Evolutionary Algorithm for the Time-Dependent Orienteering Problem

Krzysztof Ostrowski(✉)

Faculty of Computer Science, Bialystok University of Technology, Bialystok, Poland
k.ostrowski@pb.edu.pl

Abstract. The Time-Dependent Orienteering Problem (TDOP) is a generalization of the Orienteering Problem where graph weights vary with time. It has many real life applications particularly associated with transport networks, in which travel time between two points depends on the moment of start. The paper presents an evolutionary algorithm with embedded local search operators and heuristic crossover, which solves TDOP. The algorithm was tested on TDOP benchmark instances and in most cases achieved optimal or near optimal results clearly outperforming other published methods.

Keywords: Time-dependent orienteering problem · Evolutionary algorithm · Metaheuristic · Local search operators

1 Introduction

The Orienteering Problem (OP) can be formulated on a graph in which each vertex has assigned a profit and each edge has a cost. The goal of the OP is to find a path between given points limited by total cost which maximizes total profit of visited vertices. The OP has many practical applications including tourist trip planning [14], transport logistics and others. For example in tourist trip planning each attraction (point of interest - POI) has some profit assigned. In this case the purpose is to find as attractive trip as possible with limited duration. It can by solved by OP algorithms.

In this paper the Time-Dependent Orienteering Problem (TDOP) [13] is tackled. TDOP is a generalization of OP in which graph weights (indentified with travel times) vary with time. In practical situations, usually travel time between two places depends on the moment of travel start. Examples of time-dependent networks are public transport networks (travel times determined by timetables) as well as other transport networks (travel times determined by time of a day and traffic). Determining most profitable paths in such networks (i.e. most attractive trips for tourists) comes down to solving TDOP.

To solve the TDOP author proposed an evolutionary algorithm with local search methods embedded. It was based on the algorithm which author developed for classic OP obtaining solutions close to optimal for benchmarks up to a few hundred vertices [24]. Some modifications were applied to the algorithm: allowance of infeasible solutions (too long paths) in a population and adaptation of operators to time dependency.

The article is organized as follows. Literature review is presented in Sect. 2. Detailed mathematical problem definition is given in Sect. 3. In Sect. 4 there is a description of proposed evolutionary algorithm. Section 5 contains computational results of the algorithm and comparisons with other metaheuristics. Finally, in Sect. 6 conclusion and further research are included.

2 Literature Review

The Orienteering Problem was introduced by Tsiligirides [1]. The OP (and its generalizations) is an NP-hard optimization problem [2] and exact algorithms for OP can be very time-consuming for larger graphs. The exact algorithms usually applied for the OP are the branch-and-cut methods [5,7]. However, because of its computational difficulty, most OP papers are devoted to heuristics and metaheuristics. Various approaches were based i.a. on greedy and randomized construction of solutions [1,2,17], local search methods [4,10], tabu search [6], ant-colony optimization [11] and genetic algorithms [8]. Author's previous OP research concentrated on development of evolutionary algorithms with local search operators, which proved successful on benchmark instances [18,22,24,25].

The Time-Dependent Orienteering Problem (TDOP) is relatively new and most of papers regarding TDOP were published in current decade. Most of them emphasize practical application of TDOP especially in transport networks. Fomin et al. [9] presented first approach to the problem - they proposed $(2 + \epsilon)$-approximation algorithm for TDOP. The problem version tackled in the article was simplified because each node had the same profit (equal to 1). LI [13] developed an exact method basing on dynamic programming and solved small instances of TDOP. He used discretized time and there were 1440 time states (number of minutes in a day). It was the first article about classic TDOP (nodes profits can be arbitrary).

The TDOP solutions can be practically used in networks of tourist attractions connected by means of public transport. A few papers concentrated on TDOP application in such networks. In paper [15] first such an application was presented - algorithm was tested on POI and public transport network of Tehran. The authors used a different name to the problem and routes utility was computed in a different way to classic OP but the problem was closely related to TDOP.

Garcia et al. [16] for the first time applied classic TDOP version in a public transport network - experiments were conducted on POI and public transport network of San Sebastian. Their solution also took into account time-windows and multiple paths (Time-Dependent Team Orienteering Problem with Time Windows). In order to solve the problem they used Iterated Local Search method (ILS). However, they used some simplifications: computations performed on average daily travel time (first approach) and assumption of periodic public transport timetables (second approach).

Recently one of the first approaches which uses real time-dependent weights in public transport network was presented [23]. Experiments were conducted on POI and public transport network of Athens. Authors proposed 20 topologies and 100 tourist preferences combining into 2000 different test cases. Authors introduced two fast heuristics (TDCSCR and TDSlCSCR), which based on iterated local search and vertex clustering, and made comparisons with ILS.

