Универзитет у Београду
Математички факултет

Милош Шошић

Хеуристички приступ решавању проблема минималног кашњења

мастер рад

Београд

2014.

Ментор: **проф. др Зорица Станимировић**
Математички факултет у Београду

Чланови комисије: **проф. др Миодраг Живковић**
Математички факултет у Београду

доц. др Мирослав Марић
Математички факултет у Београду

Датум одбране: _____

Хеуристички приступ решавању проблема минималног кашњења

Апстракт:

Проблем минималног кашњења представља варијацију проблема трговачког путника, где је циљ минимизовати суму времена потребног да се посети сваки од чворова, тј. суму кашњења. Овај проблем има широку примену, која укључује дистрибуцију робе корисницима, логистику у кризним ситуацијама, распоређивање послова итд. Циљ овог рада је развој хеуристичке методе за решавање проблема минималног кашњења, заснованој на комбинацији варијанте методе променљивих околина и методе симулираног каљења, чији су елементи прилагођени карактеристикама разматраног проблема. Предложени хибридни алгоритам је тестиран на јавно доступним инстанцама за овај проблем и резултати су упоређени са најбољим решењима из литературе, као и са оптималним решењима, уколико су позната.

Кључне речи: Проблем минималног кашњења, Метода променљивих околина, Симулирано каљење, Хибридизација хеуристика

Heuristic approach to solving Minimum latency problem

Abstract:

Minimum latency problem is a variant of the well-known Traveling Salesman Problem (TSP). The goal of the Minimum latency problem is to minimize the sum of travel times to each vertex, that is, sum of latencies. Real-life applications of Minimum latency problem are numerous, including the delivery of goods, logistics for emergency relief and job scheduling. In this work, a hybrid heuristic method is developed for solving the Minimum latency problem, based on Variable neighborhood search and Simulated annealing. The proposed hybrid algorithm is tested on publicly available instances for this problem and the results are compared with the best solutions from the literature and the optimal solutions, in cases when they are known.

Keywords: Minimum latency problem, Variable neighborhood search, Simulated annealing, Hybrid heuristic method

Садржај

1 Увод

Проблем трговачког путника (TSP – енг. *Traveling Salesman Problem*) је један од највише проучаваних проблема дискретне оптимизације. Овај проблем је први пут математички формулисао ирски математичар В. Р. Хамилтон (*W. R. Hamilton*) 1800-тих и припада класи NP-тешких проблема. TSP има широку примену у многим областима као што је распоређивање послова, производња микрочипова, секвенцирање ДНК, рутирање возила итд.

Нека је $G = (V, A)$ комплетан граф, где је $V = \{0, \dots, n\}$ скуп чворова, тј. градова, а $A = \{(i, j) \mid i, j \in V, i \neq j\}$ скуп грана графа. Нека је $C = [c_{ij}]$ матрица трошкова где c_{ij} може одговарати раздаљини чвора i од чвора j. Циљ проблема трговачког путника је пронаћи најкраћи пут тако да се сваки град посети тачно једном. Ако је матрица C симетрична, тада се ради о симетричном проблему трговачког путника ($sTSP$ – енг. *symmetrical TSP*) а ако постоје индекси i, j тако да $c_{ij} \neq c_{ji}$, тада се решава асиметрични TSP ($aTSP$). Ако важи неједнакост троугла за удаљености између чворова тада се ради о метричком TSP-у.

Егзактни решавачи за проблем трговачког путника, развијани су још од 1950-их година, када је развијена прва целобројна линеарна формулација [15]. Након ње су уследиле и друге, [38], [23], које нису биле самосталне методе него су захтевале и визуелно проверавање решења релаксације проблема. Први самостални алгоритам, објављен је у [40] и [41]. Касније, у [31], решено је оптимално 9 од 10 инстанци TSP-а које садрже по 100 градова, а у [25] димензија решења највеће инстанце износи 2392. Тренутно најбољи егзактни решавач је *Concorde,* представљен у [5], где се може наћи детаљнији опис алгоритма. *Concorde* је успео да реши оптимално 19 од 21 инстанце димензија између 1000 и 2392 за време мање од $3345{,}3$ секунде док је за преостале две требало $13999{,}9$ и $18226404{,}4$ секунде. Највећа инстанца решена овим алгоритмом је димензије 85900 [6].

Како је за решавање чак и инстанци средњих димензија TSP-а егзактним решавачем потребно доста времена, овај проблем је погодан за решавање приближним алгоритмима, тј. хеуристикама. Овај приступ не гарантује оптимално решење, али достиже решење близу оптималном за разумно време израчунавања. Тренутно најбољи приближни алгоритам представљен у [7] гарантује апроксимациони фактор $1 + 1/c$ где је c произвољна вредност. Развијено је пуно хеуристика које достижу решење 2-3% лошије од оптималног. Оне се деле на конструкционе хеуристике (*Nearest Neighbor algorithm* [29]*, Christofides algorithm* [34]), хеуристике које итеративно поправљају решење (2-*opt algorithm* [29], *Lin-Kernighan algorithm* [35]), хеуристике инспирисане

природом (*Ant Colony System* [17]*, Genetic Algorithm* [29]) и друге (*Tabu Search* [29]*, Simulated Annealing* [29]).

Једна од значајнијих варијација проблема TSP је $mTSP$ *(*енг. *multiple TSP)*, где постоји више путујућих трговаца уместо једног, а циљ је пронаћи за сваког трговца Хамилтонов циклус тако да су сви градови посећени тачно једном. Као генерализација проблема TSP, овај проблем има широку примену у штампарству [24], распоређивању екипе [54], планирању мисија [28] и распоређивању возила [46]. Како је $mTSP$ комплексан проблем, било је покушаја његовог егзактног решавања релаксацијом на проблем TSP*,* али без пуно успеха [9]. Међу најзначајнијим хеуристикама, издваја се алгоритам неуронских мрежа [8], табу претрага [21] и [13].

Генерализовани TSP је такође варијација проблема трговачког путника где је разлика у односу на TSP у постојању више група (енг. *cluster*) градова. Циљ је пронаћи подскуп градова H тако да је време обиласка ових градова минималао, при чему важи да сваки град из H припада тачно једној групи. Овај проблем има примену у рутирању авиона [45], достави поште, рутирању возила [32] и другим [18], [19], [33]. Први тачни решавач овог проблема, представљен је у [18] али како је време извршавања на инстанцама димензије 442 са 89 група достизало један дан, изучаване су и хеуристичке методе. Генетски алгоритам је представљен у [53], а друге хеуристике у [48], [51] и [47].

Проблем минималног кашњења (MLP – енг. *Minimum Latency Problem*) је варијација проблема TSP у којој треба минимизовати суму времена потребног да се стигне до сваког града. Циљ овог рада је развијање хибридног алгоритма за приближно решавање проблема MLP, који се базира на варијацији методе променљивих околина и симулираном каљењу.

У поглављу 2 се ближе описује проблем минималног кашњења, у поглављу 3 методе променљивих околина а у поглављу 4 симулирано каљење. У поглављу 5 се описује предложени хибридни алогоритам. Поглавље 6 садржи опис тестирања хибридног алгоритма и представљени су добијени резултати.

2 Проблем минималног кашњења

Нека је $G = (V, A)$ комплетан граф, где је $V = \{0, \dots, n\}$ скуп чворова, тј. локација које треба посетити, а $A = \{(i, j) \mid i, j \in V, i \neq j\}$ скуп грана графа G, где је уз сваку грану познато времене потребно за пут преко те гране. Постоји један истакнути чвор (обично чвор 0) који представља почетни чвор са којег обилазак почиње. Нека је l_i кашњење до i-тог чвора које се израчунава као потребно време да се стигне од истакнутог до чвора i. Циљ проблема минималног кашњења је пронаћи Хамилтонов пут који минимизује суму $\sum_{i=0}^{n} l_i$. У овом раду, сваки обилазак графа почиње са чвором 0 и завршава се када обиђе осталих n чворова. MLP је у литератури познат и по другим именима (енг. *Traveling Repairman Problem, Delivery Man Problem, Cumulative Traveling Salesman Problem, School Bus Driver Problem*).

MLP има широку примену, посебно у дистрибуцији робе корисницима, логистици у кризним ситуацијама, распоређивању послова итд. Код проблема путујућег сервисера, познате су локације клијената и сервисера, као и времена пута између клијената и потребна времена за сервисирање сваког клијента. Потребно је пронаћи пут којим сервисер обилази клијенте тако да је њихово укупно време чекања минимално. Сличан је проблем курира, где курир треба да пронађе пут који минимизује укупно време чекања клијената на доставу пошиљке. Ови проблеми су клијентски орјентисани проблеми рутирања јер функција циља даје предност минимизацији времена чекања клијената у односу на дужину пута возила.

Код распоређивања послова на машини (енг. *Machine Scheduling Context*), овај проблем се јавља у следећем облику. Постоји скуп послова које машина треба да обави као и време које је потребно да се машина поново подеси да ради посао j након завршеног посла i. Ово време представља време пута од чвора i до чвора j. Треба пронаћи пермутацију задатих послова тако да се минимизује потребно време за завршетак свих послова. [4]

2.1 Математичка формулација

Нека је $G = (V, A)$ комплетан граф и нека је дат Хамилтонов пут, тј. пермутација чворова где је чвор 0 на првом месту, који представља неко решење проблема. Нека је $C = [c_{ij}]$ ненегативна матрица трошкова где c_{ij} одговара трошку гране од чвора i до чвора j. Нека $[i]$ представља чвор на i-том месту у датом путу. Кашњење $l_{[i]}$ се може

израчунати као $l_{[i]} = l_{[i-1]} + c_{[i-1][i]}$, где $[0] = 0$ и $l_{[0]} = l_0 = 0$. Функција циља је представљена сумом $z = \sum_{i=0}^{n} l_{[i]}$, а како важи

$l_{[1]} = l_{[0]} + c_{[0][1]}$

$l_{[2]} = l_{[1]} + c_{[1][2]} = c_{[0][1]} + c_{[1][2]}$

…

$l_{[n]} = l_{[n-1]} + c_{[n-1][n]} = c_{[0][1]} + c_{[1][2]} + \cdots + c_{[n-1][n]}$

тако је

$$z = \sum_{i=0}^{n} l_{[i]} = nc_{[0][1]} + (n-1)c_{[1][2]} + \cdots + 2c_{[\mathrm{n}-2][\mathrm{n}-1]} + c_{[\mathrm{n}-1][\mathrm{n}]}.$$

Код проблема курира и путујућег сервисера, вредност c_{ij} чине време пута t_{ij} и време сервисирања s_i, тј.:

$$c_{ij} = \begin{cases} t_{0j}, & \text{за } i = 0,\ j = 1, \dots, n \\ s_i + t_{ij}, & \text{за } i = 1, \dots, n,\ j = 1, \dots, n,\ i \neq j \end{cases}$$

У овом раду, решава се проблем где је матрица C симетрична, тј. дужина пута између два чвора је иста у оба смера.

У свакој пермутацији чворова, Хамилтонов пут почиње од чвора 0 и обилази остале чворове по реду у којем је дата пермутација. Ако грана (i, j) спаја k-тог са $k+1$-им чланом пермутације, тада је допринос те гране функцији циља $(n-k)c_{[k][k+1]} = (n-k)c_{ij}$.

Уводимо следеће променљиве одлучивања:

$$x_i^{(k)} = \begin{cases} 1, & \text{ако је чвор } i \text{ на позицији } k \\ 0, & \text{иначе} \end{cases}$$

$$y_{ij}^{(k)} = \begin{cases} 1, & \text{ако је чвор } i \text{ на позицији } k \text{ а чвор } j \text{ на позицији } k+1 \\ 0, & \text{иначе} \end{cases}$$

Променљиве $x_i^{(1)}, (i = 1, \dots, n)$ се користе у функцији циља при израчунавању кашњења од чвора 0 до позиције 1, тј. чвора i, а променљиве $y_{ij}^{(k)}, (i = 1, \dots, n,\ j = 1, \dots, n,\ i \neq j, k = 1, \dots, n-1)$ за рачунање кашњења осталих чворова.

