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/bin*, so prikazani v tabeli 6.11. Iz njih hitro izvemo, da se algoritem prav tako kot njegov predhodnik, algoritem *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/exp*, ni izkazal najboljše, saj je izmed 23-ih primerov našel optimalno rešitev le pri 7-ih grafih.

Tabela 6.11: Rezultati algoritma *DE+RAI+OR-opt-V2* s strategijo *DE/rand/1/bin*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1375,00	6,92	39,00
3	ftv35	36	1473	1473	0,00	1540,00	4,55	47,00
4	ftv38	39	1530	1532	0,13	1600,00	4,58	35,00
5	p43	43	5620	5620	0,00	5629,00	0,16	5,00
6	ftv44	45	1613	1613	0,00	1701,00	5,46	39,00
7	ftv47	48	1776	1776	0,00	1833,00	3,21	46,00
8	ry48p	48	14422	14507	0,59	14897,00	3,29	234,00
9	ft53	53	6905	6905	0,00	7322,00	6,04	231,00
10	ftv55	56	1608	1628	1,24	1696,00	5,47	39,00
11	ftv64	65	1839	1846	0,38	1916,00	4,19	40,00
12	ft70	70	38673	38964	0,75	39663,00	2,56	300,00
13	ftv70	71	1950	1965	0,77	2037,00	4,46	42,00
14	ftv90	91	1579	1581	0,13	1650,00	4,50	51,00
15	kro124p	100	36230	36323	0,26	37874,00	4,54	823,00
16	ftv100	101	1788	1794	0,34	1867,00	4,42	54,00
17	ftv110	111	1958	1960	0,10	2049,00	4,65	59,00
18	ftv120	121	2166	2172	0,28	2275,00	5,03	57,00
19	ftv130	131	2307	2320	0,56	2441,00	5,81	66,00
20	ftv140	141	2420	2429	0,37	2561,00	5,83	73,00
21	ftv150	151	2611	2626	0,57	2770,00	6,09	69,00
22	ftv160	161	2683	2701	0,67	2857,00	6,49	64,00
23	ftv170	171	2755	2789	1,23	2926,00	6,21	65,00
24	rbg323	323	1326	1374	3,62	1411,00	6,41	17,00
25	rbg358	358	1163	1226	5,42	1282,00	10,23	24,00
26	rbg403	403	2465	2478	0,53	2501,00	1,46	12,00
27	rbg443	443	2720	2748	1,03	2781,00	2,24	17,00

6.4.3 Algoritem *DE+RAI+OR-opt-V2* s strategijo *Xing*

Algoritem *DE+RAI+OR-opt-V2* s strategijo *Xing* je dal rezultate, ki so prikazani v tabeli 6.12. Iz nje lahko vidimo, da je algoritem optimalno rešil 6 primerov, 17 pa mu jih ni uspelo.

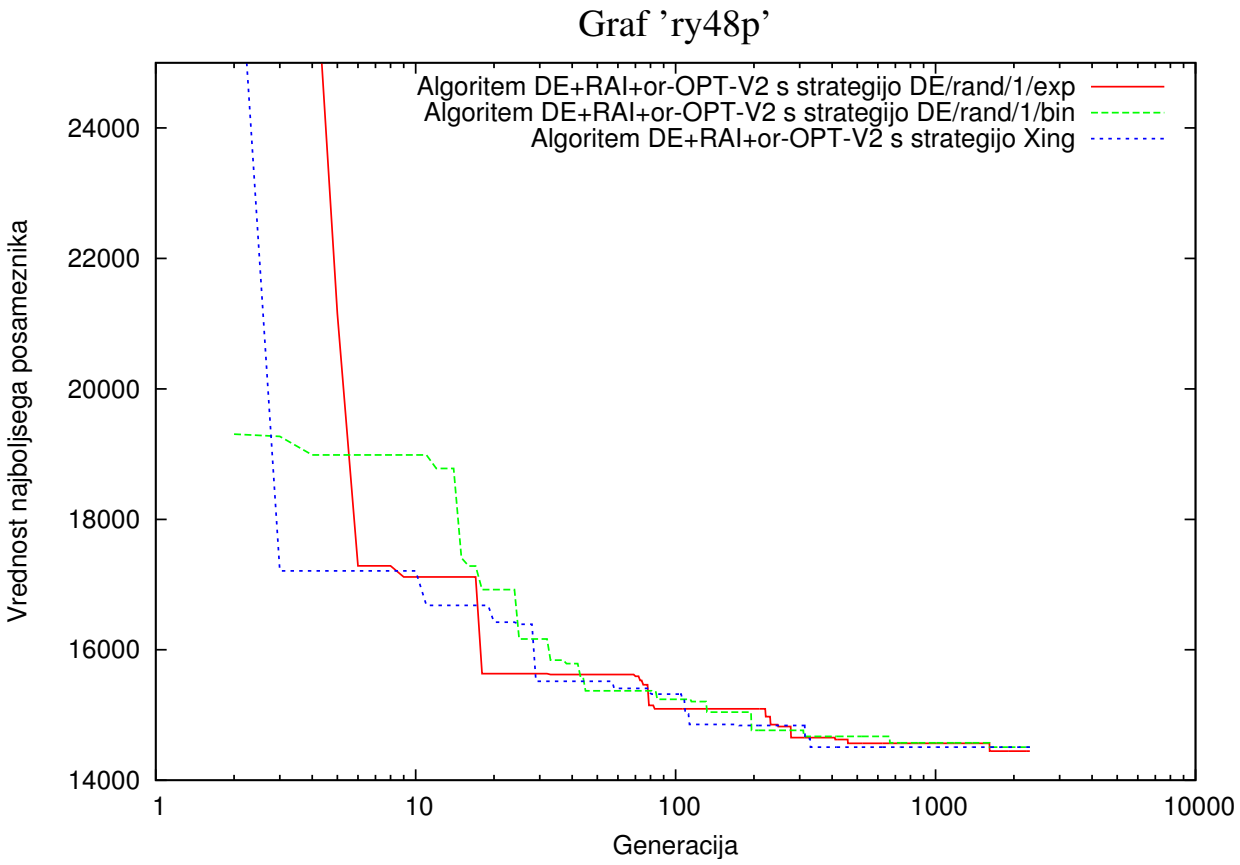
Tako lahko zopet za algoritem *DE+RAI+OR-opt-V2*, ki smo ga kombinirali s strategijo *Xing*, povemo, da so rezultati, ki jih je dobil, slabi v primerjavi z vsemi ostalimi algoritmi, ki so opisani v prejšnjih poglavjih.