Another solution of TDOP tested on real travel times of POI and public transport network (city of Bialystok) was proposed by the author of this paper [21]. The algorithm based on local search procedures and was tested in three variants which processed network weights in different ways. Time-dependent variant performed computation on real travel times, mean variant operated on static network of daily averaged travel times (like in approaches described above) and hybrid variant used information from both approaches.

Verbeeck et al. [19] developed new benchmark instances for TDOP. These networks model street traffic. The authors proposed ant-colony approach, tested it on mentioned benchmarks and compared with optimal solutions (obtained by them). The tested metaheuristic achieved high quality results in short execution time. Gunawan et al. [20] modified Verbeeck's benchmarks (discretization of time) and tested a few approaches: greedy construction, iterated local search (ILS), adaptive ILS and variable neighborhood descent. The best solutions were generated by adaptive ILS metaheuristic.

3 Definition of the Time-Dependent Orienteering Problem

Let $G = (V, E)$ be a directed, weighted graph. In this paper each weight between vertices i and j ($i, j \in V$) is identified with travel time between these vertices and is a function $w_{ij}(t)$ dependent on the moment of travel start t. Each vertex i has a nonnegative profit p_i and a nonnegative visit time τ_i. Given the time-dependent graph, moment of start t_0, time limit T_{max}, start and end vertices (s and e) the purpose of TDOP is to find a path from s to e starting at time t_0 which maximizes total profit of visited vertices and its total cost (travel time) does not exceed T_{max}. TDOP can be formulated as Mixed Integer Programming problem. Let x_{ij} is 1 iff a path contains direct travel from i to j and 0 otherwise. Let y_i is 1 iff a path contains vertex i and 0 otherwise. Let t_i be a time of arrival at vertex i - this function is defined only for vertices contained in a path. The purpose of TDOP is to maximize formula 1 without violating constraints given by formulas 2–8. Formula 2 limits total time budget. Formula 3 indicates that the path starts in s and ends in e. Constraint 4 means that each vertex can be visited only once while formula 5 is a relation between variables x and y. Formulas 6 and 7 are used to compute arrival times at visited vertices. Constraint 8 guarantees that there are no sub cycles.

$$max \sum_{i \in V} (p_i \cdot y_i) \tag{1}$$

$$t_e + \tau_e - t_0 \leq T_{max} \tag{2}$$

$$\sum_{j \in V} x_{sj} = \sum_{i \in V} x_{ie} = 1 \tag{3}$$

$$\underset{k \in V \setminus \{s,e\}}{\forall} (\sum_{i \in V} x_{ik} = \sum_{j \in V} x_{kj} \leq 1 \tag{4}$$

$$\underset{i \in V}{\forall} (y_i = max(\sum_{j \in V} x_{ij}, \sum_{j \in V} x_{ji})) \tag{5}$$

$$t_s = t_0 \tag{6}$$

$$\underset{i \in V, j \in V}{\forall} (x_{ij} = 1 \Rightarrow t_j = t_i + \tau_i + w_{ij}(t_i + \tau_i)) \tag{7}$$

$$\underset{S \subset V}{\forall} (\sum_{i \in S} y_i < \sum_{i \in V} y_i \Rightarrow \sum_{i,j, \in S} x_{ij} < \sum_{i \in S} y_i) \tag{8}$$

4 Proposed Algorithm Description

To solve the TDOP author proposed an evolutionary algorithm with local search methods embedded (during mutation), disturb operator, 2-point crossover and deterministic crowding as selection mechanism. Operators have two forms (random and heuristic) - their frequency and specificity are controlled by algorithm parameters. Infeasible solutions (too long paths) can be included in a population as well and they are penalized by fitness function. A path representation is used in the proposed algorithm - each gene in a chromosome is associated with visited vertex in a corresponding path. After random initialization of paths, procedures of crossover, selection, mutation and disturbance and applied repeatedly. Evolution terminates after N_g generations or earlier if there is no improvement in the last C_g generations. The result is the best feasible path obtained during algorithm run.

4.1 Evalutation

Let S be a path with total profit p_S and total cost (travel time) c_S. Fitness function of a feasible path (not exceeding T_{max}) is equal to its profit. If path S is too long its fitness is: $fitness(S) = p_S \cdot (\frac{T_{max}}{c_S})^k$. Fitness function decreases as solution is farther from feasibility boarder T_{max}. Parameter k is associated with penalty severity: larger values mean stronger penalties. Fitness function is adaptive: at the beginning $k = 1$ and every 10 generations there is a check - if more than α percent of individuals are infeasible then k is increased by 0.1.

4.2 Initialization

At the beginning initial population of P_{size} random paths is generated. Construction of each solution begins with adding start vertex s. Afterwards, random vertices are iteratively added at the end of the path as long as further insertions are possible without violating T_{max} constraint (including cost of traveling back to end vertex).