Имајући у виду горе наведену нотацију, проблем минималног кашњења се може формулисати као проблем линеарног целобројног програмирања на следећи начин [4]:

$$\min z = n \sum_{i=1}^{n} c_{0i} x_i^{(1)} + \sum_{k=1}^{n-1} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} (n-k) c_{ij} y_{ij}^{(k)} \qquad (2.1)$$

при ограничењима:

$$\sum_{k=1}^{n} x_i^{(k)} = 1, \qquad i = 1,2,\dots,n \tag{2.2}$$

$$\sum_{i=1}^{n} x_i^{(k)} = 1, \qquad k = 1,2,\dots,n \tag{2.3}$$

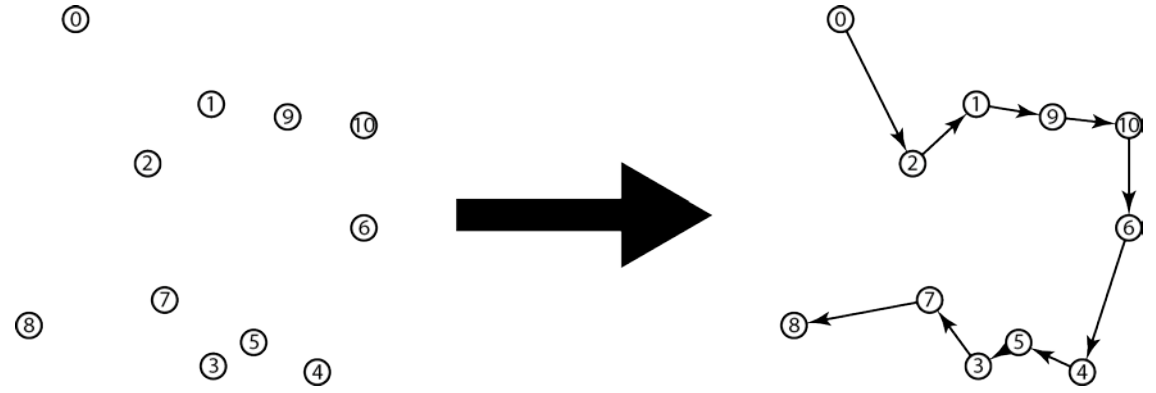
$$\sum_{\substack{j=1 \\ j\neq i}}^{n} y_{ij}^{(k)} = x_i^{(k)}, \qquad i = 1,2,\dots,n-1, \;\; k = 1,2,\dots,n-1 \tag{2.4}$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n-1} y_{ji}^{(k)} = x_i^{(k+1)}, \qquad i = 1,\dots,n, \;\; k = 1,2,\dots,n-1 \tag{2.5}$$

$$x_i^k \in \{0,1\}, \qquad i = 1,\dots,n, \;\; k = 1,\dots,n \tag{2.6}$$

$$y_{ij}^{(k)} \in \{0,1\}, \qquad i = 1,\dots,n, \;\; j = 1,\dots,n, \;\; j \neq i, \;\; k = 1,\dots,n-1 \tag{2.7}$$

Ограничење (2.2) обезбеђује да се сваки чвор налази на тачно једној позицији у пермутацији, (2.3) је услов да се на свакој позицији налази тачно један чвор. Условом (2.4) захтева се да са сваке позиције води само једна грана ка следећем чвору а (2.5) да на сваку позицију само једна грана води са претходног чвора. Ограничења (2.6) и (2.7) односе се на бинарну природу променљивих одлучивања.



Слика 1. Пример инстанце са 11 чворова и оптималан пут

Проблем минималног кашњења се може илустровати следећим примером. Нека је дата инстанца проблема минималног кашњења са 11 чворова, где је матрица $C = [c_{ij}]$ дата табелом 1.

Табела 1. Пример матрице трошкова C за инстанцу са слике 1.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	37	38	86	100	86	83	69	72	55	72
1	37	0	20	59	67	57	46	47	67	18	36
2	38	20	0	48	63	48	53	32	47	34	51
3	86	59	48	0	21	4	43	19	47	57	62
4	100	67	63	21	0	18	35	39	68	60	59
5	86	57	48	4	18	0	39	21	51	54	59
6	83	46	53	43	35	39	0	49	82	31	24
7	69	47	32	19	39	21	49	0	32	51	62
8	72	67	47	47	68	51	82	32	0	78	91
9	55	18	34	57	60	54	31	51	78	0	18
10	72	36	51	62	59	59	24	62	91	18	0

На слици 1 може се видети распоред чворова и оптималан пут за ту инстанцу. Кашњења се рачунају као трошак пута до чворова, па тада:

$$l_{[0]} = 0$$

$$l_{[1]} = c_{[0][1]} = c_{0\,2} = 38$$

$$l_{[2]} = l_{[1]} + c_{[1][2]} = l_{[1]} + c_{2\,1} = 58$$

$$l_{[3]} = l_{[2]} + c_{[2][3]} = l_{[2]} + c_{1\,9} = 76$$

$$l_{[4]} = l_{[3]} + c_{[3][4]} = l_{[3]} + c_{9\,10} = 94$$

$$l_{[5]} = l_{[4]} + c_{[4][5]} = l_{[4]} + c_{10\,6} = 118$$

$$l_{[6]} = l_{[5]} + c_{[5][6]} = l_{[5]} + c_{6\,4} = 153$$

$$l_{[7]} = l_{[6]} + c_{[6][7]} = l_{[6]} + c_{4\,5} = 171$$

$$l_{[8]} = l_{[7]} + c_{[7][8]} = l_{[7]} + c_{5\,3} = 175$$

$$l_{[9]} = l_{[8]} + c_{[8][9]} = l_{[8]} + c_{3\,7} = 194$$

$$l_{[10]} = l_{[9]} + c_{[9][10]} = l_{[9]} + c_{7\,8} = 226.$$

Тада функција циља има вредност:

$$z = \sum_{i=0}^{n} l_{[i]} = 0 + 38 + 58 + 76 + 94 + 118 + 153 + 171 + 175 + 194 + 226 = 1303$$

што представља оптимално решење за инстацу са слике 1.

2.2 Постојеће методе за решавање проблема минималног кашњења

Како проблем минималног кашњења припада класи NP-тешких проблема [49], егзактне методе могу решите само инстанце мањих димензија. Први такав алгоритам [36], био је енумерациони, заснован на формулацији нелинеарног целобројног програмирања, користећи Лагранжову релаксацију за добијање доњих ограничења. У том раду, решене су инстанце са мање од 30 чворова. Следи [20] у којем је предложена формулација целобројног програмирања а алгоритам за решавање је гранање са ограничавањем (енг. *branch-and-bound*) користећи матроидне структуре проблема за добијање доњих ограничења. Решене су инстанце до 60 чворова. Касније, у [39] је изложена формулација целобројног програмирања која користи предности везе са проблемом линеарног поретка где је представљен алгоритам одсецања равни који користи валидне неједнакости у тој формулацији. Два рада, [2] и [3], представила су *branch-cut-and-price* приступ којим су решене инстанце до 107 чворова. Ово је тренутно једини егзактни алгоритам који решава инстанце до ових димензија.

Проблем минималног кашњења је решаван и приближним алгоритмима. Први овакав алгоритам [10] имао је апроксимациони фактор 144. Тренутно најбољи приближни алгоритам [14] има апроксимациони фактор 3.59.

Не постоји пуно радова у којима је MLP решаван помоћу метахеуристика. Један од њих је [50], где је коришћена метахеуристика која спаја GRASP методу и методу променљивих околина. За проблем минималног кашњења са профитом, развијена је табу претрага у [16]. Варијација проблема рутирања возила [44] може да се прилагоди тако да решава MLP. У том раду, описан је меметски алгоритам као и ефикасна процедура провере новог потеза у околини са $O(1)$ операција. Тренутно најзначајнији рад у којем се користи метахеуристика за решавање MLP-а је [52] у којем се спајају метода GRASP, итеративна локална претрага и варијација методе променљивих околина. Метахеуристика предложена у [52] достиже оптимална решења за инстанце до 107 чворова где су оптимална решења позната, а за веће димензије постиже најбоља решења тренутно позната.

У неким од поменутих радова, нпр. [52], уместо Хамилтоновог пута треба пронаћи Хамилтонов циклус, тј. после обиласка свих чворова одлази се до чвора 0.

3 Метода променљивих околина

Методе комбинаторне оптимизације које користе локалну претрагу долазе до бољих решења итеративно примењујући локалне промене над тренутно најбољим решењем све док се не достигне локални оптимум. Ове локалне промене припадају околини $N(x)$ тренутног решења x. Код оваквих метода може се доћи до стања када алгоритам остане у неком локалном и не достигне глобални оптимум, тј. оптимално решење проблема. Ово стање се тада решава на различите начине механизмом који се назива пертурбација.

Метода променљивих околина (VNS – енг. *Variable Neighborhood Search*) је метахеуристика представљена у [42], која се базира на методама локалне претраге, где се користи систематска промена структуре околина како би се побољшало тренутно решење у процесу локалне претраге и пертурбације. Ова метода је осмишљена за приближно решавање проблема комбинаторне оптимизације али је временом проширена и корисити се при решавању проблема целобројног, мешовитог целобројног, нелинеарног програмирања и других. Стога, неке од области примене VNS методе су локацијски проблеми, проблеми распоређивања и упаривања, проблеми рутирања возила, рачунарске мреже, и многи други (за детаљнији преглед области примене, погледати [26]).

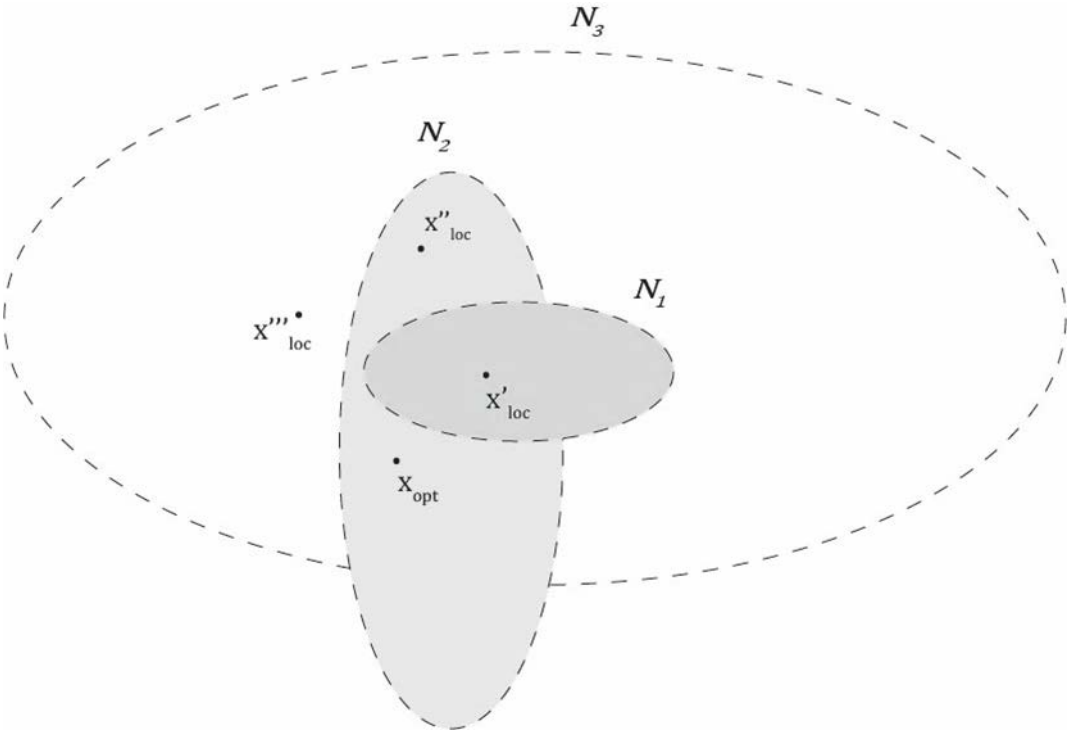
За проблеме који се заснивају на проблему трговачког путника, VNS алгоритам и његове варијације су постигли пуно успеха. За проблем TSP са временским ограничењима ($TSPTW$ – енг. *TSP with time windows*), тренутно најбоља решења су постигнута у [43], користећи генерализовани VNS. За проблем генерализованог TSP-а, такође тренутно најбољи приближни алгоритам за велике инстанце је варијација методе променљивих околина, изложена у [27]. Проблем трговачког путника са достављањем добара $FIFO$ политиком (енг. *pickup and delivery TSP with $FIFO$ loading*) је до сад најуспешније решен варијацијом истог алгоритма у [12]. Следи детаљнији опис VNS метахеуристике [26].

Нека је S дати простор решења и $f: S \to R$ функција циља за дати проблем. Нека су $N_k (k = 1, \dots, k_{max})$ пажљиво одабране структуре околина а $N_k(x)$ скуп решења из k-те околине решења x, где $x \in S$. Означимо оптимално решење са x_{opt} (глобални минимум). У околини N_k, x' је локални минимум ако не постоји $x \in N_k(x')$ тако да важи $f(x) < f(x')$. Да би различите околине биле добро дефинисане, треба да важи следеће:

1) Локални минимум једне околине не мора бити локални минимум друге околине

2) Глобални минимум је локални минимум свих могућих околина
3) У многим проблемима важи, локални минимуми различитих околина су релативно близу.

Последња претпоставка подразумева да локални минимум често пружа корисне информације о глобалном минимуму. На пример, може се десити да неке од променљивих узимају исте вредности у оба решења али како се не може утврдити које су те променљиве у току извршавања алгоритма, примењује се пажљива претрага околине локалног минимума у потрази за бољим решењем. Пример структура околина може се видети на слици 2.



