THE COMPARISON OF GENETIC ALGORITHMS WHICH SOLVE ORIENTEERING PROBLEM USING COMPLETE AND INCOMPLETE GRAPH

Krzysztof Ostrowski, Jolanta Koszelew

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: The purpose of this work was to compare two forms of genetic algorithm (complete and incomplete graph version) which solves Orienteering Problem (OP). While in most papers concerning OP graph is complete and satisfies triangle inequality, in our versions such assumptions may not be satisfied. It could be more practical as transport networks are graphs which do not have to satisfy those conditions. In such cases, graphs are usually complemented with fictional edges before they can be used by classic OP solving algorithms which operate on complete graphs. This paper answers the question: Is it better (in terms of results quality and time consumption) to transform graphs to classic OP form before running algorithm (complete graph version) or to solve OP on graphs without any assumptions and changes (incomplete graph version)? The computer experiment was conducted on the real transport network in Poland and its results suggest that it is worth checking both versions of the algorithm on concrete networks.

Keywords: orienteering problem, OP, transport network, genetic algorithm, GA, incomplete graph, complete graph

1. Introduction

The orienteering problem (OP) is still one of the most challenging optimization problems. It is related to the travelling salesman problem (TSP). The main difference is that not all cities have to be visited and each of them has some profit. The goal is to maximize total profit within a given time frame. The OP has a lot of practical applications (i.e. logistics, planning and tourism [8] [12]). For example, it could be very helpful in trip planning and such systems for tourists are developed [18][2]. Several exact solutions of the OP were proposed, including linear and dynamic programming [7][6]. However, the OP (like TSP) is an NP-hard problem [4][3]

Zeszyty Naukowe Politechniki Bialostockiej. Informatyka, vol. 8, pp. 61-77, 2011.

and exact solutions are impractical in terms of time consumption for bigger sized problems. Thus, algorithms with various heuristic strategies are implemented to solve the problem more efficiently [17]. One of the first heuristics (including Monte Carlo method) were applied by Tsiligirides [15]. Others proposed methods include i.e. 2-opt and 3-opt procedures and centre of gravity usage [10][4][5]. One of the most effective heuristic solving the OP is a guided local search heuristic [16][11]. Artificial neural networks and genetic algorithms were also used to solve the OP [19][13]. The OP itself has also several extensions and variants i.e. team orienteering problem (TOP) and orienteering problem with time windows (OPTW) [1][14].

In the paper two versions of genetic algorithm (GA) with mutation are presented to solve the OP. One of them (IG) operates on an incomplete graph and the other (CG) performs edge completion before running the main genetic algorithm on a complete graph. The paper is organised as follows. Section 2 presents the definition of the OP with a network example. Section 3 gives a detailed specification of both GA versions with several examples. In section 4 the experimental results of these two algorithms (with a couple of different heuristics) were compared on a real transport network. Conclusions are presented in section 5.

2. Problem definition

Given a set of n vertices (each vertex has some nonnegative profit), travel time between every pair of vertices and the starting point (s) and the end point (e), the purpose of the OP is to find the path (limited by travel time t_{max}) between vertices s and e that maximizes the total profit (computed as the sum of profits of visited vertices). Each point can be visited at most once.

The OP can be also defined using an undirected, complete graph $G = (V, E)$, when V ($|V| = n$) is the vertex set and E is the edge set. Each vertex i is associated with some nonnegative profit p_i and each edge connecting vertices i and j is associated with some travel time t_{ij}. The goal is to find a Hamiltonian path of a subgraph of G (between start (s) and end (e) vertices) which maximizes the total collected profit and is limited by constraint t_{max}.

The problem introduced in this article is OP with one substantial modification: each vertex can be visited more than once during the travel, but the total profit is increased only when a given vertex is visited for the first time. In addition, graph edges do not have to satisfy triangle inequality. The paper presents two versions of algorithms solving OP. Both of them are genetic algorithms (GA) with local search procedure in the form of mutation. In the first algorithm version presented in the article there are direct connections (travel times) only between some pairs of vertices (incomplete and undirected graph) and no edge completion is performed before running the OP solving algorithm. In the second version it is assumed that there is a direct connection (travel time) between every pair of points (complete and undirected graph). If this assumption is not satisfied, a graph is complemented with virtual edges using Dijkstra algorithm. There is an important note about the score calculation: if a given path includes a virtual edge between vertices i and j then all vertices on the shortest path from i to j are considered when computing the total profit.