Tabela 6.12: Rezultati algoritma *DE+RAI+OR-opt-V2* s strategijo *Xing*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1288,00	0,16	9,00
3	ftv35	36	1473	1475	0,14	1485,00	0,81	5,00
4	ftv38	39	1530	1530	0,00	1548,00	1,18	12,00
5	p43	43	5620	5620	0,00	5620,00	0,00	0,00
6	ftv44	45	1613	1613	0,00	1644,00	1,92	10,00
7	ftv47	48	1776	1777	0,06	1777,00	0,06	2,00
8	ry48p	48	14422	14507	0,59	14521,00	0,69	30,00
9	ft53	53	6905	6915	0,14	6919,00	0,20	22,00
10	ftv55	56	1608	1608	0,00	1608,00	0,00	2,00
11	ftv64	65	1839	1850	0,60	1851,00	0,65	7,00
12	ft70	70	38673	38994	0,83	39149,00	1,23	102,00
13	ftv70	71	1950	1973	1,18	1973,00	1,18	4,00
14	ftv90	91	1579	1581	0,13	1581,00	0,13	7,00
15	kro124p	100	36230	36241	0,03	36241,00	0,03	0,00
16	ftv100	101	1788	1790	0,11	1790,00	0,11	0,00
17	ftv110	111	1958	1964	0,31	1964,00	0,31	0,00
18	ftv120	121	2166	2172	0,28	2172,00	0,28	0,00
19	ftv130	131	2307	2331	1,04	2331,00	1,04	0,00
20	ftv140	141	2420	2433	0,54	2433,00	0,54	0,00
21	ftv150	151	2611	2637	1,00	2642,00	1,19	2,00
22	ftv160	161	2683	2704	0,78	2705,00	0,82	5,00
23	ftv170	171	2755	2771	0,58	2771,00	0,58	0,00
24	rbg323	323	1326	1367	3,09	1409,00	6,26	18,00
25	rbg358	358	1163	1217	4,64	1279,00	9,97	23,00
26	rbg403	403	2465	2475	0,41	2501,00	1,46	12,00
27	rbg443	443	2720	2732	0,44	2773,00	1,95	16,00

Na sliki 6.4 vidimo potek reševanja najboljšega posameznika z algoritmom *DE+RAI+OR-opt-V2* v vseh treh omenjenih strategijah za graf *ry48p*. Glede na to, da x-os predstavlja logaritemska lestvica, lahko rečemo, da so rešitve do 20-te generacije dokaj hitro konvergirale k optimalni rešitvi. Od 20-te generacije dalje pa se je hitrost konvergence manjšala in v nobenem od primerov ni dosegala optimalne rešitve.

Zato smo se odločili, da bomo algoritem *DE+RAI+OR-opt-V2* popravili. Tako smo razvlili nov algoritem *DE+RAI+OR-opt-V3*, katerega rezultati so prikazani v naslednjem poglavju.



Slika 6.4: Reševanje grafa *ry48p* z algoritmom *DE+RAI+OR-opt-V2*

6.5 Algoritem *DE+RAI+OR-opt-V3*

V tem poglavju opisujemo rezultate algoritma *DE+RAI+OR-opt-V3*. Algoritem predstavlja nadgradnjo algoritma *DE+RAI+OR-opt-V2*. Predstavitev obeh algoritmov lahko vidimo v poglavju 5.

Algoritem *DE+RAI+OR-opt-V3* smo prav tako kot vse prejšnje kombinirali z različnimi strategijami in sicer:

- *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/exp*,
- *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/bin* in
- *DE+RAI+OR-opt-V3* s strategijo *Xing*.

Algoritem se je izkazal kot najboljši med vsemi algoritmi *DEATSP*. Zato smo algoritem z vsako strategijo izvedli 25-krat za posamezni graf. Pri reševanju nesimetričnega problema trgovskega potnika smo se ozirali predvsem na izračun optimalnih rešitev. Čas, ki je prav tako faktor zmogljivosti določenega hevrističnega algoritma, nas pri reševanju ni zanimal, seveda pa smo algoritem razvijati tako, da je čim hitreje uspel priti do rešitve, ki je optimalna, oziroma vsaj blizu optimalne.

V nadaljevanju opisujemo rešitve problemov za nesimetričnega trgovskega potnika, ki smo jih reševali z algoritmom *DE+RAI+OR-opt-V3* nad grafi iz knjižnice *TSPLIB*. Prikažemo povprečne rezultate izmed 25-ih rešitev za vsako strategijo, najboljše dobljene rezultate ter grafične opise potekov reševanja problemov nesimetričnega trgovskega potnika (ATSP).