Слика 2. Пример структура околина.

На слици 2, N_1, N_2, N_3 представљају различите околине а x'_{loc}, x''_{loc}, x'''_{loc} респективно њихове локалне оптимуме. x_{opt} представља глобални оптимум.

Основни VNS алгоритам састоји се из две фазе које се извршавају у свакој итерацији. Прва фаза је „размрдавање“ где се прави случајни потез у тренутној околини након чега следи друга фаза, локална претрага која налази локални минимум за текућу околину.

Алгоритам 1. Процедура $\boldsymbol{VNS}()$

1. **иницијализација:** конструисати почетно решење x, избор структура околина $N_k(k \leftarrow 1, \dots, k_{max})$, изабрати критеријум заустављања
2. **while** критеријум заустављања није испуњен
3. $k \leftarrow 1$

4. while $k \neq k_{max}$

5. **размрдавање:** случајним потезом у околини $N_k(x)$ бира се тачка x'

6. **локална претрага:** претражује се околина $\mathrm{N_k(x')}$ тачке $\mathrm{x'}$ и налази локални оптимум $\mathrm{x''}$

7. **промена решења:** ако је $f(x'') < f(x)$ тада $x \leftarrow x''$ и $k \leftarrow 1$, иначе $k \leftarrow k + 1$

8. **излаз:** x

Почетно решење може бити било које допустиво решење, обично добијено неким похлепним или алгоритмом са случајним одабиром променљивих одлучивања. Критеријум заустављања може бити утрошено процесорско време, максимални број итерација, максимални број итерација без побољшања решења итд. Локална претрага у овом алгоритму може подразумевати претрагу за најбољим решењем у тренутној околини (енг. *best-improvement*) или претрагу за првим бољим решењем (енг. *first-improvement)*. Уместо локалне претраге може се користити и било која друга s-хеуристика.

Једна значајна процедура локалне претраге са променљивим околинама је метода променљивог спуста (VND – енг. *Variable Neighborhood Descent*) дата следећим алгоритмом.

Алгоритам 2. Процедура $\boldsymbol{VND}(\boldsymbol{x})$

1. **иницијализација:** избор структура $N_k(k \leftarrow 1, \dots, k_{max})$

2. $k \leftarrow 1$

3. while $k \neq k_{max}$

4. **локална претрага:** претражује се околина $N_k(x)$ тачке x и налази локални оптимум x'

5. **промена решења:** ако је $f(x') < f(x)$ тада $x \leftarrow x'$ и $k \leftarrow 1$ иначе $k \leftarrow k + 1$

6. **излаз:** x

Ако се ова метода користи у другој фази алгоритма VNS, добија се генерализовани VNS ($GVNS$ – енг. *General VNS*) који се за неке проблеме комбинаторне оптимизације истакао као најефикаснија метода за приближно решавање. У процедури VND се тада користе исте структуре околина као из дела иницијализације алгоритма $GVNS$.

Алгоритам 3. Процедура $\boldsymbol{GVNS}()$

1. **иницијализација :** конструисати почетно решење x, избор структура околина $N_k(k \leftarrow 1, \dots, k_{max})$, изабрати критеријум заустављања

2. while критеријум заустављања није испуњен

3. $k \leftarrow 1$

4. while $k \neq k_{max}$

5. размрдавање: случајним потезом у $N_k(x)$ бира се x'

6. локална претрага: $x'' \leftarrow VND(x')$

7. промена решења: ако је $f(x'') < f(x)$ тада $x \leftarrow x''$ и $k \leftarrow 1$, иначе $k \leftarrow k + 1$

8. **излаз:** x

4 Симулирано каљење

Симулирано каљење (SA – енг. *Simulated Annealing*) је веома широко проучавана метахеуристика која припада групи метода локалне претраге. Користи се при решавању дискретних и у мањој мери континуалних оптимизационих проблема. Главна одлика ове методе је то што дозвољава решењу да постане лошије како би се извршила пертурбација и избегла прерана конвергенција алгоритма ка неком локалном оптимуму.

Мотивација за ову метахеуристику је проистекла из процеса каљења метала у металургији. Метал се загрева и контролисано хлади како би се достигла најбоља конфигурација кристала и избегли дефекти. На овај начин, ако је погодно изабрана шема хлађења, метал који се на крају добије има побољшану структуру веза између атома и повољне особине за даљу обраду.

За дати проблем комбинаторне оптимизације, нека је S простор решења, $f: S \rightarrow R$ функција циља и нека је са $N(x)$ дефинисана околина тачке $x \in S$. Треба пронаћи x' тако да важи $f(x') \leq f(x)$ за свако $x \in S$. Симулирано каљење у свакој итерацији упоређује два решења, тренутно, x, и решење из његове околине, x', добијено на случајан начин. Уколико је x' боље решење, тренутно решење постаје x', а ако је лошије, x' се прихвата са одређеном вероватноћом која зависи од параметра алгоритма T. Овај параметар представља тренутну температуру процеса каљења и смањује се у току алгоритма према унапред одређеној шеми хлађења која подразумева скуп температура t_k $(k = 1, \dots, n)$. За свако t_k, бира се број итерација M_k на тој температури. Најзначајнији део алгоритма је вероватноћа прихватања решења x' која се израчунава на следећи начин:

$$P(\omega) = \begin{cases} 1, & f(x') - f(x) < 0 \\ e^{-\frac{f(x') - f(x)}{T}}, & f(x') - f(x) \geq 0 \end{cases} \tag{4.1}$$

где је ω догађај прихватања решења x'.

Алгоритам 4. Процедура $SA()$

1. **иницијализација:** изабрати почетно решење, критеријум заустављања, шему хлађења t_k и M_k
2. $k \leftarrow 1$
3. **while** критеријум заустављања није испуњен
4. $m \leftarrow 0$

5. while $m \neq M_k$

6. изабрати $x' \in N(x)$ на случајан начин

7. $\Delta_{x,x'} \leftarrow f(x') - f(x)$

8. **промена решења:** ако је $\Delta_{x,x'} < 0$ онда $x \leftarrow x'$,
иначе $x \leftarrow x'$ са вероватноћом $e^{-\frac{\Delta_{x,x'}}{T}}$

9. $m \leftarrow m + 1$

10. $k \leftarrow k + 1$

11. излаз: x

где критеријум заустављања може бити исти као и код VNS алгоритма.

При већој температрути већа је и вероватноћа прихватања лошијег решења а како температура опада, та вероватноћа постаје све мања. На тај начин се на почетку избегава прерана конвергенција методе и омогућава претрага различитих ланаца решења док се на крају налази локални минимум прихватањем искључиво бољих решења. Успех методе у великој мери зависи од шеме хлађења па ову шему треба пажљиво конструисати.

Неки од радова где је описано симулирано каљење за решавање проблема трговачког путника су [30] и [1] где је представљена анализа различитих шема хлађења, а паралелни алгоритам симулираног каљења је развијен у [37].

Више о симулираном каљењу и доказу конвергенције методе ка глобалном оптимуму може се наћи у [22].

5 Предложени хибридини алгоритам за проблем минималног кашњења

Идеја хибридних метахеуристика постоји већ дуго времена иако је термин „хибридизација“ прихваћен тек у последњих десетак година [11]. Под овим термином подразумева се комбиновање различитих метахеуристка у сложенији алгоритам. Успешна хибридизација представља спој добрих особина метахеуристика које се користе у алгоритму, како би се постигла квалитетнија решења или ефикаснији алгоритам за дати проблем.

За проблем минималног кашњења хибридни алгоритам који ће бити представљен користи генерализовану методу променљивих околина као основни алгоритам са измењеном фазом локалне претраге. Ова фаза се састоји из комбиновања методе променљивог спуста са методом симулираног каљења. Идеја је да у почетку алгоритам дође до доброг решења користећи методу променљивог спуста као локалну претрагу, а онда, уместо ње се користи метода симулираног каљења да би се дошло до квалитетнијег решења. Алгоритам је дат псеудокодом:

Алгоритам 5. Процедура $\boldsymbol{GVNS\_SA}()$

1. иницијализација : конструисати почетно решење x, избор структура околина $N_k(k \leftarrow 1, \dots, k_{max})$, изабрати критеријум заустављања, максимални број итерација MAX_{ITER}, параметар α, шему хлађења t_k i M_k
2. $iter \leftarrow 1$
3. while $iter < MAX\_ITER$
4. $k \leftarrow 1$
5. while $k \neq k_{max}$
6. размрдавање: случајним потезом у околини $N_k(x)$ бира се тачка x'
7. локална претрага: ако је $iter < \alpha$ тада $x'' \leftarrow VND(x')$, иначе $x'' \leftarrow SA(x')$
8. промена решења: ако је $f(x'') < f(x)$ тада $x \leftarrow x''$ i $k \leftarrow 1$, иначе $k \leftarrow k + 1$
9. $iter \leftarrow iter + 1$
10. излаз: x

У овом алгоритму се при позивању процедуре SA не конструише изнова почетно решење већ се за њено почетно узима решење x'.

Параметар MAX_{ITER} зависи од дате инстанце и износи $(\text{број чворова})/4$. Параметар α представља број итерација после којег треба започети примењивање симулираног каљења уместо методе VND. Емпиријски је закључено да добра вредност за $\propto$ износи $MAX_{ITER}/2$. Ова вредност доноси добар однос квалитета решења и брзине извршавања.

5.1 Кодирање и простор решења

Решење за MLP је Хамилтонов пут па се решење природно кодира низом бројева. Ови бројеви представљају индексе чворова у том путу а њихов редослед одговара редоследу обиласка чворова. Простор решења S је скуп свих пермутација бројева из скупа $\{0, \dots, n\}$ које почињу бројем 0 (јер је први чвор фиксиран). Пример једног решења за инстанцу MLP-а где важи $n = 10$ дат је на слици 3.

0	2	1	9	10	6	4	5	3	7	8

Слика 3. Пример, кодирање решења са слике 1.

Нотација за представљање руте решења биће, за пример са слике 3, $x = [0, 2, 1, 9, 10, 6, 4, 5, 3, 7, 8]$.

5.2 Конструкција почетног решења

Почетно решење у фази иницијализације алгоритма $GVNS$ је похлепна хеуристика (NNH – енг. *Nearest Neighbor Heuristic*) која конструише руту чвор по чвор. Почиње се од чвора 0 и сваки наредни чвор је онај који је најближи последњем чвору у тренутном путу. Овај алгоритам даје боље почетно решење од избора руте на случајан начин па се зато користи у овој имплементацији, а дат је следећим псеудокодом:

Алгоритам 6. Процедура $\boldsymbol{NNH}()$

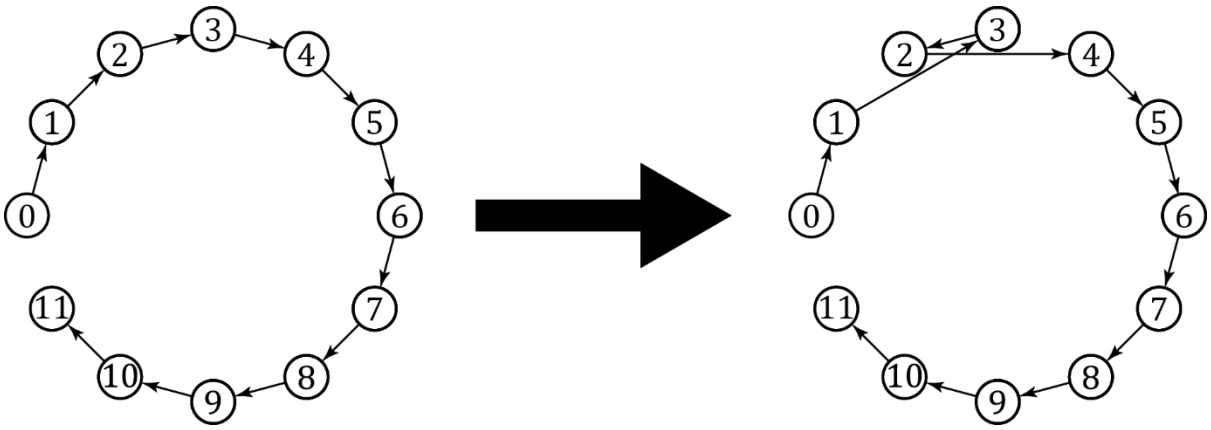
1. **иницијализација:** рута $x \leftarrow [0]$
2. $k \leftarrow 1$
3. **while** $k \leq n$
4. рути x се додаје чвор c, који је најближи од преосталих последњем чвору у рути
5. $k \leftarrow k + 1$
6. **излаз:** x

5.3 Структуре околина

Како би метода променљивих околина постизала квалитетна решења, потребно је пажљиво одабрати скуп структура околина. У овом раду те структуре су добро познате околине из литературе за проблем трговачког путника. Околина решења је скуп решења која се могу добити применом процедуре која дефинише ту околину. Процедуре које индукују редом околине N_1, N_2, N_3, N_4 и N_5 за изложени хибридни алгоритам, биће изложене по величини околина које конструишу, почевши од најмање.