At the problem input there are graph G (with matrix of travel times t and vector of profits p) and maximum travel time t_{max}. At the problem output route r in graph G is obtained. It starts and ends in the vertex number 1, its travel time is not greater than t_{max} and its total profit is maximized. In our experiment graph G is a real transport network with cities (and profits) and connections between them.

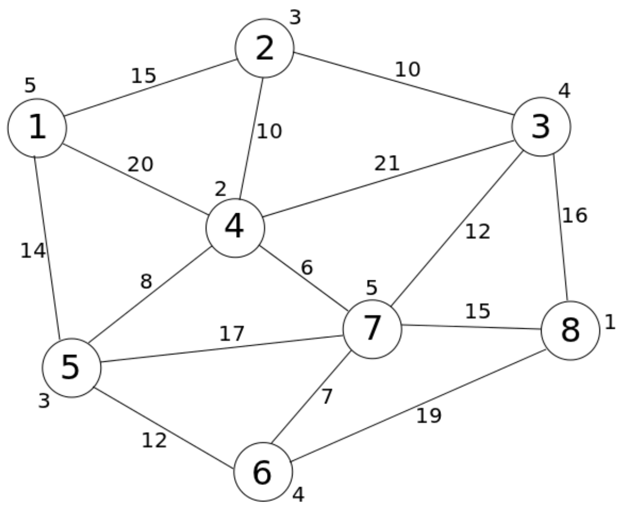

Fig. 1. A graph representing an exemplary transport network

In the picture (fig. 1) there is an exemplary network of 8 cities. Travel time (t_{ij}) values are marked on edges and profit (p_i) values are marked near vertices (cities). This network is used in all examples from the paper. Let $t_{max} = 80$. A cycle r=1, 5, 4, 7, 6, 7, 3, 2, 1 can be a solution. Its travel time is equal to 79 (14+8+6+7+7+12+12+15) and its total profit is 26 (5+3+2+5+4+4+3 - only first visits to cities 1 and 6 are counted).

3. Algorithm specification

Both presented versions are genetic algorithms with mutation. Main steps in both versions are the same but genetic operations are different in some important details. Tours are encoded into a chromosome as a sequence of vertices (cities). It is the most natural way of adopting GA for OP. For example, the cycle from previous section can be represented as an individual (1, 5, 4, 7, 6, 7, 3, 2, 1) and its fitness is 26 (equal to the total profit).

3.1 Incomplete graph version

This algorithm is an improved (in terms of time complexity) version of [9]. First, an initial population of P_{size} solutions is generated. At the beginning a random vertex v adjacent to vertex 1 (the start point) is chosen and t_{1v} is added to the current travel time. If the current travel time does not exceed $0.5 \cdot t_{max}$, the tour generation is continued. Now we start at the vertex v and choose random vertex u adjacent to v. At every step we exclude the visited vertex from the set of possibilities - it prevents from visiting a given vertex repeatedly. If the current tour length is greater than $0.5 \cdot t_{max}$, the last vertex is rejected and we return to vertex 1 the same way in reverse order. This way of generating the individuals of the initial population means that they are symmetrical in respect of the middle vertex in the tour. However, these symmetries are removed by the algorithm. An example of an initial population is shown in table 1.

After generating initial population, the GA starts to improve the current population through repetitive application of selection, crossover and mutation. The algorithm stops after N_g generations and the resulting tour is the best individual from the final generation. First, tournament selection is applied - we select t_{size} random, different individuals from the current population and the best one from the group is copied to the next population. The whole tournament group is returned to the old population. After P_{size} repetitions of this step a new population is created (a selection example in table 2).

The example presented in tab. 2 shows how selection improves average population fitness. However, if t_{size} is too high (relatively to P_{size}), the population converges very fast.

The crossover is performed as follows: first, two random parental individuals are selected. Afterwards we randomly choose a common gene (crossing point) in both parents (first and last genes are not considered). If there are no common genes, crossover cannot be done and no changes in chosen chromosomes are applied.