6.5.1 Algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/exp*

Rezultati povprečnih rešitev za algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/exp* so prikazani v tabeli 6.13. Iz njih je razvidno, da je algoritem optimalno rešil 21 primerov, za dva primera pa mu to ni uspelo, in sicer:

- pri grafu *ft70*, kjer je povprečna vrednost najboljših posameznikov, ki jo je dobil algoritem v 25-ih poskusih reševanja, enaka 38722, kar je za 0,12% slabše kot optimalna rešitev, ki znaša 38673, in
- pri grafu *ftv170*, kjer je algoritem dobil povprečno vrednost najboljših posameznikov 2757, kar je za 0,08% slabše od optimalne (2755).

Algoritem je reševal tudi grafe *rbg*, kjer je v 25-ih poskusih reševanja dobil povprečno vrednost najboljših posameznikov, ki je enaka optimalni reštvi in sicer pri grafu *rbg403*. Le za 0,05% je povprečna vrednost najboljših posameznikov zgrešila optimalno rešitev še na grafu *rbg443*, kjer je v povprečju obhod znašal 2721.

Tabela 6.13: Povprečni rezultati 25-ih poskusov za algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/exp*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1302,75	1,28	26,00
3	ftv35	36	1473	1473	0,00	1478,75	0,29	7,50
4	ftv38	39	1530	1530	0,00	1539,25	0,46	11,00
5	p43	43	5620	5620	0,00	5621,00	0,01	1,00
6	ftv44	45	1613	1613	0,00	1643,00	1,63	16,00
7	ftv47	48	1776	1776	0,00	1781,50	0,24	5,25
8	ry48p	48	14422	14422	0,00	14535,25	0,59	65,25
9	ft53	53	6905	6905	0,00	6956,50	0,57	56,50
10	ftv55	56	1608	1608	0,00	1620,75	0,59	14,25
11	ftv64	65	1839	1839	0,00	1854,75	0,64	10,50
12	ft70	70	38673	38722	0,12	39055,75	0,99	140,25
13	ftv70	71	1950	1950	0,00	1966,75	0,63	13,50
14	ftv90	91	1579	1579	0,00	1595,25	0,79	17,75
15	kro124p	100	36230	36230	0,00	36476,25	0,51	399,50
16	ftv100	101	1788	1788	0,00	1806,75	0,70	24,75
17	ftv110	111	1958	1958	0,00	1983,75	1,18	28,25
18	ftv120	121	2166	2166	0,00	2200,50	1,48	26,50
19	ftv130	131	2307	2307	0,00	2350,50	1,71	32,50
20	ftv140	141	2420	2420	0,00	2463,25	1,82	38,50
21	ftv150	151	2611	2611	0,00	2675,75	2,33	47,50
22	ftv160	161	2683	2683	0,00	2758,00	2,62	42,75
23	ftv170	171	2755	2757	0,08	2840,25	3,08	45,00
24	rbg323	323	1326	1347	1,62	1387,00	4,47	11,75
25	rbg358	358	1163	1176	1,14	1235,25	6,16	16,25
26	rbg403	403	2465	2465	0,00	2475,75	0,33	4,50
27	rbg443	443	2720	2721	0,05	2737,25	0,47	6,75

6.5.2 Algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/bin*

Povprečni rezultati izmed 25-tih rešitev problema nesimetričnega trgovskega potnika, ki smo ga reševali na grafih iz knjižnice *TSPLIB* z algoritmom *DE+RAI+OR-*

opt-V3 in strategijo *DE/rand/1/bin*, so prikazani v tabeli 6.14.

Kot je iz nje razvidno, je algoritem v povprečju rešil 20 primerov, kjer je uspel vedno najti optimalno rešitev. To pa mu ni uspelo pri grafih *ft70*, *ftv140* in *ftv170*. Največja razlika od optimalne je bila pri reševanju grafa *ftv170*, kjer je odstotek odstopanja znašal 0,17. To pomeni, da je algoritem v povprečju dobil rešitev, ki je enaka 2764, medtem ko optimalna znaša 2755.

Za *rbg* grafe lahko povemo, da je algoritem v vseh reševanjih uspešno poiskal optimano rešitev pri grafu *rbg403*. Obhod na grafu *rbg443* pa se je v povprečju od optimalne razlikoval le za 0,01%.

Tabela 6.14: Povprečni rezultati 25-ih poskusov za algoritem *DE+RAI+OR-opt-V3* s strategijo *DE/rand/1/bin*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1286,25	0,00	5,75
3	ftv35	36	1473	1473	0,00	1477,75	0,07	5,75
4	ftv38	39	1530	1530	0,00	1539,75	0,16	8,50
5	p43	43	5620	5620	0,00	5622,00	0,01	1,00
6	ftv44	45	1613	1613	0,00	1639,75	1,20	15,75
7	ftv47	48	1776	1776	0,00	1782,50	0,10	5,75
8	ry48p	48	14422	14422	0,00	14538,25	0,18	70,50
9	ft53	53	6905	6905	0,00	6940,75	0,11	44,25
10	ftv55	56	1608	1608	0,00	1627,50	1,04	15,75
11	ftv64	65	1839	1839	0,00	1867,25	1,15	13,75
12	ft70	70	38673	38752	0,13	39290,50	1,15	152,00
13	ftv70	71	1950	1950	0,00	1982,75	1,19	18,00
14	ftv90	91	1579	1579	0,00	1618,50	2,17	31,75
15	kro124p	100	36230	36230	0,00	36770,25	1,13	488,25
16	ftv100	101	1788	1788	0,00	1840,75	2,53	40,75
17	ftv110	111	1958	1958	0,00	2038,00	3,94	59,25
18	ftv120	121	2166	2166	0,00	2257,50	3,96	53,00
19	ftv130	131	2307	2307	0,00	2420,75	4,80	62,25
20	ftv140	141	2420	2423	0,12	2538,00	4,48	73,75
21	ftv150	151	2611	2611	0,00	2765,00	5,49	80,50
22	ftv160	161	2683	2683	0,00	2845,25	5,45	72,00
23	ftv170	171	2755	2764	0,17	2926,00	5,80	69,00
24	rbg323	323	1326	1347	1,27	1400,75	5,17	10,25
25	rbg358	358	1163	1174	0,95	1263,50	8,15	14,75
26	rbg403	403	2465	2465	0,00	2481,25	0,16	5,00
27	rbg443	443	2720	2720	0,01	2746,25	0,49	7,00