5.3.1 Процедура $swap\mathrm{Two}$

Процедура $swapTwo$ замењује места два суседна чвора у датом путу решења. Како би се претражила цела $swapTwo$ околина решења потребно је извршити $O(n)$ операција.

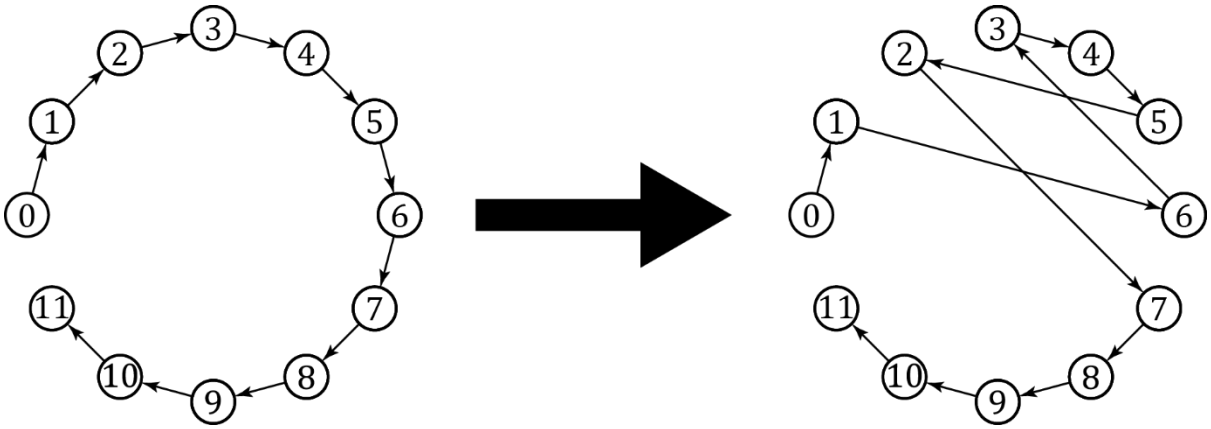


Слика 4. Пример извршавања процедуре swapTwo.

На слици 4 се рута $\mathrm{x} = [0,1,2,3,4,5,6,7,8,9,10,11]$ се процедуром $swapTwo(2)$ трансформише у руту $\mathrm{x}' = [0,1,3,2,4,5,6,7,8,9,10,11]$. Тиме се мењају позиције чворова 2 и 3.

5.3.2 Процедура $swap$

$swap$ процедура узима као аргумент два индекса и замењује места чворовима под тим индексима у тренутној рути. $swap$ околина се претражује извршавајући $swap$ процедуру над сваком комбинацијом два индекса. Ово је уопштење процедуре $swapTwo$ и сложеност претраге околине дефинисане овом операцијом је $O(n^2)$.



Слика 5. Пример извршавања процедуре swap.

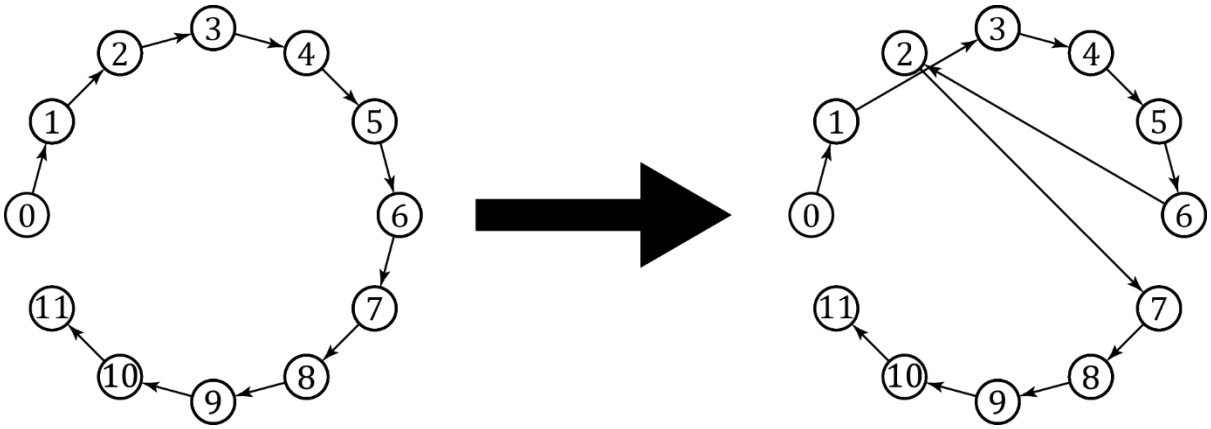
На слици 5 се рута $\mathrm{x} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$ процедуром $\mathrm{swap}(2, 6)$ трансформише у руту $\mathrm{x}' = [0, 1, 6, 3, 4, 5, 2, 7, 8, 9, 10, 11]$, чиме се мењају позиције чворова 2 и 6.

5.3.3 Процедура $removeInsert$

Ова процедура такође зависи од два аргумента, индекс места у рути са којег треба уклонити чвор, i, и индекс места где тај чвор треба сместити, j. За испитивање целе $removeInsert$ околине, треба извршити $O(n^3)$ операција због промене позиције чворова на позицијама између i и j.

Алгоритам 7. Процедура $\boldsymbol{removeInsert(i, j)}$

1. $temp \leftarrow x_i$
2. $k \leftarrow i$
3. **while** $k < j$
4. $\quad x_k \leftarrow x_{k+1}$
5. $\quad k \leftarrow k + 1$
6. $x_j \leftarrow temp$
7. **излаз:** x

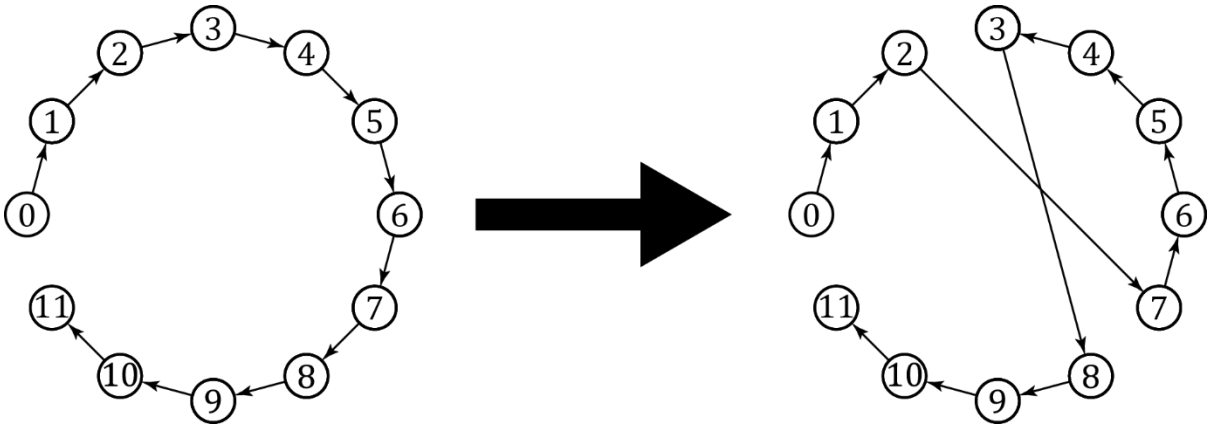


Слика 6. Пример извршавања процедуре $removeInsert$.

На слици 6 рута $\mathrm{x} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$ се процедуром $removeInsert(2, 6)$ трансформише у руту $\mathrm{x}' = [0, 1, 3, 4, 5, 6, 2, 7, 8, 9, 10, 11]$. Овиме се чвор 2 уклања са старог места у рути и поставља на ново, између чворова 6 и 7.

5.3.4 Процедура 2-opt

2-opt је нешто сложенија процедура која се састоји из уклањања две гране из датог пута и додавања две нове гране. Овој операцији треба проследити два параметра, тј. индексе, полазних чворова грана које се уклањају из тренутне руте. Начин додавања две нове гране приказан је на слици 7. 2-opt захтева промену смера обиласка дела пута између две гране које се уклањају па је сложеност претраге целе околине $O(n^3)$.



Слика 7. Пример извршавања процедуре 2-opt.

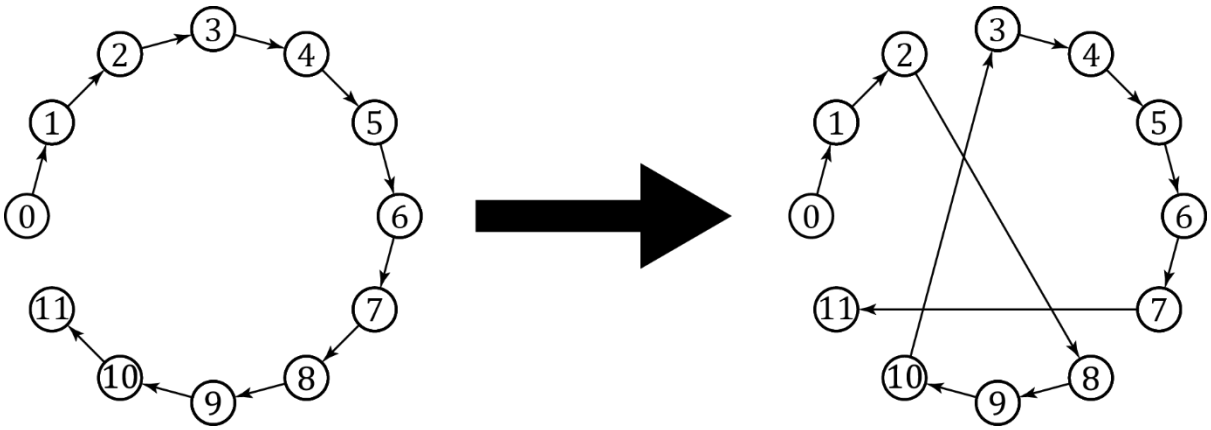
Рута $\mathrm{x} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$ се на слици 7 процедуром $2\text{-opt}(2, 7)$ трансформише у руту $\mathrm{x}' = [0, 1, 2, 7, 6, 5, 4, 3, 8, 9, 10, 11]$. Тада се уклањају гране између чворова 2 и 3 и између 7 и 8, сегмент руте између чворова 3 и 7 мења смер и додају се гране између чворова 2 и 7 и између 3 и 8.

Алгоритам 8. Процедура $2\text{-}opt(i, j)$

1. $l \leftarrow i + 1$
2. $r \leftarrow j$
3. **while** $l < r$
4. $temp \leftarrow x_l$
5. $x_l \leftarrow x_r$
6. $x_r \leftarrow temp$
7. $l \leftarrow l + 1$
8. $r \leftarrow r - 1$
9. **излаз:** x

5.3.5 Процедура *or-opt*

Процедура o$r\text{-}opt$ уклања три гране из тренутне руте и додаје три нове гране тако да је резултујући пут допустив. Параметри који се прослеђују су индекси i, j и k, полазних чворова три гране. Сложеност претраге ове околине је $O(n^4)$ због премештања чворова између грана које се уклањају.



Слика 8. Пример извршавања операције o$r\text{-}opt$.

На слици 8 се рута $x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$ процедуром $or\text{-}opt(2, 7, 10)$ трансформише у руту $x' = [0, 1, 2, 8, 9, 10, 3, 4, 5, 6, 7, 11]$. Уклањају се гране између чворова 2 и 3, 7 и 8, 10 и 11, а додају се гране између чворова 2 и 8, 10 и 3, 7 и 11.

Процедура $or\text{-}opt$ мања руту тренутног решења тако што замењује место сегмента почевши од x_{i+1} до x_j и сегмента од x_{j+1} до x_k.

5.4 Динамичко рачунање функције циља

За проблем трговачког путника, израчунавање функције циља после промене у некој околини захтева константан број операција. Ово није случај код MLP-а где се промена једног чвора одражава на све чворове који се налазе након њега у рути. Стога, са директним рачунањем функције циља, сложеност претраге околине расте за мултипликативни фактор $O(n)$ (на пример, за претрагу $swap$ околине, са циљем проналасна бољег решења, потребно је $O(n^3)$ операција). Овакво директно рачунање захтева промену решења како би било могуће. Да би се избегло директно израчунавање функције циља, у [44] је предложена провера потеза у околини (тј. провера да ли је ново решење боље од старог) која има амортизовану временску сложеност $O(1)$ и које нема потребе за променом решења како би се израчунала функција циља. У овом раду се предлаже сличан механизам провере квалитета новог решења, заснован на [52].

Механизам испитивања новог решења састоји се из спајања сегмената на које се рута раставља приликом потеза у околини, при чему се користе глобалне особине тренутног пута и тих сегмената. Потребно је дефинисати ове глобалне податке како би се могао испитати сваки нови потез у некој околини.