Table 1. An initial population example ($P_{size} = 5$, $t_{max} = 80$)

No	Individual	Fitness	Travel time
1	(1, 2, 3, 7, 3, 2, 1)	17	74
2	(1, 5, 6, 7, 6, 5, 1)	17	66
3	(1, 4, 7, 6, 7, 4, 1)	16	66
4	(1, 5, 7, 5, 1)	13	62
5	(1, 5, 4, 7, 6, 7, 4, 5, 1)	19	77

Table 2. An example of a tournament selection performed on the population from table 1

No	Numbers of individuals selected into the tournament group	Number of the best individual	The best individual (next population member)	Fitness of the best individual
1	1, 3, 4	1	(1, 2, 3, 7, 3, 2, 1)	17
2	2, 4, 5	5	(1, 5, 4, 7, 6, 7, 4, 5, 1)	19
3	2, 3, 4	2	(1, 5, 6, 7, 6, 5, 1)	17
4	1, 2, 5	5	(1, 5, 4, 7, 6, 7, 4, 5, 1)	19
5	1, 2, 3	1	(1, 2, 3, 7, 3, 2, 1)	17

After that, two new individuals are created as a result of exchanging chromosome fragments (from the crossing point to the end of the chromosome) in both parents. If one of the children does not preserve t_{max} constraint, it is replaced by the fitter parent in the new population. If both children do not preserve this constraint, the parents replace them in the new population (no changes applied).

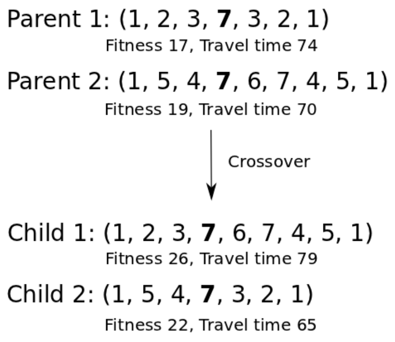
Parent 1: (1, 2, 3, **7**, 3, 2, 1)
Fitness 17, Travel time 74
Parent 2: (1, 5, 4, **7**, 6, 7, 4, 5, 1)
Fitness 19, Travel time 70
Crossover
Child 1: (1, 2, 3, **7**, 6, 7, 4, 5, 1)
Fitness 26, Travel time 79
Child 2: (1, 5, 4, **7**, 3, 2, 1)
Fitness 22, Travel time 65

Fig. 2. An example of crossover ($t_{max} = 80$), crossing point in bold

In the example presented in fig. 2 two children with improved fitness and low enough travel time are created. Both parents are symmetrical but the symmetry in the offspring individuals is removed by crossover. After selection and crossover the population undergo mutation. First we select a random individual to be mutated. There are two possible kinds of mutation: inserting a new gene and removing an existing gene. Several mutation versions (with different heuristics) have been implemented in the presented algorithm. In some of them only inserting mutation can be performed, but in others both kinds of mutation are possible with the probability of 0.5. During the inserting mutation all the possibilities of inserting a new gene that is not present in the chromosome are considered (without exceeding t_{max}) and the best is chosen. Depending on the heuristic used it can be the one with lowest travel time increase, highest fitness gain or best $fitness/travelTime$ ratio.

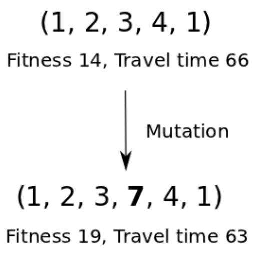

Fig. 3. An example of mutation ($t_{max} = 80$, highest fitness gain heuristic), inserted gene in bold

In the example presented in fig. 3 there are two possibilities of inserting a new gene into chromosome (1, 2, 3, 4, 1) without exceeding t_{max}. One of them is inserting gene 7 between genes 3 and 4 (fitness gain is 5) and the other is inserting gene 5 between genes 4 and 1 (fitness gain is equal to 3). The first of them (with higher fitness gain) is chosen. There are also several variants of removing mutation. In all of them we consider only genes which can be removed without perturbing path continuity - between their neighbouring genes in the chromosome there should be an edge in the graph. After obtaining the set of possibilities, an appropriate heuristic is performed. In the first heuristic only genes that appear in the chromosome more than once are considered (except first and last genes). If there are no such candidates, the removing mutation is not performed. Otherwise we choose the gene in order to shorten the travel time as much as possible. In the second heuristic we concentrate on the lowest fitness loss. In the best-case scenario the fitness loss is zero (we remove a gene that has duplicates in the chromosome), but it is possible that fitness loss is greater than zero (no duplicates in the chromosome).

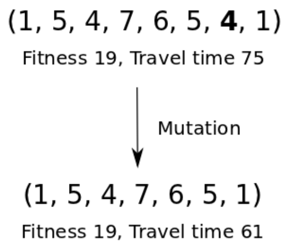

Fig. 4. An example of mutation ($t_{max} = 80$, highest fitness gain heuristic), removed gene in bold

In the example presented in fig. 4 we have three candidates to remove: gene 5 (between genes 1 and 4), gene 4 (between genes 5 and 7) and gene 4 (between genes 5 and 1). These genes meet two conditions: they have duplicates and their chromosome neighbours are connected in the graph. If we remove gene 5, the path is shorten by 2 (14+8-20), but if we remove gene 4 (between genes 5 and 7) travel time is even greater. Thus, the best candidate is gene 4 (between genes 5 and 1) - we shorten travel time by 14 (20+8-14).