6.5.3 Algoritem *DE+RAI+OR-opt-V3* s strategijo *Xing*

Algoritem *DE+RAI+OR-opt-V3* s strategijo *Xing* je v povprečju izmed najboljših posameznikov v 25-ih poiskusov vedno uspešno poiskal optimalno rešitev 17-krat, kar prikazuje tabela 6.15. 6-krat pa algoritmu to ni uspelo.

Najslabše se je algoritem v povprečju odrezal pri grafu *ftv170*, kjer je odstopanje od optimalne rešitve (2755) bilo 0,64%, kar pomeni, da je obhod trgovskega potnika po grafu znašal 2772.

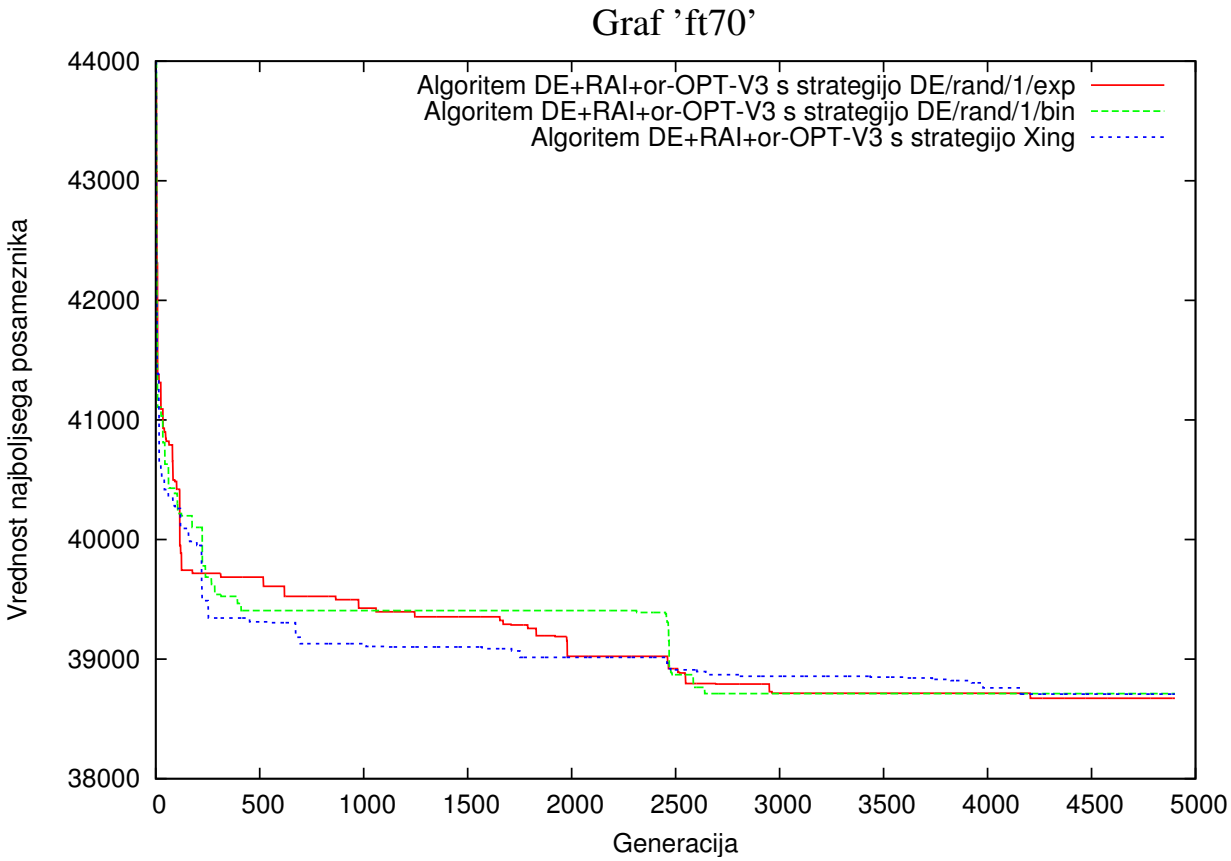
Tabela 6.15: Povprečni rezultati 25-ih poskusov za algoritem *DE+RAI+OR-opt-V3* s strategijo *Xing*