Нека је $\sigma = (\sigma_1, \dots, \sigma_k)$ сегмент руте. Тада се уводе:

- $T(\sigma)$ – представља време потребно за обилазак сегмента σ
- $C(\sigma)$ – представља цену обиласка сегмента σ у смислу збира кашњења
- $R(\sigma)$ – представља број чворова у сегменту σ

За семент σ који се састоји од једног чвора важи $T(\sigma) = 0$, цена обиласка $C(\sigma) = 0$ и $R(\sigma) = 0$ ако је чвор 0-ти, иначе $R(\sigma) = 1$.

Нека је $\sigma = (\sigma_1, \dots, \sigma_k)$ и $\sigma' = (\sigma'_1, \dots, \sigma'_k)$. За спајање два сегмента важе правила:

$$T(\sigma \oplus \sigma') = T(\sigma) + c_{[\sigma_k][\sigma'_1]} + T(\sigma') \tag{5.1}$$

$$C(\sigma \oplus \sigma') = C(\sigma) + \mathrm{R}(\sigma')\left(T(\sigma) + c_{[\sigma_k][\sigma'_1]}\right) + C(\sigma') \tag{5.2}$$

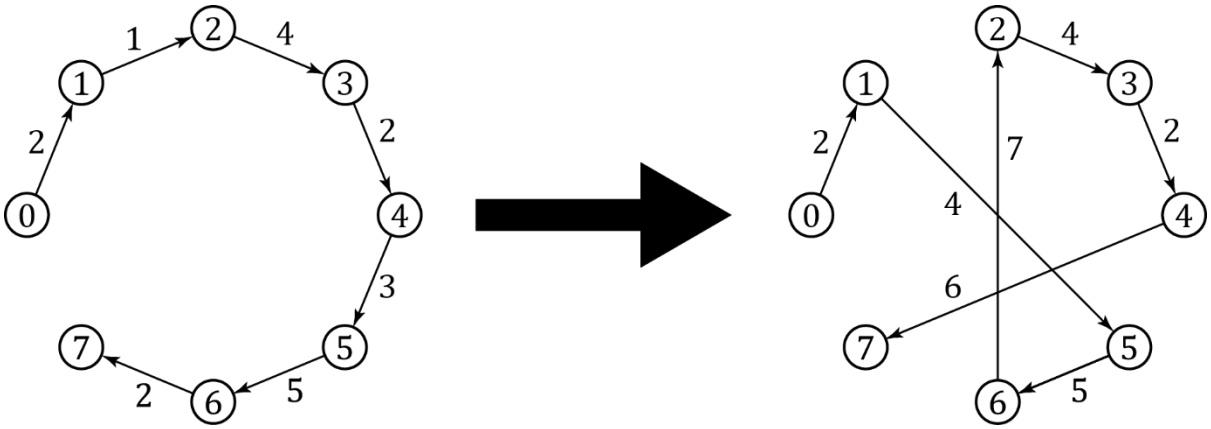
$$R(\sigma \oplus \sigma') = R(\sigma) + \mathrm{R}(\sigma') \tag{5.3}$$

где је $\oplus$ оператор спајања, а $c_{[\sigma_k][\sigma'_1]}$ трошак пута од чвора на позицији σ_k у рути до чвора на позицији σ'_1. Ако је сегмент σ' сегмент σ у обрнутом смеру, тада $T(\sigma) = T(\sigma')$, $R(\sigma) = \mathrm{R}(\sigma')$ а $C(\sigma') = kT(\sigma) - C(\sigma)$, што омогућава лако спајање сегмената који промене смер што се користи у $2\text{-}opt$ околини.

Како би било могуће реализовати наведено динамичко испитивање новог решења, потребно је извршити предпроцесирање уведених глобалних података за сваки сегмент тренутног пута. Ова операција има сложеност $O(n^2)$ и извршава се након сваке промене решења. Такође се може приметити да тада нема потребе за променом

решења како би се одредила његова функција циља (а тиме одредило да ли је боље од тренутног решења), већ се та вредност може израчунати користећи правила (5.1)-(5.3) над одговарајућим сегментима тренутне околине у константном времену. Ово смањује сложеност претраге околина $removeInsert$ и $2\text{-}opt$ на $O(n^2)$ а $or\text{-}opt$ околине на $O(n^3)$ што значајно умањује време извршавања хибридног алгоритма.

Следећи пример илуструје динамичку проверу цене новог решења. На слици 9 је дата инстанца са 8 чворова, њено почетно решење и изглед новог решења које се добија операцијом $or\text{-}opt$, заједно са временима потребним за обилазак грана на путу.



Слика 9. Рута $x = [0, 1, 2, 3, 4, 5, 6, 7]$ и решење у $or\text{-}opt$ околини x.

Да би се искористило динамичко рачунање функције циља новог решења са слике 9, потребно је спојити сегменте $(0, 1)$, $(5, 6)$, $(2, 3, 4)$ и (7), а потребно је и познавање већ поменутих глобалних информација о овим сегментима. Сада се могу извршити спајања и може се проверити цена без претходне промене почетног решења. Овај процес, који користи правила (5.1)-(5.3), као и глобални подаци, дати су у следећој табели.

Табела 2. Пример спајања сегмената.

σ	$T(\sigma)$	$C(\sigma)$	$R(\sigma)$
$(0, 1)$	2	2	1
$(5, 6)$	5	5	2
$(2, 3, 4)$	6	10	3
(7)	0	0	1
$(0, 1) \oplus (5, 6)$	11	19	3
$(0, 1) \oplus (5, 6) \oplus (2,3,4)$	24	83	6
$(0, 1) \oplus (5, 6) \oplus (2,3,4) \oplus (7)$	30	**113**	7

Када се израчуна цена новог решења, може се закључити да ли је боље од старог и према томе извршити ажурирање решења.

5.5 Шема хлађења

У имплементацији хибридног алгоритма, температура T се на почетку симулираног каљења иницијализује на вредност $T_{MAX} = \lfloor n * 6 \rfloor$ где је n број чворова инстанце која се решава. Параметар T се мења у свакој итерацији геометријском прогресијом са фактором $\gamma = 0{,}9$ док не достигне минималну температуру $T_{MIN} = \lfloor n/2 \rfloor$. Број итерација на свакој температури износи $M_{MAX} = \frac{\lfloor n*6 \rfloor}{T}$ па број итерација на почетку износи 1 а при крају извршавања алгоритма SA око 12. Ови параметри су добијени емиријски.

6 Експериментална анализа

Хибридни алгоритам изложен у овом раду, имплементиран је користећи програмски језик *C++,* у *Visual Studio 2013* окружењу на *Windows 8.1 Pro 64bit* оперативном систему. Тестирање је извршено на *Intel i7-3770k @4.0Ghz* процесору са *16GB RAM*-а, а при извршавању коришћен је само један *thread*.

Алгоритам је тестиран на два скупа инстанци. Први скуп, одабран у [2], је димензија инстанци од 42 до 107 чворова и припада јавно доступној библиотеци $TSPLIB$ за проблем трговачког путника. Инстанце ове библиотеке представљају стварне примере географског распореда градова из разних држава, а користе се као тест инстанце за упоређивање алгоритама који решавају варијације проблема трговачког путника као и других сличних проблема. Други скуп представља инстанце генерисане на случајан начин, представљене у [50] где су и тестиране. У овом раду из другог скупа тестираће се инстанце димензија $10, 30, 50, 100$ и 200 где су координате чворова из униформне расподеле $U[1, 100]$ и инстанце димензије 500 чије су координате чворова из $U[1, 500]$. Свака група различите димензије има по 20 инстанци.

6.1 Резултати добијени предложеним хибридним алгоритмом

Свака инстанца тестирана је по 10 пута како би се добио увид у просечно решење (колона *пр. реш.* у табелама) које постиже хибридни алгоритам као и у одступање просечног решења (у табелама, колона *одст.*) од оптималног решења (горњег ограничења). Оптимално решење, где је познато, се у табелама налази у колони *опт. реш*, а где није, приказано је најбоље решење из литературе (у табелама *најб. реш.*). Одступање просечног решења од оптималног (тј. најбољег, где није познато оптимално) у процентима се рачуна као *одст.* $= 100\frac{(\text{пр.реш.} - \text{опт.реш.})}{\text{опт.реш.}}$, а ако оптимално решење није познато, онда се одступање рачуна по формули *одст.* $= 100\frac{(\text{пр.реш.} - \text{најб.реш.})}{\text{опт.реш.}}$, где *најб. реш.* представља најбоље познато решење. Колона *пр. време* у табелама представља просечно време из 10 извршавања алгоритма на истој инстанци проблема.

У табели 3, приказана су решења предложеног хибридног алгоритма на скупу инстанци из библиотеке $TSPLIB$, одабраном у [2] где су добијена оптимална решења за

све осим две инстанце. Касније је у [52] изложен алгоритам $GILS\text{-}RVND$ који је за те две инстанце поправио горње ограничење а на осталим достигао оптимална па се резултати тестирања упоређују са тим алгоритмом. Треба напоменути да су решења у овој табели Хамилтонови циклуси а не путеви.

Оптимална решења, тј. најбоља где нису позната оптимална, у табелама су болдирана.

Табела 3. Резултати тестирања хибридног алгоритма $GVNS\text{-}SA$ на инстанцама одабранит у [2].

	GILS-RVND		*GVNS-SA*			
инст.	*опт. реш.*	*пр. време (s)*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
dantzig42	**12528**	0,16	**12528**	12528	0,00	0,21
swiss42	**22327**	0,16	**22327**	22327	0,00	0,20
att48	**209320**	0,32	**209320**	209320	0,00	0,29
gr48	**102378**	0,33	**102378**	102378	0,00	0,27
hk48	**247926**	0,30	**247926**	248069	0,06	0,26
eil51	**10178**	0,49	**10178**	10249	0,69	0,29
berlin52	**143721**	0,46	**143721**	143863	0,10	0,33
brazil58	**512361**	0,78	**512361**	512361	0,00	0,46
st70	**20557**	1,65	**20557**	20557	0,00	0,70
eil76	**17976**	2,64	**17976**	18070	0,52	0,87
pr76	**3455242**	2,31	**3455242**	3459378	0,12	0,94
gr96	**2097170**	6,19	**2097170**	2101268	0,20	1,68
rat99	57.986*	11,27	57986	58211	0,39	2,15
kroA100	**983128**	8,59	**983128**	992584	0,96	2,30
kroB100	**986008**	9,21	**986008**	986008	0,00	2,16
kroC100	**961324**	8,17	**961324**	961324	0,00	2,12
kroD100	**976965**	8,46	**976965**	977748	0,08	2,10
kroE100	**971266**	8,31	**971266**	971354	0,01	2,03
rd100	**340047**	8,52	**340047**	340579	0,16	1,95
eil101	27.519*	12,76	27519	27717	0,74	2,10
lin105	**603910**	8,42	**603910**	603910	0,00	2,36
pr107	**2026626**	10,89	**2026626**	2026626	0,00	2,66

** оптималност решења није доказана, вредност представља горње ограничење*

Може се приметити да је хибридни алгоритам изложен у овом раду за све инстанце одабране у [2] достигао оптимална решења или најбоље познато горње ограничење за највише пар секунди уз највеће одступање просечног решења од $0{,}96\%$.

У табели 4 су приказани резултати тестирања предложеног алгоритма на инстанцама димензије 10 конструисаним у [50] на случајан начин. Ове инстанце су решене оптимално са одступањем просечног решења једнаким 0. У табели 5 су приказани резултати тестирања на инстанцама димензије 30, а у табели 6 на инстанцама димензије 50. За обе димензије се добија оптимално решење на свим инстанцама. Одступање просечног решења на инстанцама димензије 30 само у једном

случају има вредност различиту од 0, а код димензије 50 у 8 случајева је веће од 0 са највећом вредношћу $0{,}45\%$.

У табели 7, приказани су резултати тестирања инстанци димензије 100 конструисаних у [50], где је познато само горње ограничење које је касније побољшано у [52]. Стога се резултати упоређују са резултатима из [52]. За 19 инстанци је достигнуто најбоље познато решење а одсупање просечног решења на свим инстанцама не прелази $0{,}75\%$. Просечно време извршавања је највише $2{,}19$ секунди.

Табела 4. Резултати тестирања хибридног алгоритма $GVNS\text{-}SA$ на инстанцама димензије 10 конструисаним у [50].