3.2 Complete graph version

In the second algorithm version there are some necessary steps to do before performing the GA. The graph is complemented with virtual edges. First, Dijkstra algorithm is run from every vertex (altogether n times). If there is no edge between vertices i and j we add virtual edge (i, j) computed during the Dijkstra algorithm. It contains more information than a real edge - besides the travel time, vertices on the shortest path between i and j are also remembered. Virtual edge (j,i) is constructed independently during a different run of Dijkstra algorithm. Its travel time in undirected graphs is the same as in edge (i, j), but the path itself could be totally different (a lot of different shortest paths possible). This partially reduces the chromosome symmetry obtained during generating the initial population - although the chromosome is still symmetrical, its inner paths (virtual edges) could differ significantly. In our exemplary graph (presented in fig. 1) both virtual edges between vertices 1 and 8 have travel time of 41, but the associated shortest paths are different (1-4-7-8 and 8-3-2-1).

As a result of the edge completion described above, calculating the individual fitness is different from the one applied the first algorithm version. While encoding tour vertices into a chromosome is the same, vertices "hidden" in virtual edges (shortest paths) are also considered when calculating fitness. It means that profits of all vertices (hidden or not) are summed up only during the first visit of a given vertex. In this section term ’main path’ means the sequence of chromosome genes (vertices) whereas term ’full path’ refers to the sequence of all vertices in the route (including those hidden in virtual edges).

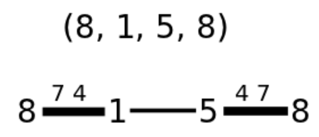

Fig. 5. An example of a chromosome. Virtual edges (shortest paths) in bold and hidden vertices above

In the example presented in fig. 5 there is a chromosome representing tour 8-1-5-8 (main path) which actually is 8-(7-4)-1-5-(4-7)-8 (full path). The shortest path from vertex 8 to vertex 1 is 8-7-4-1 and the shortest path from vertex 5 to vertex 8 is 5-4-7-8 - these are virtual edges. The edge from vertex 1 to vertex 5 is real. To calculate overall profit (fitness) we take into account vertices 8, 7, 4, 1, and 5 - the fitness is 16. The total travel time is 84 - the sum of 41 (shortest path from vertex 8 to vertex 1), 14 and 29 (shortest path from vertex 5 to vertex 8).

The first step of the GA is to generate an initial population. It is similar to the procedure performed in the incomplete graph version. The main difference is that the graph is full and from a given vertex we can construct the path to any other (except those already included in the main path). In the process of creating individuals both real and virtual paths could be added.

After generating the initial population, the GA starts its standard procedure of repeated selection, crossover and mutation. The tournament selection and crossover are performed in the same way as in the incomplete graph version of the algorithm. During the crossover only vertices from the main path are considered when selecting crossing point (virtual and real edges treated in the same way).

Mutation is the only step which is strongly different in both versions of the GA. We select a random individual and then inserting or removing mutation is performed, both with the probability of 0.5 (in some variants inserting mutation is the only option). During inserting mutation all the possibilities are checked - between every pair of neigbouring genes in the chromosome (main path) we can insert any gene different than these two (the graph is full and gene duplicates in the chromosome are allowed) and t_{max} is the only limit. From this set of possibilities the best one is chosen - it depends on the heuristic used. The first heuristic selects insertion which results in the greatest fitness gain. The second one is based on finding the greatest $fitness/travelTime$ ratio in the newly mutated individual. Normally during inserting mutation one edge is removed and two edges are added. In this version of the algorithm any edge could be a path (virtual edge). Thus, the algorithm operates on whole paths and heuristics are more complex - when calculating fitness and travel time, every hidden vertex in virtual edges has to be considered.