	graf	n	opt.	min.	%	povp.	%	std.dev.
1	br17	17	39	39	0,00	39,00	0,00	0,00
2	ftv33	34	1286	1286	0,00	1286,00	0,00	0,00
3	ftv35	36	1473	1473	0,00	1473,50	0,04	0,00
4	ftv38	39	1530	1530	0,03	1530,50	0,00	0,00
5	p43	43	5620	5620	0,00	5620,00	0,00	0,00
6	ftv44	45	1613	1613	0,00	1613,00	0,00	0,00
7	ftv47	48	1776	1776	0,00	1776,00	0,00	0,00
8	ry48p	48	14422	14422	0,00	14422,25	0,00	2,00
9	ft53	53	6905	6905	0,00	6905,00	0,00	0,25
10	ftv55	56	1608	1608	0,00	1608,00	0,00	0,00
11	ftv64	65	1839	1839	0,00	1839,00	0,00	0,00
12	ft70	70	38673	38707	0,09	38710,00	0,07	11,00
13	ftv70	71	1950	1952	0,14	1952,75	0,00	0,00
14	ftv90	91	1579	1579	0,00	1579,00	0,00	0,00
15	kro124p	100	36230	36230	0,00	36230,00	0,00	0,00
16	ftv100	101	1788	1788	0,00	1788,00	0,00	0,00
17	ftv110	111	1958	1958	0,00	1958,00	0,00	0,00
18	ftv120	121	2166	2166	0,00	2166,00	0,00	0,00
19	ftv130	131	2307	2307	0,00	2307,00	0,00	0,00
20	ftv140	141	2420	2421	0,06	2421,50	0,06	0,00
21	ftv150	151	2611	2611	0,00	2611,00	0,00	0,00
22	ftv160	161	2683	2683	0,03	2683,75	0,03	0,00
23	ftv170	171	2755	2772	0,64	2772,50	0,56	0,00
24	rbg323	323	1326	1342	1,24	1377,75	3,69	9,00
25	rbg358	358	1163	1176	1,14	1215,25	4,38	12,75
26	rbg403	403	2465	2465	0,00	2467,25	0,07	2,00
27	rbg443	443	2720	2720	0,00	2724,50	0,13	2,00

Pri *rbg* grafih pa je algoritem s strategijo *Xing* našel optimalno rešitev pri dveh grafih v 25-ih poskusih, kar je boljše od algoritma v kombinaciji strategij

DE/rand/1/exp in *DE/rand/1/bin*, kjer mu je to uspelo samo pri grafu *rbg403*. Optimalno rešitev je našel še pri grafu *rbg443*.

Iz slike 6.5, ki prikazuje reševanje problema nesimetričnega trgovskega potnika na grafu *ft70*, je razvidno, da algoritem s strategijami zelo hitro konvergira k rešitvi. Algoritem je tako v povprečju rešitev bil po 3000 generacijah že zelo blizu rešitve.



Slika 6.5: Reševanje grafa *ft70* z algoritmom *DE+RAI+OR-opt-V3*

6.5.4 Primerajva najboljših rešitev za algoritem *DE+RAI+OR-opt-V3*

Primerjajmo še najboljše rešitve algoritma *DE+RAI+OR-opt-V3* v kombinacijah z vsemi tremi strategijami. Rezultati so prikazani v tabeli 6.16.

Algoritem s strategijo *DE/rand/1/exp* je med najboljšimi rezultati povsod našel optimalno rešitev, kakor prikazuje stolpec, ki je označen z oznako '*min.*'.

Algoritem s strategijo *DE/rand/1/bin* pa med najboljšimi rešitvami ni našel optimalne rešitve pri pri grafu *ft70*, kjer se je rešitev, ki jo je dobil (38675), od optimalne (38673) razlikovala za 0,01%.

Vse optimalne rešitve pa je algoritem zopet našel v kombinaciji s strategijo *Xing*.

Tabela 6.16: Primerjava najboljših in povprečnih rezultatov za algoritem *DE+RAI+OR-opt-V3*

	graf	n	opt.	*DE/rand/1/exp*		*DE/rand/1/bin*		*Xing*	
				min.	povp.	min.	povp.	min.	povp.
1	br17	17	39	39	39	39	39	39	39
2	ftv33	34	1286	1286	1286	1286	1286	1286	1286
3	ftv35	36	1473	1473	1473	1473	1473	1473	1473
4	ftv38	39	1530	1530	1530	1530	1530	1530	1530
5	p43	43	5620	5620	5620	5620	5620	5620	5620
6	ftv44	45	1613	1613	1613	1613	1613	1613	1613
7	ftv47	48	1776	1776	1776	1776	1776	1776	1776
8	ry48p	48	14422	14422	14422	14422	14422	14422	14422
9	ft53	53	6905	6905	6905	6905	6905	6905	6905
10	ftv55	56	1608	1608	1608	1608	1608	1608	1608
11	ftv64	65	1839	1839	1839	1839	1839	1839	1839
12	ft70	70	38673	38673	38722	38675	38752	38673	38707
13	ftv70	71	1950	1950	1950	1950	1950	1950	1952
14	ftv90	91	1579	1579	1579	1579	1579	1579	1579
15	kro124p	100	36230	36230	36230	36230	36230	36230	36230
16	ftv100	101	1788	1788	1788	1788	1788	1788	1788
17	ftv110	111	1958	1958	1958	1958	1958	1958	1958
18	ftv120	121	2166	2166	2166	2166	2166	2166	2166
19	ftv130	131	2307	2307	2307	2307	2307	2307	2307
20	ftv140	141	2420	2420	2420	2420	2423	2420	2421
21	ftv150	151	2611	2611	2611	2611	2611	2611	2611
22	ftv160	161	2683	2683	2683	2683	2683	2683	2683
23	ftv170	171	2755	2755	2757	2755	2764	2755	2772
24	rbg323	323	1326	1336	1347	1338	1347	1336	1342
25	rbg358	358	1163	1169	1176	1172	1174	1169	1176
26	rbg403	403	2465	2465	2465	2465	2465	2465	2465
27	rbg443	443	2720	2720	2721	2720	2720	2720	2720

Pri *rbg* grafih vidimo, da je algoritem *DE+RAI+OR-opt-V3* z vsemi strategijami našel optimalno rešitev pri grafu *rbg403*, ki znaša 2465 in pri grafu *rbg443*, kjer je dolžina obhoda enaka 2720. Algoritmu pa ni z nobeno strategijo uspelo najti optimalne rešitve pri grafih *rbg323* in *rbg358*. S strategijama *DE/rand/1/exp* in *Xing* je za dolžino obhoda pri grafu *rbg323* izračunal rešitev 1336, s strategijo

DE/rand/1/bin pa 1338. Tukaj še naj enkrat omenimo, da smo iskanje rešitev pri *rbg* grafih izvajali le n-krat (n - število vseh generacij), medtem ko smo pri vseh ostalih rešitve poskušali iskati n^2-krat.