		GVNS-SA			
инст.	*опт. реш.*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
TRP-S10-R1	**1303**	**1303**	1303	0,00	0,04
TRP-S10-R2	**1517**	**1517**	1517	0,00	0,04
TRP-S10-R3	**1233**	**1233**	1233	0,00	0,04
TRP-S10-R4	**1386**	**1386**	1386	0,00	0,04
TRP-S10-R5	**978**	**978**	978	0,00	0,04
TRP-S10-R6	**1477**	**1477**	1477	0,00	0,04
TRP-S10-R7	**1163**	**1163**	1163	0,00	0,04
TRP-S10-R8	**1234**	**1234**	1234	0,00	0,04
TRP-S10-R9	**1402**	**1402**	1402	0,00	0,04
TRP-S10-R10	**1388**	**1388**	1388	0,00	0,04
TRP-S10-R11	**1405**	**1405**	1405	0,00	0,04
TRP-S10-R12	**1150**	**1150**	1150	0,00	0,04
TRP-S10-R13	**1531**	**1531**	1531	0,00	0,04
TRP-S10-R14	**1219**	**1219**	1219	0,00	0,04
TRP-S10-R15	**1087**	**1087**	1087	0,00	0,04
TRP-S10-R16	**1264**	**1264**	1264	0,00	0,04
TRP-S10-R17	**1058**	**1058**	1058	0,00	0,04
TRP-S10-R18	**1083**	**1083**	1083	0,00	0,04
TRP-S10-R19	**1394**	**1394**	1394	0,00	0,04
TRP-S10-R20	**951**	**951**	951	0,00	0,04

Табела 5. Резултати тестирања хибридног алгоритма $GVNS\text{-}SA$ на инстанцама димензије 30 конструисанит у [50].

		GVNS-SA			
инст.	*опт. реш.*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
TRP-S30-R1	**3175**	**3175**	3175	0,00	0,07
TRP-S30-R2	**3248**	**3248**	3249	0,02	0,05
TRP-S30-R3	**3570**	**3570**	3570	0,00	0,07
TRP-S30-R4	**2983**	**2983**	2983	0,00	0,07
TRP-S30-R5	**3248**	**3248**	3248	0,00	0,06
TRP-S30-R6	**3328**	**3328**	3328	0,00	0,06
TRP-S30-R7	**2809**	**2809**	2809	0,00	0,06
TRP-S30-R8	**3461**	**3461**	3461	0,00	0,06
TRP-S30-R9	**3475**	**3475**	3475	0,00	0,07
TRP-S30-R10	**3359**	**3359**	3359	0,00	0,06
TRP-S30-R11	**2916**	**2916**	2916	0,00	0,06
TRP-S30-R12	**3314**	**3314**	3314	0,00	0,07
TRP-S30-R13	**3412**	**3412**	3412	0,00	0,07
TRP-S30-R14	**3297**	**3297**	3297	0,00	0,06
TRP-S30-R15	**2862**	**2862**	2862	0,00	0,06
TRP-S30-R16	**3433**	**3433**	3433	0,00	0,06
TRP-S30-R17	**2913**	**2913**	2913	0,00	0,06
TRP-S30-R18	**3124**	**3124**	3124	0,00	0,06
TRP-S30-R19	**3299**	**3299**	3299	0,00	0,07
TRP-S30-R20	**2796**	**2796**	2796	0,00	0,07

Табела 6. Резултати тестирања хибридног алгоритма $GVNS\text{-}SA$ на инстанцама димензије 50 конструисанит у [50].

		GVNS-SA			
инст.	*опт. реш.*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
TRP-S50-R1	**12198**	**12198**	12198	0,00	0,30
TRP-S50-R2	**11621**	**11621**	11621	0,00	0,30
TRP-S50-R3	**12139**	**12139**	12139	0,00	0,30
TRP-S50-R4	**13071**	**13071**	13163	0,70	0,30
TRP-S50-R5	**12126**	**12126**	12126	0,00	0,29
TRP-S50-R6	**12684**	**12684**	12685	0,01	0,30
TRP-S50-R7	**11176**	**11176**	11176	0,00	0,31
TRP-S50-R8	**12910**	**12910**	12912	0,01	0,30
TRP-S50-R9	**13149**	**13149**	13149	0,00	0,29
TRP-S50-R10	**12892**	**12892**	12892	0,00	0,32
TRP-S50-R11	**12103**	**12103**	12106	0,03	0,30
TRP-S50-R12	**10633**	**10633**	10633	0,00	0,31
TRP-S50-R13	**12115**	**12115**	12115	0,00	0,30
TRP-S50-R14	**13117**	**13117**	13129	0,09	0,29
TRP-S50-R15	**11986**	**11986**	11986	0,00	0,32
TRP-S50-R16	**12138**	**12138**	12138	0,00	0,31
TRP-S50-R17	**12176**	**12176**	12205	0,24	0,33
TRP-S50-R18	**13357**	**13357**	13420	0,47	0,31
TRP-S50-R19	**11430**	**11430**	11430	0,00	0,33
TRP-S50-R20	**11935**	**11935**	11938	0,02	0,30

Табела 7. Резултати тестирања хибридног алгоритма GVNS-SA на инстанцама димензије 100 конструисанит у [50].

	GILS-RVND		GVNS-SA			
инст.	*најб. реш.*	*пр. време (s)*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
TRP-S100-R1	**32779**	7,05	**32779**	32788	0,03	1,86
TRP-S100-R2	**33435**	7,51	**33435**	33457	0,07	1,96
TRP-S100-R3	**32390**	7,07	**32390**	32446	0,17	1,98
TRP-S100-R4	**34733**	7,27	**34733**	34843	0,32	2,06
TRP-S100-R5	**32598**	8,87	**32598**	32638	0,12	1,99
TRP-S100-R6	**34159**	7,82	**34159**	34336	0,52	1,98
TRP-S100-R7	**33375**	8,74	**33375**	33450	0,23	2,09
TRP-S100-R8	**31780**	7,08	**31780**	31829	0,15	2,00
TRP-S100-R9	**34167**	7,47	**34167**	34258	0,27	1,93
TRP-S100-R10	**31605**	6,78	**31605**	31605	0,00	1,97
TRP-S100-R11	**34188**	7,75	**34188**	34419	0,68	2,02
TRP-S100-R12	**32146**	7,20	**32146**	32260	0,36	2,13
TRP-S100-R13	**32604**	7,78	**32604**	32848	0,75	2,11
TRP-S100-R14	**32433**	6,29	32438	32438	0,02	2,15
TRP-S100-R15	**32574**	7,13	**32574**	32638	0,19	1,94
TRP-S100-R16	**33566**	7,97	**33566**	33568	0,01	2,17
TRP-S100-R17	**34198**	9,23	**34198**	34234	0,11	2,19
TRP-S100-R18	**31929**	7,07	**31929**	32077	0,46	2,07
TRP-S100-R19	**33463**	8,08	**33463**	33529	0,20	1,97
TRP-S100-R20	**33632**	8,73	**33632**	33732	0,30	1,99

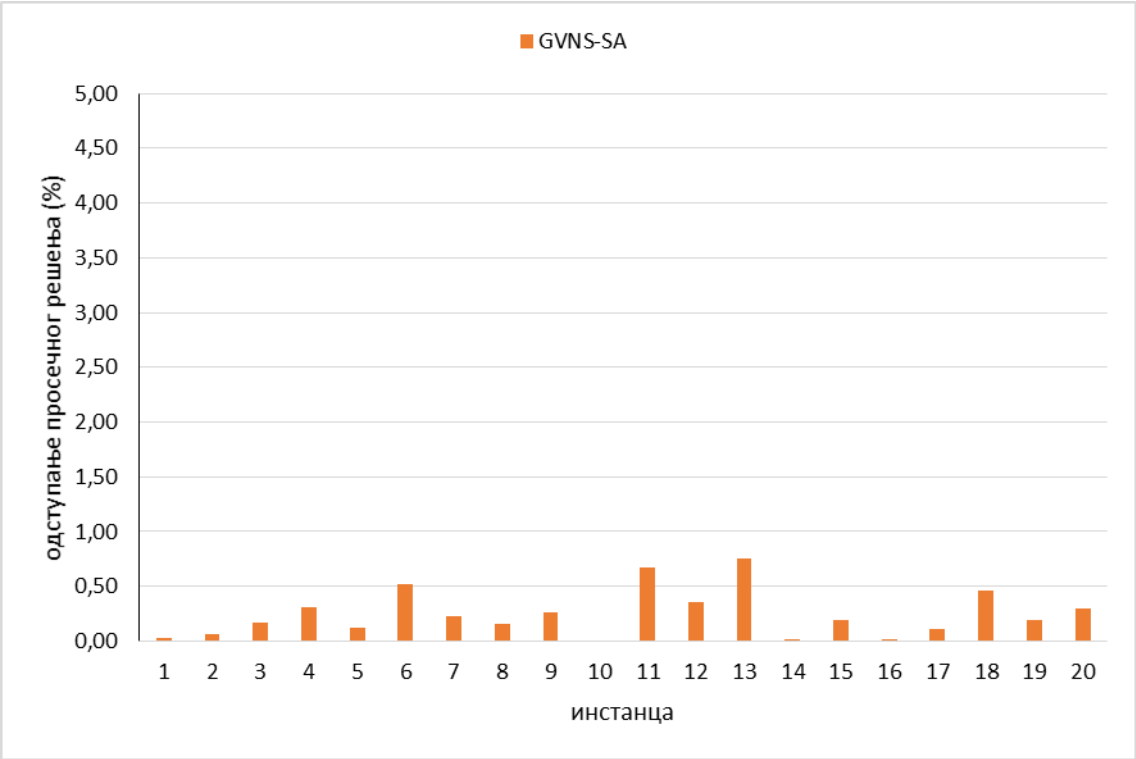


График 1. Одступање просечног решења хибридног алгоритма GVNS-SA на инстанцама димензије 100 конструисанит у [50].

Табела 8. Резултати тестирања хибридног алгоритма GVNS-SA на инстанцама димензије 200 конструисанит у [50].

	GILS-RVND		GVNS-SA			
инст.	*најб. реш.*	*пр. време (s)*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
TRP-S200-R1	**88787**	73,39	88806	89272	0,55	20,20
TRP-S200-R2	**91977**	68,07	92448	93116	1,24	19,06
TRP-S200-R3	**92568**	67,11	**92568**	93204	0,69	20,00
TRP-S200-R4	**93174**	72,17	**93174**	93827	0,70	19,46
TRP-S200-R5	**88737**	70,77	**88737**	89766	1,16	18,06
TRP-S200-R6	**91589**	70,52	**91589**	92814	1,34	19,78
TRP-S200-R7	**92754**	72,80	92756	93846	1,18	21,49
TRP-S200-R8	**89048**	75,15	89118	89786	0,83	19,12
TRP-S200-R9	**86326**	65,45	**86326**	86945	0,72	19,32
TRP-S200-R10	**91552**	74,25	91650	92392	0,92	18,93
TRP-S200-R11	**92655**	73,15	92815	93301	0,70	19,76
TRP-S200-R12	**91457**	76,74	**91457**	92060	0,66	19,12
TRP-S200-R13	**86155**	72,96	**86155**	86909	0,88	18,64
TRP-S200-R14	**91882**	70,94	91989	92607	0,79	20,04
TRP-S200-R15	**88912**	70,41	91589	89737	0,93	19,39
TRP-S200-R16	**89311**	77,89	**89311**	90866	1,74	20,72
TRP-S200-R17	**89089**	71,17	**89089**	89777	0,77	19,79
TRP-S200-R18	**93619**	77,03	93972	94697	1,15	19,53
TRP-S200-R19	**93369**	71,08	93676	94574	1,29	18,51
TRP-S200-R20	**86292**	70,61	86331	87347	1,22	20,14

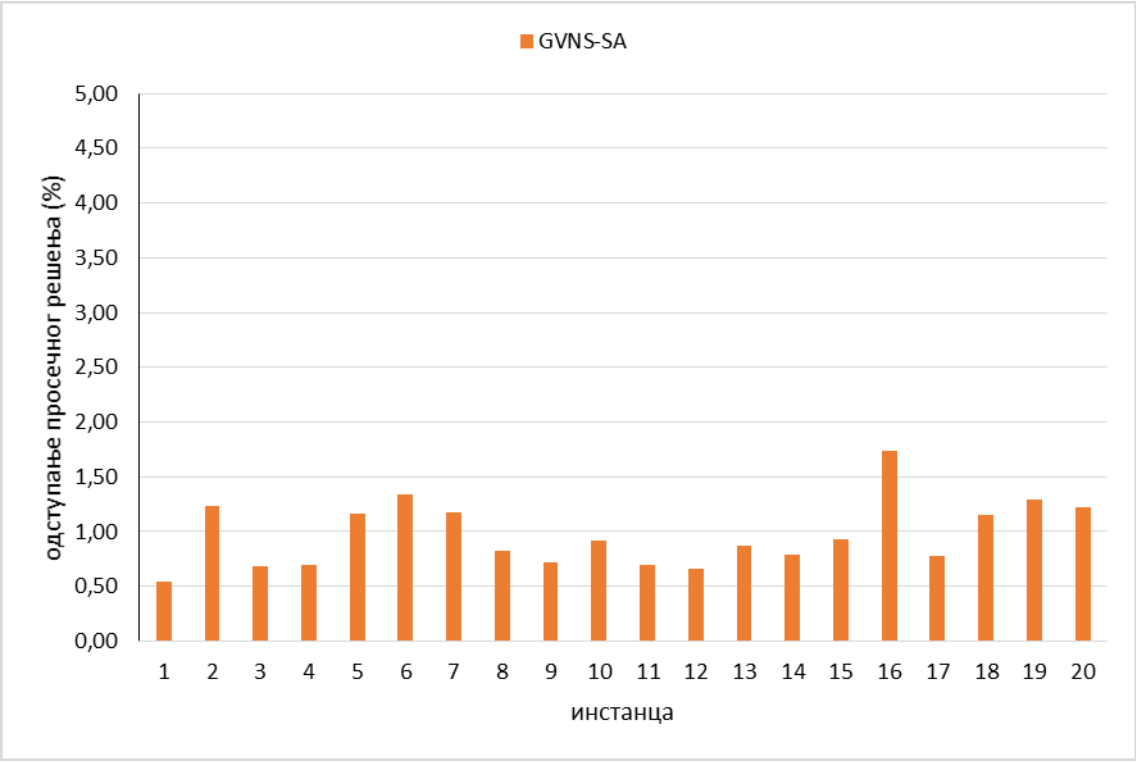


График 2. Одступање просечног решења хибридног алгоритма GVNS-SA на инстанцама димензије 200 конструисанит у [50].