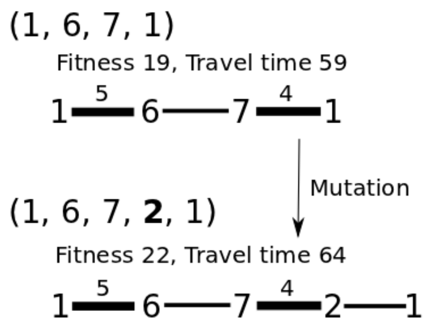

Fig. 6. An example of mutation (t_{max}=80, max $fitness/travelTime$ heuristic), inserted gene in bold

In the example presented in fig. 6 we have chromosome (1, 6, 7, 1). Edges 1-6 and 7-1 are virtual and the full path is 1-(5)-6-7-(4)-1. There are a few possibilities of inserting a new gene without exceeding t_{max}. The best is presented in the example - inserting gene 2 between genes 7 and 1 results in maximum $fitness/travelTime$ ratio of 0.344 (fitness 22, travel time 64). One of other options is inserting gene 3 between genes 7 and 1. The mutated chromosome would be (1, 6, 7, 3, 1) with the full path 1-(5)-6-7-3-(2)-1. Its fitness would be 24, which is better than in the chosen possibility, however with travel time of 70 its $fitness/travelTime$ ratio is slightly worse (0,343).

Removing mutation also checks all possibilities - any gene (except first and last) could be removed. There are several heuristics presented - two of them are the same as those in the inserting mutation (greatest fitness loss (could be less than 0) and greatest $fitness/travelTime$ ratio in a mutated individual). The third heuristic chooses the option with the greatest $fitness^2/travelTime$ ratio. Evaluating new fitness and travel time is performed in a similar way to the inserting mutation - if some removed or inserted edge is virtual, the algorithm takes into account all vertices in the shortest path. Thus, calculation is more complex but it is compensated by shorter chromosomes than those in the first algorithm version (less genes in the main path).

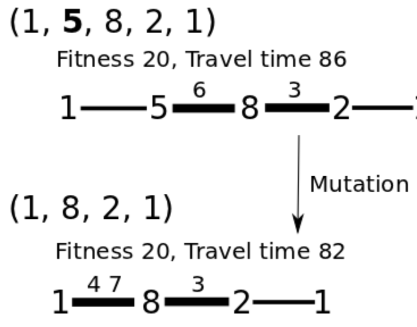

Fig. 7. An example of mutation ($t_{max} = 100$, max $fitness^2/travelTime$ heuristic), removed gene in bold

In the chromosome presented in fig. 7 there are three potential candidates to remove - genes 5, 8 and 2. If gene 5 is extracted (shown in the example) fitness (20) is unchanged and travel time is shorter (82) - the resulted $fitness^2/travelTime$ is 4.88, which is the best of all possibilities. During this operation we remove edge 1-5 and virtual edge 5-(6)-8. Instead, shortest path 1-(4-7)-8 from vertex 1 to vertex 8 is added. The second option is eliminating gene 8 and obtaining the chromosome (1, 5, 2, 1) - the real path would be much shorter: 1-5-(4)-2-1 (travel time 47, fitness 13, $fitness^2/travelTime3,60$). The third option (removing gene 2) results in the chromosome (1, 5, 8, 1) (real path 1-5-(6)-8-(7-4)-1). With fitness (20) and travel time (86) unchanged, its $fitness^2/travelTime$ ratio would be 4.65.

4. Experimental results

Experiments were conducted on the real road network of 306 cities in Poland. The tested data of the network can be found on the website [20] in two text files: cities.txt and distances.txt. The network file (distances.txt) was created from a real map - it includes main segments of roads from the whole Poland. The capital of Poland, Warsaw, was established as the central depot (vertex 1). Profits associated with a given city (written in the file cities.txt) were determined according to the number of inhabitants in the city. The more inhabitants the higher profit associated with a given city but its maximum value is 5. The profit was calculated as follows:

$$profit = \frac{inhabitants}{10000} \tag{1}$$

Both versions of the algorithm (for complete and incomplete graph) with several different heuristic variants were tested on the network presented above and its results were compared. The chosen heuristics were among the best of all heuristics tested on the transport network used in our experiment. Algorithm parameters were the same in almost all runs ($P_{size} = 300$, $t_{size} = 3$, $N_g = 100$). Only number of generations (N_g) is increased to 150 in the last two presented comparisons for IG. Increasing P_{size} brings minor changes in both versions and increasing t_{max} in IG has similar but generally less pronounced effects than changing N_g. CG converges more quickly and in this case increasing algorithm parameters results in marginal differences. Experiments were conducted on six t_{max} values: 500, 1000, 1500, 2000, 2500, 3000. The result of a given algorithm run is the best chromosome (highest fitness) from the final population. Statistics were obtained from 30 runs of each algorithm variant. Analysed parameters were: average result (mean), 95% confidence interval (CI) for mean and the best result (max) from 30 algorithm runs.