Iz teh podatkov lahko izzvemo, da je najboljša uspešnost algoritma *DE+RAI+OR-opt-V3* v kombinacijah s strategijami enaka 99,23%, če izvzamemo grafe *rbg*, kjer je njegova uspešnost 50%, kjer je število poskusov iskanja rešitve dvakrat manjša.

6.6 Primerjava rezultatov algoritmov *DEATSP*

V poglavju opisujemo in predstavljamo primerjavo rezultatov vseh algoritmov *DEATSP* nad grafi iz knjižnice *TSPLIB*.

Ti rezultati so prikazani v tabeli 6.17, kjer stoplec '*algoritem*' označuje ime algoritma *DEATSP*, stolpec '*strategija*' pa strategijo ob izbranem algoritmu. Stolpec '*pogl.*' prikazuje številko poglavja, kjer je izbrana kombinacija algoritma in strategija opisana. V stolcu '*opt. da/ne*' vidimo, kolikokrat je algoritem našel optimalno rešitev in kolikokrat ne. Zadnji stolpec '*% opt.*' pa prikazuje odstotek uspešnosti najdbe optimalne rešitve za nabor 23-ih grafov (grafi *rbg* niso vključeni).

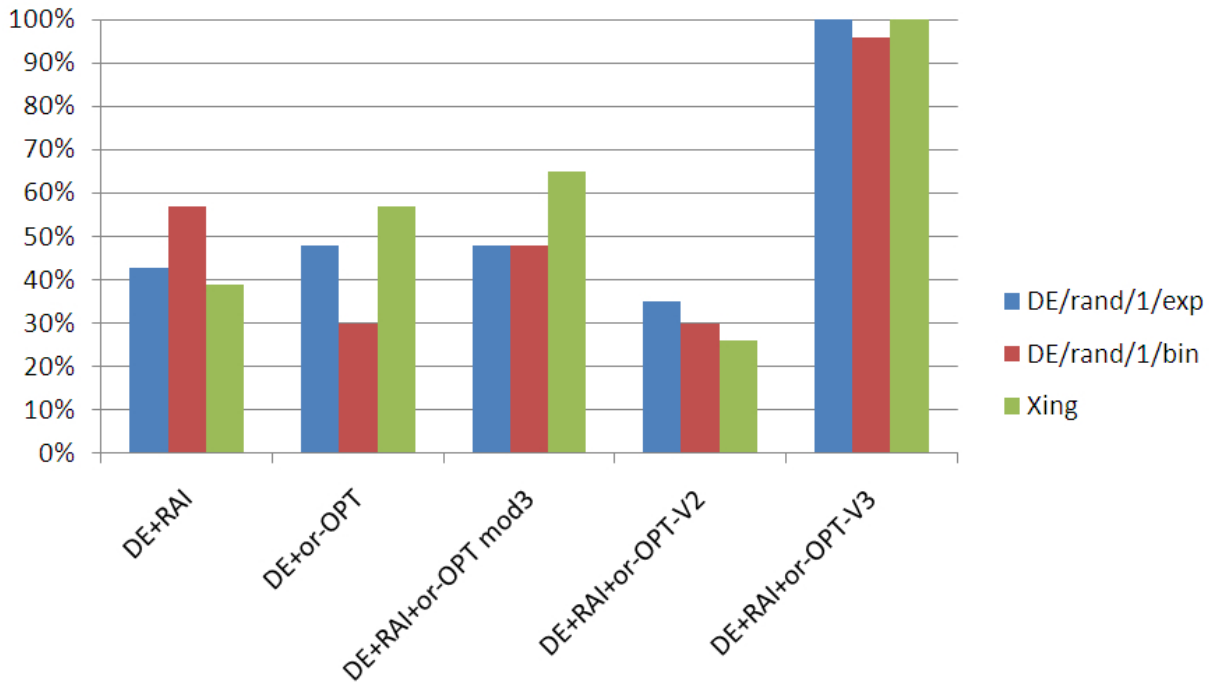
Tabela 6.17: Primerjava rešitev za algoritme *DEATSP*

	algoritem	strategija	pogl.	opt. da/ne	% opt.
1	*DE+RAI*	*DE/rand/1/exp*	6.1.1	10 / 13	43%
		DE/rand/1/bin	6.1.2	13 / 10	57%
		Xing	6.1.3	9 / 14	39%
2	*DE+OR-opt*	*DE/rand/1/exp*	6.2.1	11 / 12	48%
		DE/rand/1/bin	6.2.1	7 / 16	30%
		Xing	6.2.1	13 / 10	57%
3	*DE+RAI+OR-opt mod 3*	*DE/rand/1/exp*	6.3.1	11 / 12	48%
		DE/rand/1/bin	6.3.2	11 / 12	48%
		Xing	6.3.3	15 / 8	65%
4	*DE+RAI+OR-opt-V2*	*DE/rand/1/exp*	6.4.1	8 / 15	35%
		DE/rand/1/bin	6.4.2	7 / 16	30%
		Xing	6.4.3	6 / 17	26%
5	*DE+RAI+OR-opt-V3*	*DE/rand/1/exp*	6.5.1	23 / 0	100%
		DE/rand/1/bin	6.5.2	22 / 1	96%
		Xing	6.5.3	23 / 0	100%

Naj še omenimo, da so rezultati, ki so prikazani v tabeli za algoritem *DE+RAI+OR-opt-V3* najboljši rezultati in ne povprečje 25-ih rešitev obhodov grafov.