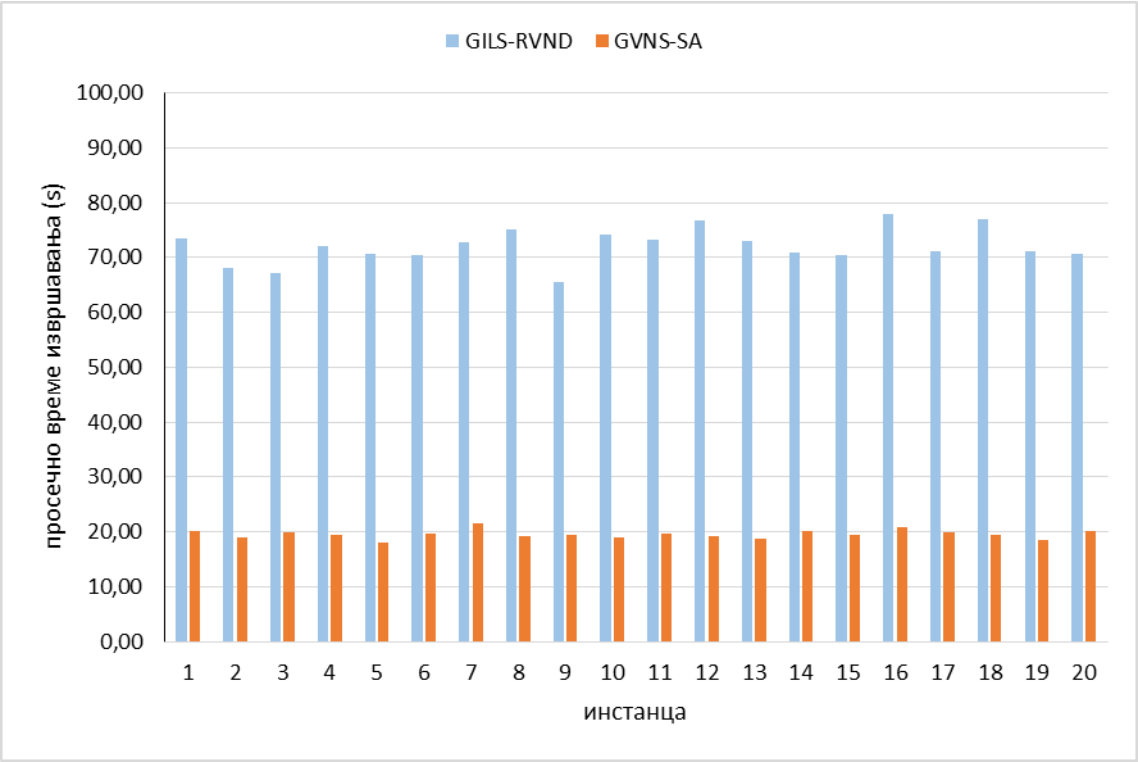


График 3. Просечно време извршавања хибридног алгоритма $GVNS\text{-}SA$ на инстанцама димензије 200 конструисанит у [50].

Табела 9. Резултати тестирања хибридног алгоритма $GVNS\text{-}SA$ на инстанцама димензије 500 конструисанит у [50].

	GILS-RVND		*GVNS-SA*			
инст.	*најб. реш.*	*пр. време (s)*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
TRP-S500-R1	**1841386**	1738,48	1871090	1889008	2,59	483,38
TRP-S500-R2	**1816568**	1476,13	1835543	1847732	1,72	525,37
TRP-S500-R3	**1833044**	1557,48	1845870	1866666	1,83	513,50
TRP-S500-R4	**1809266**	1597,06	1821669	1846250	2,04	489,95
TRP-S500-R5	**1823975**	1530,94	1842106	1864735	2,23	463,15
TRP-S500-R6	**1786620**	1576,91	1812489	1826810	2,25	510,18
TRP-S500-R7	**1847999**	1584,67	1875923	1901250	2,88	458,71
TRP-S500-R8	**1820846**	1565,01	1844134	1864047	2,37	477,74
TRP-S500-R9	**1733819**	1409,23	1751823	1764179	1,75	478,03
TRP-S500-R10	**1762741**	1621,85	1797515	1804639	2,38	455,73
TRP-S500-R11	**1797881**	1530,98	1810796	1830966	1,84	471,39
TRP-S500-R12	**1774452**	1554,75	1795137	1820156	2,58	489,39
TRP-S500-R13	**1873699**	1598,46	1900017	1909469	1,91	457,19
TRP-S500-R14	**1799171**	1701,90	1822190	1829656	1,69	460,45
TRP-S500-R15	**1791145**	1623,79	1801765	1823544	1,81	476,89
TRP-S500-R16	**1810188**	1583,70	1834573	1840784	1,69	451,12
TRP-S500-R17	**1825748**	1549,80	1857894	1871635	2,51	488,32
TRP-S500-R18	**1826263**	1620,02	1864287	1876765	2,77	497,81
TRP-S500-R19	**1779248**	1602,87	1825407	1828812	2,79	501,49
TRP-S500-R20	**1820813**	1507,96	1847916	1849547	1,58	455,57

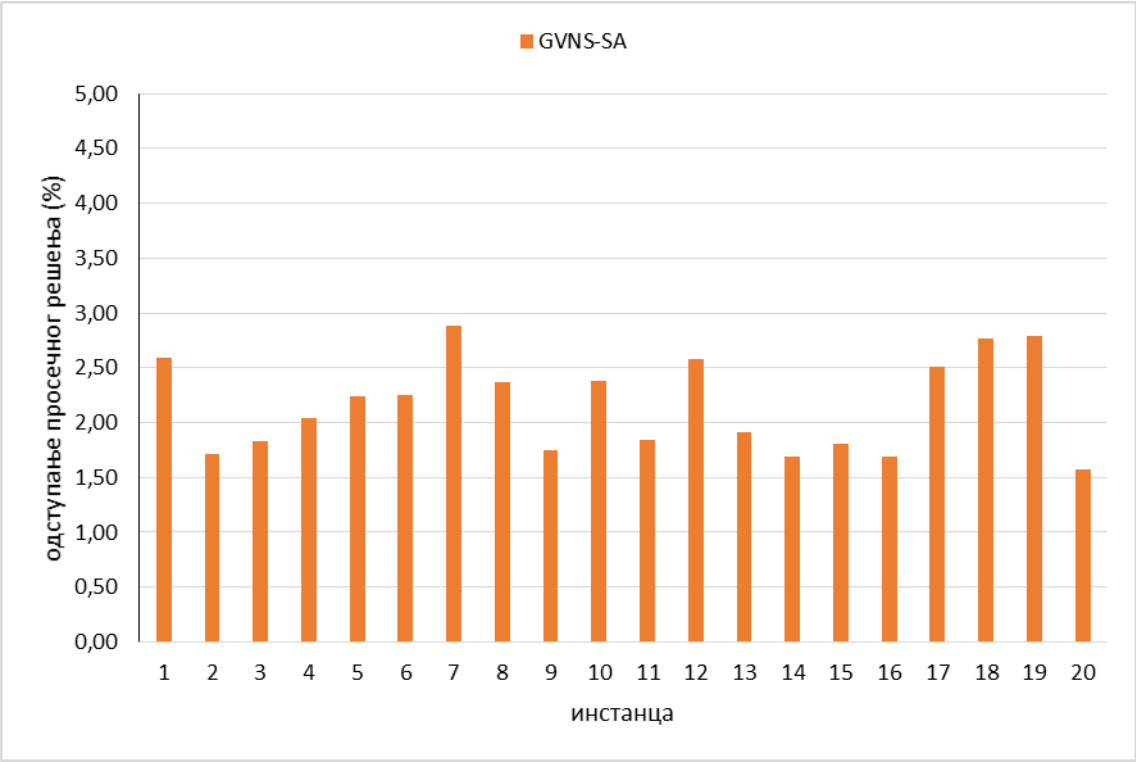


График 4. Одступање просечног решења хибридног алгоритма GVNS-SA на инстанцама димензије 500 конструисанит у [50].

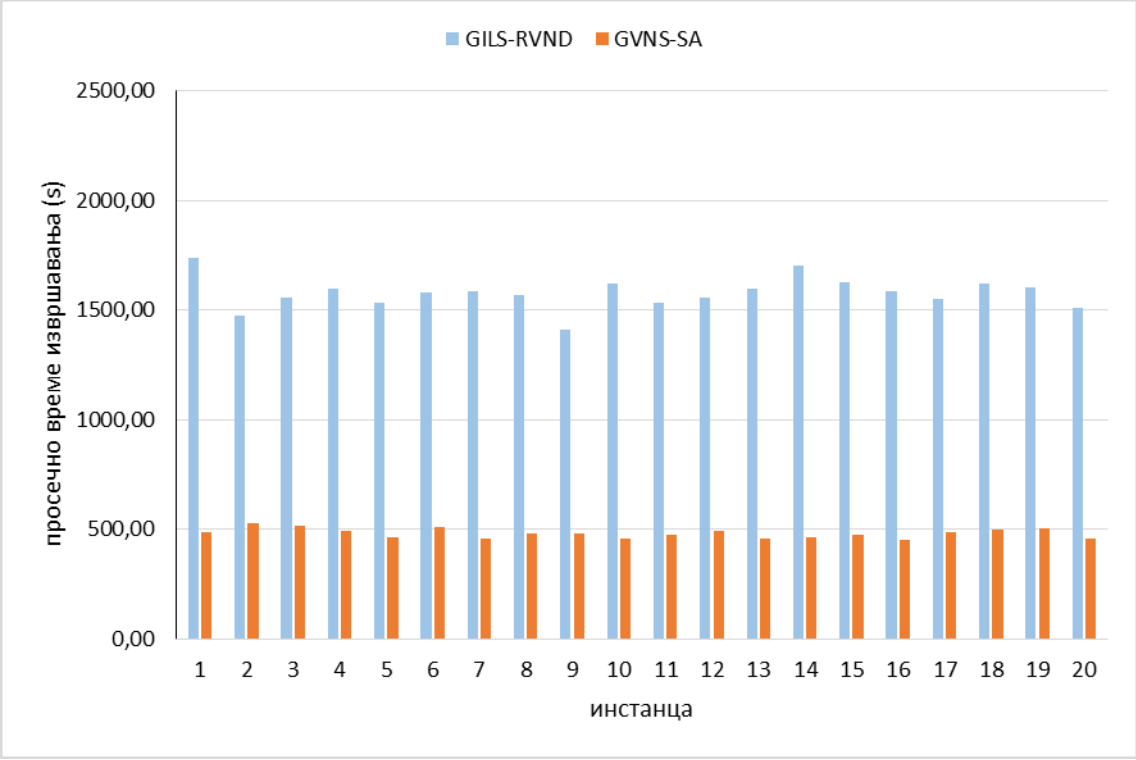


График 5. Просечно време извршавања хибридног алгоритма GVNS-SA на инстанцама димензије 500 конструисанит у [50].

У табели 9, упоређују се резултати на инстанцама димензије 500 генерисаних у [50] где су најбоља горња ограничења добијена у [52]. Оступање просечног решења је највише $2{,}79\%$ а просечно време извршавања око 500 секунди.

У табели 10 су приказани резултати тестирања на скупу инстанци из библиотеке $TSPLIB$ које су одабране у [50]. Најбоља горња ограничења су такође добијена у [52] па се резултати упоређују са тим ограничењима.

Табела 10. Резултати тестирања хибридног алгоритма $GVNS$-SA на скупу инстанци из библиотеке $TSPLIB$ одабраних у [50].