First, algorithms without removing mutation were compared. Two different inserting heuristics (highest fitness gain, highest $fitness/travelTime$ ratio after mutation) were chosen.

Table 3. Results compared (inserting mutation: highest fitness gain, no removing mutation)

	Complete graph version (CG)			Incomplete graph version (IG)		
t_{max}	Mean	CI for mean	Max	Mean	CI for mean	Max
500	59.6	±2.3	79	44.2	±3.2	65
1000	102.3	±3.6	118	86.7	±5.7	142
1500	127.1	±5.3	160	140.3	±10.3	193
2000	161.3	±6.5	215	188.6	±8.4	227
2500	179.2	±8.8	227	221.4	±8.1	264
3000	200.1	±6.6	244	241.9	±9.8	285

When using highest fitness gain heuristic (tab. 3), CG performs better for the lowest t_{max} values. However, IG is significantly better for higher t_{max} (>1000) in terms of both average and best results.

Table 4. Results compared (inserting: highest $fitness/travelTime$ after mutation, no removing mutation)

	CG			IG		
t_{max}	Mean	CI for mean	Max	Mean	Ci for mean	Max
500	61.8	±3.4	79	46.9	±2.9	63
1000	112.6	±5.0	149	99.6	±7.6	157
1500	140.7	±7.9	209	160.9	±11.6	206
2000	169.9	±8.1	209	202.8	±8.3	237
2500	193.3	±9.5	253	228.3	±10.2	267
3000	220.5	±10.7	295	254.7	±7.4	287

Results of both algorithms improve when using heuristic of highest $fitness/travelTime$ ratio after mutation. The tendency is the same as in the previous heuristic - while CG is on average better for lower t_{max}, IG excels for higher t_{max}. However, the best results (max) from 30 runs are more similar than in the previous heuristic, with CG better even for some higher t_{max} values (i.e. 3000).

After first comparisons removing mutation was added to both versions of GA. Lowest fitness loss heuristic (when removing) combined with highest $fitness/travelTime$ ratio after mutation (when inserting) was the best combination for complete graph version of GA and one of the best variants (in terms of mean results) for incomplete graph algorithm version.

Table 5. Results compared (inserting: highest $fitness/travelTime$ after mutation, removing: lowest fitness loss)

	CG			IG		
t_{max}	Mean	CI for mean	Max	Mean	Ci for mean	Max
500	61.9	±3.5	92	52.3	±3.2	71
1000	109.5	±5.4	144	92.7	±6.2	123
1500	146.7	±8.8	204	151.5	±9.7	192
2000	190.6	±9.7	248	198.2	±8.0	241
2500	219.1	±10.7	281	228.6	±6.9	264
3000	256.9	±11.4	320	244.7	±8.8	279

It can be seen that removing mutation significantly improved results of CG for higher t_{max} values. At the same time performance of IG slightly dropped. As a result, CG is still better for lower t_{max} values but its mean values are also closer for higher t_{max}. What is more, for longest routes (t_{max} 2500-3000) CG achieves highest maximum values.

There is one more combination of heuristics regarding incomplete graph algorithm version which should be mentioned. When inserting a gene we choose the option with highest $fitnessGain^2/travelTimeIncrease$ ratio. When removing we consider only genes which have some duplicate in the chromosome. If any of them are found, we choose the option with highest travel time loss. In terms of mean results this combination is similar to the version from the previous table but it is better for $t_{max} = 1000$. More importantly, it is one of algorithm variants which improves further for higher number of GA generations (N_g=150). It is shown in the tab. 6.

Table 6. Comparison 1 between results for N_g=100 and N_g=150 - incomplete graph version (inserting: highest $fitnessGain^2/travelTimeIncrease$ ratio, removing: highest travel time loss)

	$N_g = 100$			$N_g = 150$		
t_{max}	Mean	CI for mean	Max	Mean	Ci for mean	Max
500	52.4	±3.0	69	54.3	±3.0	69
1000	101.4	±7.1	149	105.6	±6.9	157
1500	162.3	±7.4	194	178.8	±9.1	215
2000	195.3	±7.4	230	213.5	±9.1	269
2500	225.4	±7.9	263	243.6	±9.5	288
3000	241.8	±7.1	270	261.6	±8.3	295

For mediocre and high t_{max} values there is an improvement of about 10% in results between 100th and 150th generation. Mean values for higher t_{max} are the best of all results obtained during the experiment. In tab. 7 there is one more comparison - inserting heuristic is the same as in tab. 6 but removing heuristic is not performed.