Iz rezultatov tabeli, lahko hitro opazimo, da je algoritem *DE+RAI+OR-opt-V3* z vsemi strategijami mnogo boljši od ostalih algoritmov *DEATSP*. Prav tako hitro razvidno, da je algoritem *DE+RAI+OR-opt-V2* dal najslabše rezultate. Njegova povprečna uspešnost je bila okoli 30%.

Uspešnost algoritmov *DEATSP* prikažimo še na grafikonu (slika 6.6). Na x osi je prikazan posamezni algoritem z vsako strategijo. Na osi y pa je prikazana uspešnost algoritmov v kombinacijah s strategijami. Strategije so označene z modro, rdečo in zeleno barvo, kot prikazuje legenda.



Slika 6.6: Prikaz uspešnosti algoritmov *DEATSP*

6.7 Praktičen primer iskanja obhoda slovenskih mest

V poglavju 5.1 smo na sliki 5.1 prikazali primer obhoda zračne poti (cikel) med osmimi slovenskimi mesti, ki pa ni optimalen. Zato smo se odločili poiskati optimalen cikel zračne poti.

Zračne razdalje so prikazane v tabeli 5.1. Podatke razdalj in mest smo izvzeli iz programskega paketa *TIS karta* [31]. Skupna razdalja najkrajšega obhoda znaša 518 km [31].

Da je ta podatek resničen, smo pokazali tudi z uporabo algoritmov *DEATSP*. Vsi algoritmi so našli isto rešitev, ki je prikazana na sliki 6.7:

$$MB \longrightarrow MS \longrightarrow NM \longrightarrow KP \longrightarrow NG \longrightarrow KR \longrightarrow LJ \longrightarrow CE \longrightarrow MB$$

Slika 6.7: Optimalna obhodna zračna pot med osmimi slovenskimi mesti [31]

Poglavje 7

Zaključek

V diplomskem delu smo obravnavali reševanje problema nesimetričnega trgovskega potnika, ki je eden izmed (naj)zahtevnejših algoritmov v družini problemov trgovskega potnika. Po uvodu smo predstavili sorodna dela, kjer smo izpostavili hevristične algoritme in osnove evolucijskih algoritmov, kamor spada tudi diferencialna evolucija.

V naslednjih poglavjih smo spoznali algoritem diferencialne evolucije in osnove hevrističnih algoritmov *RAI* in *OR-opt*, ki so ju avtorji razvili za reševanje optimizacijskih problemov, kamor spada tudi problem trgovskega potnika.

Z združitvijo diferencialne evolucije in hevrističnih algoritmov *RAI* in *OR-opt*, smo razvilili nove algoritme, ki smo jih poimenovali algoritmi *DEATSP*, katerih implementacije in psevdokode smo predstavili v jedru diplomskega dela. Algoritme smo kombinirali z različnimi strategijami diferencialne evolucije, kjer smo prav tako uvedli novo strategijo *Xing*.

Tako smo načrtovali in implementirali 5 različnih algoritmov, ki smo jih kombinirali s tremi strategijami (*DE/rand/1/exp*, *DE/rand/1/bin* in *Xing*) – torej skupaj 15 različic programov. Izmed teh se je najboljše izkazal algoritem *DE+RAI+OR-opt-V3*, katerega uspešnost iskanja optimalnih obhodov za nesimetričnega trgovskega potnika na naboru 23-ih grafov iz knjižnice *TSPLIB* je bila 99,23%.

V nadaljnje raziskovanje bi lahko vključili predvsem časovno optimizacijo algoritmov *DEATSP*, samoadaptivnost vhodnih parametrov in optimizacijo na področju pomnilniškega prostora ter tako dosegli, da bi naši hevristični algoritmi *DEATSP* delovali hitreje, bolje, ter da bi morda še uspešneje reševali probleme nesimetričnega trgovskega potnika z diferencialno evolucijo in/ali z drugimi hevrističnimi algoritmi.

Literatura

[1] E. Aarts in J. K. Lenstra, urednika (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA.

[2] T. Bäck (2002). Adaptive Business Intelligence Based on Evolution Strategies: Some Application Examples of Self-Adaptive Software. *Information Sciences*, 148:113–121.

[3] T. Bäck, D. B. Fogel in Z. Michalewicz, uredniki (1997). *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press.

[4] V. Batagelj (1993). Algoritmi za reševanje splošnega problema trgovskega potnika. *Uporabna Informatika*, (1):27–32.

[5] J. Brest, B. Bošković, S. Greiner, V. Žumer in M. S. Maučec (2007a). Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 11(7):617–629.

[6] J. Brest, Š. Brest in J. Žerovnik (2005). Primerjava hevrističnih algoritmov za trgovskega potnika. *Zbornik Elektotehniške in računalniške konference ERK*.

[7] J. Brest, S. Greiner, B. Bošković, M. Mernik in V. Žumer (2006). Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657.

[8] J. Brest in J. Žerovnik (1999). An approximation algorithm for the asymmetric traveling salesman problem. *Ric. oper. 28:59–67*.

[9] J. Brest, J. Žerovnik in V. Žumer (2003). Algoritem za nesimetrični problem trgovskega potnika. *Elektrotehniški vestnik*, 70(1-2):40–45.

[10] J. Brest, V. Žumer in M. S. Maučec (2007b). Population size in differential evolution algorithm. *Elektrotehniški vestnik*, 74(1-2):55–60.

[11] A. E. Eiben in J. E. Smith (2003). *Introduction to Evolutionary Computing*. Natural Computing. Springer-Verlag, Berlin.