	GILS-RVND		*GVNS-SA*			
инст.	*најб. реш.*	*пр. време (s)*	*најб. реш.*	*пр. реш.*	*одст. (%)*	*пр. време (s)*
st70	**19215**	1,51	19710	19751	2,79	0,59
rat99	**54984**	9,47	56573	56862	3,42	1,77
kroD100	**949594**	6,90	951609	951944	0,25	1,70
lin105	**585823**	6,19	586751	586751	0,16	1,75
pr107	**1980767**	8,13	1981991	1981995	0,06	2,13
rat195	**210191**	75,56	217104	218160	3,79	13,54
pr226	**7100308**	59,05	7118051	7118796	0,26	25,77
lin318	**5560679**	220,59	5569520	5613412	0,95	79,35
pr439	**17688561**	553,74	17800061	17855062	0,94	246,79
att532	**5581240**	1792,61	5652178	5691485	1,98	601,87

7 Закључак

У овом раду описане су две метахеуристике које имају широку примену у приближном решавању разних оптимизационих проблема, који се за велике димензије не могу решити практично егзактним решавачем. Описане метахеуристике су метода променљивих околина и метода симулираног каљења, за које је изложен псеудокод и значајније особине алгоритама.

Изложен је проблем минималног кашњења, описан његов значај и дата математичка формулација проблема. Развијена је хибридна метахеуристика која спаја варијацију методе промељивих околина и симулирано каљење. У фази локалне претраге овог хибридног алгоритма се на почетку користи метода VND како би се добило добро међурешење које се у другом делу алгоритма унапређује методом симулираног каљења. Дат је детаљан опис имплементације, објашњене су структуре коришћених околина. Описана је динамичка провера квалитета решења која се користи приликом претраге околине. Ова динамичка провера значајно смањује сложеност алгоритма а тиме и време извршавања.

Хибридни алгоритам методе променљивих околина и методе симулираног каљења за проблем минималног кашњења је први пут описан у овом раду. Имплементација овог алгоритма је тестирана на јавно доступним инстанцама библиотеке $TSPLIB$ као и на скупу инстанци конструисаних на случајан начин у [50]. Постигнута су тренутно најбоља решења из [52] за $TSPLIB$ инстанце димензија 42 до 107 одабране у [2], уз одступање просечног решења од $0{,}96\%$ и просечна времена од $0{,}21$ до $2{,}66$ секунди. За инстанце генерисане у [50] постигута су оптимална решења за све инстанце димензија $10, 30$ и 50, а за инстанце димензије 100, за 19 од 20 инстанци је достигнуто горње ограничење из [52] где су одступања просечног решења за све инстанце мање од $0{,}75\%$. За инстанце из [50] димензија 200 и 500 постигнута су решења близу најбољим из [52] са одступањем просечног решења мањим од 2.88% уз знатно краће просечено време извршавања. На последњем скупу инстанци које припадају библиотеци $TSPLIB$, одабране у [50] а чија су најбоља решења постигнута у [52], хибридни алгоритам описан у овом раду достиже решења близу тренутно најбољим са одступањем просечног решења највише $3{,}79\%$ и битно краћим просечним временом извршавања. За 102 од 109 инстанци димензија 42 до 107 постигнута су оптимална решења или најбоља горња ограничења која су тренутно позната. На инстанцама димензија 195 до 532 постигута су решења близу најбољих горњих ограничења.

Како нису достигнута најбоља позната решења за инстанце димензија већих од 195, постоји простор за даље усавршавање овог хибридног алгоритма. Даљи рад на алгоритму обухвата развијање нових структура околина како би се утврдила најбоља комбинација постојећих и нових околина, као и разматрање других хеуристичких метода које би се примењивале у фази локалне претраге и њихова најбоља комбинација.

Литература

[1] Aarts, E. H. L., Korst, J. H. M., and van Laarhoven, P. J. M., "A quantitative analysis of the simulated annealing algorithm: A case study for the traveling salesman problem", *Journal of Statistical Physics*, 50, 1-2 (1988), 187-206.

[2] Abelado, H., Fukasawa, R., Pessoa, A., and Uchoa, E., "The time dependent traveling salesman problem: Polyhedra and algorithm", *Tech. Rep. RPEP*, 15 (2010).

[3] Abeledo, H., Fukasawa, R., Pessoa, A., and Uchoa, E., "The time dependent traveling salesman problem: Polyhedra and branch-cut-and-price algorithm", *Proceedings of the 9th International Symposium on Experimental Algorithms* (2010), 202-2013.

[4] Angel-Bello, F., Alvarez, A., and Garcia, I., *"Formulation for the minimum latency problem: an experimental evaluation"*, 2011.

[5] Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J., *"The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[6] Applegate, D. L., Bixby, R. E., Chvatal, V., Cook, W. J., Espinoza, D.G., Goycoolea, M., and Helsgaun, K., "Certification of an optimal TSP tour through 85900 cities", *Operations Reearch Letters*, 37, 1 (2009), 11–15.

[7] Arora, S., "Polynomial Time Approximation Schemes for Euclidian Traveling Salesman and Other Geometric Problems", *Journal of the ACM*, 45, 5 (1998), 753-782.

[8] B. Shirrish, J. Nigel, M.R. Kabuka, "A Boolean neural network approach for the traveling salesman problem", *IEEE Transactions on Computers*, 42, 10 (1993), 1271–1278.

[9] Bektas, T., "The multiple traveling salesman problem: an overview of formulations and solution procedures", *Omega*, 34 (2006), 209–219.

[10] Blum, A., Chalasanit, P., Pulleyblankt, B., Raghavan, P., and Sudan, M., "The minimum latency problem", *Proceedings of the 26th Annual ACM Symposium on Theory of Computing* (1994), 163–171.

[11] Blum, C., Roli, A., and Sampels, M., *"Hybrid Metaheuristics: An Emerging Approach to Optimization*. Springer, 2008.

[12] Carrabs, F., Cordeau, J.-F., and Laporte, G., "TSP pickup and delivery with fifo loading", *INFORMS Journal on Computing*, 19, 4 (2007), 618 - 632.

[13] Carter, A.E. and Ragsdale, C.T., "A new approach to solving the multiple traveling salesperson problem using genetic algorithms", *European Journal of Operational Research*, 175 (2006), 246–257.

[14] Chaudhuri, K., Godfrey, B., Rao, S., and Talwar, K., "Paths, trees, and minimum latency tours", *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science* (2003), 36-45.

[15] Dantzig, G. B., Fulkerson, D. R., and Johnson, S. M., "Solution of a large-scale traveling salesman problem", *Operations Research*, 2 (1954), 393–410.

[16] Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F.C.R., and Vansteenwegen, P., "Heuristics for the traveling repairman problem with profits", *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2010* (2010), 34-44.

[17] Dorigo, M. and Gambardella, L.M. , "Ant colony system: a cooperative learning approach to the traveling salesman problem", *IEEE Transactions on Evolutionary Computation* (1997), 53 - 66.

[18] Fischetti, M., Gonzales, J. J. S., and Toth, P., "A Branch-and-Cut algorithm for the symetric generalized traveling salesman problem", *Oper. Res.*, 45, 3 (1997), 378-394.

[19] Fischetti, M., Gonzales, J. J. S., and Toth, P., "The Generalized Traveling Salesman and Orienteering Problem", *Kluwer* (2002).

[20] Fischetti, M., Laporte, G., and Martello, S., "The delivery man problem and cumulative matroids", *Operations Research*, 6, 1055-1064 (1993).

[21] Glover, F., "Artificial intelligence, heuristic frameworks and tabu search", *Managerial and Decision Economics*, 11, 5 (1990), 365–378.

[22] Glover, F. and Kochenberger, G. A., *"Handbook of Metaheuristics*. Springer, 2010.

[23] Gomory, R. E., "An algorithm for integer solutions to linear programs", *In: Graves RL & Wolfe P (eds). Recent Advances in Mathematical Programming* (1963), 269–302.

[24] Gorenstein, S., "Printing press scheduling for multi-edition periodicals", *Management Science*, 16 (1970), 373–383.

[25] Grotschel, M. and Holland, O., "Solution of Large-scale Symmetric Traveling Salesman Problems", *Mathematical Programming*, 51 (1991), 141-202.

[26] Hansen , P., Mladenovic, N., and Perez, J. A. M., "Variable neighborhood search: methods and applications", *4OR-Q J Oper. Res.*, 6 (2008), 319-360.

[27] Hu, B. and Raidl, G. R., "Effective neighborhood structures for the generalized traveling salesman problem", *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2008*, 4972 (2008).

[28] J.L. Ryan, T.G. Bailey, J.T. Moore, W.B. Carlton, "Reactive tabu search in unmanned aerial reconnaissance simulations", *The 30th Conference on Winter Simulation, IEEE Computer Society Press* (1998), 873–879.

[29] Johnson, D. S. and McGeoch, L. A., "The traveling salesman problem: A case study in local optimization", *Local search in combinatorial optimization* (1997), 215–310.

[30] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing", *Science, New Series*, 220, 4598 (1983), 671-680.

[31] Land, A. H., "The solution of some 100-city traveling salesman problems", *Technical report* (1979).

[32] Laporte, G., Asef-Vaziri, A., and Sriskandarajah, C., "Some Applications of the Generalized Traveling Salesman Problem", *Journal of the Operational* , 47 (1996), 1461-1467.

[33] Laporte, G. and Nobert, Y., "Generalized traveling salesman problem through n sets of nodes: An integer programming approach", *INFOR*, 21, 1 (1983), 61-75.

[34] Lin, J. S., "Christofides' heuristic", *Course Notes for* (2005).

[35] Lin, S. and Kernighan, B. W., "An Effective Heuristic Algorithm for the Traveling-Salesman Problem", *Operations Research*, 21, 2 (1973), 498–516.

[36] Lucena, A., "Time-dependent traveling salesman problem - the deliveryman case", *Networks* (1990), 753-763.

[37] Malek, M., Guruswamy, M., Pandya, M., and Owens, H., "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem", *Annals of Operations Research*, 21, 1 (1989), 59-84.

[38] Martin, G. T., "Solving the traveling salesman problem by integer programming", *CEIR* (1966).

[39] Mendez-Diaz, I., Zabala, P., and Lucena , A., "A new formulation for the traveling deliveryman problem", *Discrete Appled Math* (2008), 3223-3237.

[40] Miliotis, P., "Integer programmi.ng approaches to the travelling salesman problem", *Mathematical Programming*, 10 (1976), 376–378.

[41] Miliotis, P., "Using cutting planes to solve the symmetric travelling salesman problem", *Mathematical Programming*, 15 (1978), 177–188.

[42] Mladenovic, N. and Hansen, E., "Variable neighborhood search", *Computers & Operations Research*, 24, 11 (1997), 1097-1100.

[43] Mladenovic, N., Todosijevic, R., and Urosevic, D., "An efficient general variable neighborhood search for large travelling salesman problem with time windows", *Yugoslav Journal of Operations Research*, 23, 1 (2013), 19-30.

[44] Ngueveu, S. U., Prins, C., and Wolfler Calvo, R., "An effective memetic algorithm for the cumulative capacitated vehicle routing problem", *Computers & Operations Research*, 37, 11 (2010), 1877–1885.

[45] Noon, C. E., "The generalized traveling salesman problem", *Ph. D. Dissertation* (1988).

[46] Park, Y.B., "A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines", *International Journal of Productions Economics*, 73 (2001), 175–188.

[47] Pintea, C. M., Pop, P. C., and Chira, C., "The Generalized Traveling Salesman Problem solved with Ant Algorithms", *CoRR* (2013).

[48] Renaud, J. and Boctor, F. F., "An efficient composite heuristic for the symmetric generalized traveling salesman problem", *European Journal of Operational Research*, 108, 3 (1998), 571–584.

[49] Sahni, S. and Gonzalez, T., "P-complete approximation problems", *Journal of the AXM*, 3 (1976), 555-565.

[50] Salehipour, A., Sorensen, K., Goos, P., and Braysy, O., "Efficient GRASP + VND and GRASP + VNS metaheuristics for the traveling repairman problem", *A Quarterly Journal of Operations Research*, 2 (2011), 189-209.

[51] Silberholz, J. and Golden, B., "The Generalized Traveling Salesman Problem: A New Genetic Algorithm Approach", In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*. 2007.

[52] Silva, M. M., Subramanian, A., Vidal, T., and Ochi, L. S., "A simple and effective metaheuristic for the Minimum Latency Problem", *European Journal of Operational Research*, 221 (2012), 513-520.

[53] Snyder, L. and Daskin, M., "A random-key genetic algorithm for the generalized traveling salesman problem", *European Journal of Operational*, 17, 1 (2006), 38-53.

[54] Svestka, J. A. and Huckfeldt, V. E., "Computational experience with an m-salesman traveling salesman algorithm", *Management Science*, 17 (1973), 790–799.