Table 7. Comparison 2 between results for N_g=100 and N_g=150 - incomplete graph version (inserting: highest $fitnessGain^2/travelTimeIncrease$ ratio, no removing mutation)

	$N_g = 100$			$N_g = 150$		
t_{max}	Mean	CI for mean	Max	Mean	Ci for mean	Max
500	46.7	±3.1	66	47.5	±3.3	66
1000	88.2	±6.0	132	88.3	±6.0	132
1500	153.2	±11.3	213	157.4	±13.5	225
2000	206.9	±10.3	249	216.6	±12.8	283
2500	222.2	±12.2	276	227.2	±13.9	292
3000	247.2	±12.3	295	257.3	±15.2	331

While mean results differ slightly, one can see a significant improvement of best runs for higher t_{max} - these are best routes obtained during the experiment.

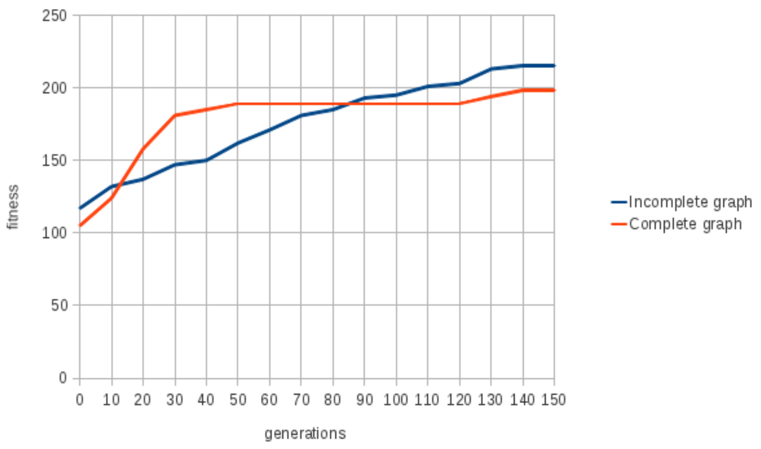

Fig. 8. Average fitness of the best individual from the n-th generation for both algorithms ($t_{max} = 2000$)
IG heuristic: inserting: highest $fitnessGain^2/travelTimeIncrease$, removing: highest travel time loss
CG heuristic: inserting: highest $fitness/travelTime$ after mutation, removing: lowest fitness loss

CG converges more quickly (as seen in fig. 8) and improvements between 100th and 150th generation are generally very small. While increasing number of generations leads to the best mean and maximum algorithm output in IG, it also results in more time consumption.

Table 8. Average execution time (in miliseconds) of both algorithm versions and various t_{max} values

t_{max}	500	1000	1500	2000	2500	3000
CG	60	90	120	140	170	220
IG ($N_g = 100$)	40	60	80	90	100	110
IG ($N_g = 150$)	60	90	110	120	140	160

One can see (table 8) that IG is faster than CG. It results from the fact that inserting mutation is more time consuming when the graph is complete - there are more different possibilities to check. Increasing N_g to 150 in IG lengthen execution time by 40-50% and then it is similar to CG time. Edge completion time in CG was not included. Algorithms were implemented in C++ and run on Intel Pentium M740 1.7 GHz CPU.

5. Conclusions

Comparison results show that the performance of both algorithm versions depends strongly on t_{max} value. For lower t_{max} (500-1000) complete graph version (CG) gives better mean results. It also gives routes with higher maximum profits. For higher t_{max} values incomplete graph version (IG) excels in terms of average results. This effect is even more pronounced when increasing N_g. In this case maximum results are also higher for IG. It can be also seen that the choice of heuristic influences the algorithm results i.e. adding removing mutation improves CG significantly contrary to IG.

It should be noted that a lot depends on a network itself. Results obtained using the network from our experiment can be different when using another network. Thus, it is recommended to check both algorithm version with various heuristics on a given network and we plan to test algorithms on even bigger network of 500 tourist facilities. It is also recommended to compare CG with GLS [11] on benchmark tests, as the latter is considered the most effective heuristic operating on complete graphs.

References

[1] I. Chao, B. Golden, E. Wasil, Theory and methodology - the team orienteering problem, European Journal of Operational Research 88, 464-474, 1996.

[2] A. Garcia, M.T. Linaza, O. Arbelaitz, P. Vansteenwegen, Intelligent Routing System for a Personalised Electronic Tourist Guide, Springer, 4, 185-197, 2009.

[3] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[4] B. Golden, L. Levy, R. Vohra, The orienteering problem, Naval Research Logistics 34, 1987.