[12] V. Feoktistov (2006). *Differential Evolution: In Search of Solutions (Springer Optimization and Its Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[13] L. J. Fogel, A. J. Owens in M. J. Walsh (1966). *Artificial Intelligence through Simulated Evolution*. Willey, New York.

[14] M. R. Garey in D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

[15] D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts.

[16] P. Korošec, J. Šilc, K. Oblak in F. Kosel (2007). The differential ant-stigmergy algorithm: An experimental evaluation and a real-world application. V *2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, strani 157–164, Singapore.

[17] J. R. Koza (1992). *Genetic programming: On the Programming of Computers by Means of Natural Selection*. Complex adaptive systems, Cambridge, MA: The MIT (Massachusetts Institute of Technology) Press, 1992.

[18] E. L. Lawler, J. K. Lenstra, R. Kan in D. B. Shmoys (1985). *The traveling salesman problem*. Wiley, New York.

[19] J. Liu in J. Lampinen (2002a). Adaptive Parameter Control of Differential Evolution. V *Proceedings of the 8th International Conference on Soft Computing (MENDEL 2002)*, strani 19–26.

[20] J. Liu in J. Lampinen (2002b). On Setting the Control Parameter of the Differential Evolution Method. V *Proceedings of the 8th International Conference on Soft Computing (MENDEL 2002)*, strani 11–18.

[21] J. Liu in J. Lampinen (2005). A Fuzzy Adaptive Differential Evolution Algorithm. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9(6):448–462.

[22] I. H. Murata T. in T. H. (September 1996). Genetic algorithms for flowshop scheduling problems. *Computers and Industrial Engineering*, 30:1061–1071(11).

[23] K. V. Price, R. M. Storn in J. A. Lampinen (2005). *Differential Evolution, A Practical Approach to Global Optimization*. Springer.

[24] K. Prnaver (2007). *Christofidesov algoritem*. Diplomsko delo, Fakulteta za naravoslovje in matematiko, Univerza v Mariboru.

[25] G. Reinelt (1991). TSPLIB— A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384.

[26] J. Rönkkönen, S. Kukkonen in K. V. Price (2005). Real-Parameter Optimization with Differential Evolution. V *The 2005 IEEE Congress on Evolutionary Computation CEC 2005*, strani 506 – 513. IEEE Press.

[27] D. J. Rosenkrantz, R. E. Stearns in P. M. L. II (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581.

[28] R. Storn in K. Price (1995). Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA.

[29] R. Storn in K. Price (1997). Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11:341–359.

[30] R. Storn in K. Price (2001). Differential evolution homepage. Web site of Price and Storn. `http://www.icsi.berkeley.edu/~storn/code.html`.

[31] TIS (2008). Telefonski imenik slovenije 2008. Telefonski imenik in zemljevid Slovenije. `http://tis.telekom.si`.

[32] H.-F. Wang in K.-Y. Wu (2004). Hybrid genetic algorithm for optimization problems with permutation property. *Comput. Oper. Res.*, 31(14):2453–2471.

[33] R. J. Wilson in J. J. Watkings (1997). *Uvod v teorijo grafov*. DMFA Ljubljana.

[34] L.-N. Xing, Y.-W. Chen in X.-S. Shen (2007). Multiprogramming genetic algorithm for optimization problems with permutation property. *Applied Mathematics and Computation*, 185(1):473–483.

[35] X. Yao, Y. Liu in G. Lin (1999). Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82.

[36] A. Zamuda (2008). Samoprilagajanje krmilnih parametrov pri algoritmu diferencialne evolucije za večkriterijsko optimizacijo. Magistrsko delo, Fakulteta za elektrotehniko, računalništvo in informatiko.

Življenjepis

Ime in priimek:	Štefan Brest	
Rojen:	28. 12. 1983	Murska Sobota
Izobrazba:	1990 - 1998	OŠ Beltinci
	1998 - 2002	Škofijska gimnazija AMS v Mariboru
	2002 - 2009	Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru
Praktično izobraževanje:	Podjetje:	Invind d.o.o. Vipavska cesta 13 5000 Nova Gorica
	Projekt:	C.H.A.I.R. Caring Hiper Able Intelligent Rider Development of speech recognition software for wheelchair users
Kontakt:	Email:	stefan.brest@gmail.com

UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO
2000 Maribor, Smetanova ul. 17

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Štefan Brest, rojen 28.12.1983 v Murski Soboti, študent Fakultete za elektrotehniko, računalništvo in informatiko, Univerze v Mariboru, smer programska oprema, izjavljam, da je diplomsko delo z naslovom

Reševanje problema nesimetričnega trgovskega potnika z diferencialno evolucijo in hevrističnimi algoritmi

pri mentorju red. prof. dr. Viljemu Žumerju, avtorsko delo. V diplomskem delu so uporabljeni viri in literatura korektno navedeni. Teksti niso prepisani brez navedbe avtorjev.

Kraj in datum: Podpis:

Maribor, 26.1.2009

UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO
2000 Maribor, Smetanova ul. 17

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV AVTORJA

Ime in priimek avtorja:	Štefan BREST
Vpisna številka:	93525263
Študijski program:	univerzitetni, Računalništvo in informatika
Naslov zaključnega dela:	Reševanje problema nesimetričnega trgovskega potnika z diferencialno evolucijo in hevrističnimi algoritmi
Mentor:	red. prof. dr. Viljem Žumer

Podpisani Štefan BREST izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal sam ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 16/2007) dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru. Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zagovora, naslov zaključnega dela) na spletnih straneh in v publikacijah UM.

Kraj in datum: Podpis:

Maribor, 26.1.2009