4.3 Crossover

During crossover $\frac{P_{size} \cdot c_p}{2}$ different pairs of parents are randomly selected from the population (c_p - crossover probability) and each pair undergoes crossover procedure. A specialized 2-point crossover operator was used. At the beginning an ordered set S of common vertices for both parents is determined. Vertices order in the set is the same as in first parent. Next, chromosome fragments between two successive vertices in S are exchanged and two offspring are created. If any duplicates appear in offspring individuals outside of exchanged fragments they are immediately removed. Crossover can be performed in two ways depending on how exchanged fragments are chosen. Random crossover variant chooses crossing points randomly while heuristic crossover maximizes fitness function of the fitter offspring (all exchange options are checked). Created offspring can exceed T_{max} limit. The probability of using heuristic crossover is indicated by parameter c_h while the probability of using random version is $1 - c_h$. In Fig. 1 there is an example of crossover.

parents

parent A: (**16**, 3, 7, **4**, 5, **9**, 12, 11, 14, **1**)
parent B: (**16**, 6, 8, 2, **4**, 13, 10, **9**, 15, **1**)

crossover variant I

child 1: (**16**, 6, 8, 2, **4**, 5, **9**, 12, 11, 14, **1**)
child 2: (**16**, 3, 7, **4**, 13, 10, **9**, 15, **1**)

crossover variant II

child 1: (**16**, 3, 7, **4**, 13, 10, **9**, 12, 11, 14, **1**)
child 2: (**16**, 6, 8, 2, **4**, 5, **9**, 15, **1**)

crossover variant III

child 1: (**16**, 3, 7, **4**, 5, **9**, 15, **1**)
child 2: (**16**, 6, 8, 2, **4**, 13, 10, **9**, 12, 11, 14, **1**)

Fig. 1. Example of 2-point crossover. Given two parents which have 4 common vertices (16, 4, 9, 1) there are three possible fragment exchanges between successive points. Variant I shows children created by fragments exchange between vertices 16 and 4, variant II is created by segments exchange between points 4 and 9 while variant III shows offspring individuals created by exchange of fragments between vertices 9 and 1.

4.4 Selection

In the algorithm no parent selection is used. Instead, survival selection is applied in the form of deterministic crowding [3]. After crossover procedure ends each offspring competes with one of its parents and the fitter individual is chosen to the next generation. To preserve population diversity competition is performed between more similar child-parent pairs. Pair arrangement is determined by distance measure. In this problem two distance measures are applied: edit distance (operations insert and delete) and cardinality of symmetric difference of vertex sets. The second is computed initially - if it suggests that child-parent pairs should be swapped then more complex procedure of edit distance is calculated.

4.5 Mutation

During mutation phase $P_{size} \cdot m_p$ individuals are randomly selected from the population (m_p is mutation probability). First, each chosen individual undergoes 2-opt procedure. This procedure tries to replace two non-adjacent edges in a path by two other edges in order to shorten the path as much as possible. Given a path $(v_1, v_2, ..., v_i, v_{i+1}, ..., v_j, v_{j+1}, ..., v_k)$ 2-opt procedure replaces edges (v_i, v_{i+1}) and (v_j, v_{j+1}) by edges (v_i, v_j) and (v_{i+1}, v_{j+1}) and reverses path fragment between v_{i+1} and v_j. The resulting path is:
$(v_1, v_2, ..., v_i, v_j, v_{j-1} ..., v_{i+1}, v_{j+1}, v_{j+2} ..., v_k)$.

Afterwards one vertex insertion or one vertex deletion is carried out. Probabilities of insertion and deletion are the same (0.5). Both insertion and deletion can be either random or heuristic (local search). Heuristic insertion from all possible non-included vertices (and all insertion places) chooses the option that maximizes ratio of vertex profit to path travel time increase. Random insertion chooses new vertex randomly but insertion place is chosen to minimize travel time increase. Heuristic deletion chooses the vertex that minimizes ratio of vertex profit to path travel time decrease. During mutation T_{max} limit can be exceeded. The ratio between local search and randomness is determined by parameter m_h (analogically to c_h in crossover phase).

4.6 Disturb

Disturb procedure can be perceived as a different type of mutation which causes bigger changes in routes. The operator chooses $P_{size} \cdot d_p$ individuals from the population (d_p is disturb probability). Each chosen individual is disturbed - a path fragment of random length (but containing at most 10 percent of all path vertices) is removed. Two types of disturb are possible: first version removes a random fragment of random length while heuristic version works analogically to deletion mutation - from all possible fragments of a given length it chooses the option minimizing ratio of fragment profit to path travel time decrease. Specificity of this operator is determined by parameter d_h.