[5] B. Golden, Q. Wang, L. Liu, A Multifaceted Heuristic for The Orienteering Problem, Naval Research Logistics, vol. 35, pp. 359-366, 1988.

[6] M. Hayes, J. M. Norman, Dynamic Programming in Orienteering: Route Choice and the Siting of Controls, Journal of the Operational Research Society, vol. 35, no. 9, pp. 791-796, 1984.

[7] G. Laporte, S. Martello, The Selective Traveling Salesman Problem, Discrete Applied Mathematics, vol. 26, pp. 193-207, 1990.

[8] J. Koszelew, Logistics for globetrotters - innovative software component for e-tourism, Logistics, vol. 6, 54-56, 2010.

[9] A. Piwonska, J. Koszelew, A Memetic Algorithm for a Tour Planning in the Selective Travelling Salesman Problem on a Road Network Springer, vol. 6804, 684-694, 2011.

[10] R. Ramesh, K. M. Brown, An Efficient Four-Phase Heuristic for the Generalized Orienteering Problem, Computers and Operations Research, vol. 18, no. 2, pp. 151-165, 1991.

[11] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. Vanden Berghe, D. Van Oudheusden, A personalized tourist trip design algorithm for mobile tourist guides, Applied Artificial Intelligence, 22:10, 964-985, 2008.

[12] W. Souffriau, P. Vansteenwegen, Tourist Trip Planning Functionalities: State-of-the-Art and Future, Springer, vol. 6385/2010, 474-485, 2010.

[13] M.F. Tasgetiren, A.E. Smith, A Genetic Algorithm for the Orienteering Problem, Proceedings of the 2000 Congress on Evolutionary Computation, San Diego, CA, pp. 1190-1195, 2000.

[14] F. Tricoire, M. Romauch, K. Doerner, R. Hartl, Heuristics for the multi-period orienteering problem with multiple time windows, Computers and Operations Research 37 (2), 351-367, 2010.

[15] T. Tsiligirides, Heuristic Methods Applied to Orienteering, Journal of Operational Research Society, vol. 35, no. 9, pp. 797-809, 1984.

[16] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, D. Van Oudheusden, A guided local search metaheuristic for the team orienteering problem, European Journal of Operational Research in Press, vol. 196, 2008.

[17] P. Vansteenwegen, W. Souffriau, D. Van Oudheusden, The orienteering problem: A survey, Elsevier, vol. 209, 2010.

[18] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, D. Van Oudheusden, The City Trip Planner: An expert system for tourists, Elsevier, vol. 38, 2011.

[19] Q. Wang, X. Sun, B.L. Golden, J. Jia, Using Artificial Neural Networks to Solve the Orienteering Problem, Annals of Operations Research, vol. 61, pp. 111-120, 1995.

[20] http://piwonska.pl/research

PORÓWNANIE ALGORYTMÓW GENETYCZNYCH ROZWIĄZUJĄCYCH ORIENTEERING PROBLEM PRZY POMOCY GRAFU PEŁNEGO I NIEPEŁNEGO

Streszczenie Celem pracy było porównanie dwóch odmian algorytmu (wersja dla grafu pełnego i niepełnego) rozwiązujących Orienteering Problem (OP). W większości artykułów dotyczących OP graf jest pełny, a jego krawędzie spełniają nierówność trójkąta, natomiast w naszej wersji takie założenia mogą nie być spełnione. Może to być bardziej praktyczne ponieważ sieci transportowe są grafami, ktore nie muszą spełniać tych warunków. W takich przypadkach grafy sa zazwyczaj uzupełniane fikcyjnymi krawędziami, a następnie działają na nich algorytmy rozwiązujące klasyczną wersje OP, które operują na grafie pełnym. Artykuł odpowiada na pytanie: czy pod względem jakości wyników i czasu obliczeń lepiej jest przekształcać graf do klasycznej formy OP przed uruchomieniem algorytmu w wersji dla grafu pełnego czy rozwiązywać OP na grafie niezmienionym i nie spełniającym dodatkowych założen (wersja dla grafu niepełnego)? Eksperyment został przeprowadzony na prawdziwej sieci transportowej w Polsce, a jego wyniki sugerują, że warto sprawdzać obie wersje algorytmu na konkretnych sieciach.

Słowa kluczowe: orienteering problem, OP, sieć transportowa, algorytm genetyczny, GA, grap pełny, graf niepełny

Artykul zrealizowano w ramach pracy badawczej S/WI/01/10.