4.7 Adaptation to Time Dependency

TDOP algorithm operates on time-dependent networks and for this reason each path modification is associated with recalculation of edge travel times after the modification point. This can be particularly time-consuming in case of heuristic operators. For this reason some modifications were applied to these parts.

Vertex insertion: After inserting new vertex all edge travel times after insertion place have to be recalculated. Insertion operators of the algorithm choose the best insertion place for an included vertex (minimizing travel time increase). Trying each insertion place and recalculating weights is time consuming. However, it is possible to determine the best insertion place in $O(N)$ run time (N - number of nodes in a path). The purpose of insertion is to minimize path travel time increase but it can be replaced by an equivalent goal of earliest possible arrival into the last vertex in a modified path. It can be computed in one loop over the path. In the i-th iteration we compare two arrival times into vertex i: one resulting from insertion new vertex directly before i and one resulting from best insertion place chosen so far. If current insertion place is better than previous best (arrival time into vertex i is earlier) we update our best choice.

2-opt: The purpose of 2-opt is to decrease path travel time as much as possible by edges replacement. Each edge exchange is associated with recalculating travel times in a large fragment of a path (part of it is reversed). In order to decrease execution time of 2-opt some modifications were applied - instead of time-consuming repeated recalculations of travel times the procedure tries to estimate total travel time of reversed path fragments. At first the initial travel time of the path (before changes) is divided into equal time intervals. Afterwards some reversed path fragments (starting at different times) are precomputed. Estimation is fast and base on finding an appropriate precomputed path (explanation in Fig. 2). Exact calculations are performed only if estimation signals that travel time can be shortened - it significantly reduces computations.

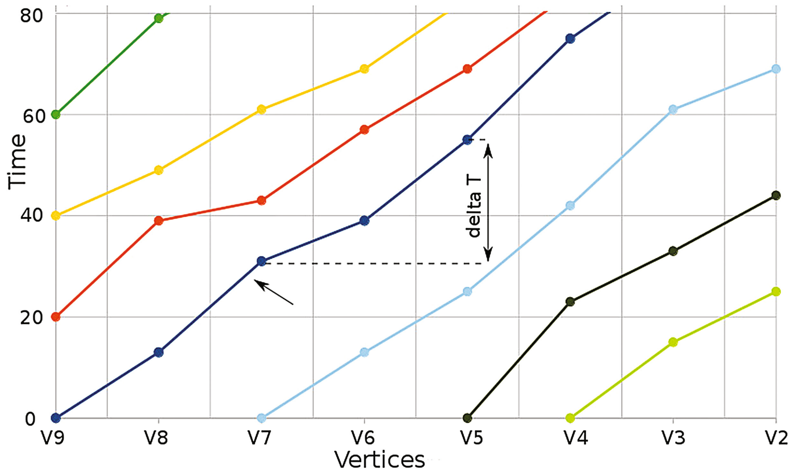

Fig. 2. Estimating travel time of reversed fragment of a path. The path is $(v_1, v_2, ..., v_{10})$ and it starts at time $t = 0$ and ends at time 80. Time was divided into $D = 4$ equal intervals and all possible vertices in reversed fragments are v_9, v_8, ... v_2. The figure can be treated as 2-dimensional array and in each cell there is a precomputed path fragment visiting a given vertex (determined by column) at a given time interval (determined by row). For example in order to estimate travel time of fragment (v_7, v_6, v_5) starting at time $t = 27$ we should find an appropriate column (v_7) and row (interval 20–40) and route fragment associated with it (pointed by arrow). Afterwards we compute the difference of arrival times between v_5 and v_7 (deltaT), which is the estimation.

5 Experimental Results

5.1 Parameters

Experiments were conducted on a computer with Intel Core i7 3.5 GHz processor and algorithms were implemented in C++. In Table 1 there are values of all parameters of proposed evolutionary algorithm. Population size was set to 100 or 30. The first population size ($P_{size} = 100$) is derived from original OP algorithm, which performed computations on larger networks. The smaller population size ($P_{size} = 30$) was introduced in order to reduce execution time in smaller TDOP instances. Parameters associated with generations were adjusted to population size - their values allow algorithm to fully converge. The remaining parameters of the algorithm were determined in a process of automatic tuning. The author used Parameters ILS [12] meta-algorithm to calibrate his method. The parameters of evolutionary algorithm were tuned on classic OP calibration instances but proved to be applicable to TDOP instances as well. More details about tuning can be found in one of author's previous papers on the OP [24].

Table 1. Parameters of the evolutionary algorithm

Param.	Value	Description	Param.	Value	Description
P_{size}	100/30	Population size	m_p	1	Mutation probab.
N_g	5000/1500	Max. generations number	c_p	1	Crossover probab.
C_g	500/150	Max. generations number without improvement	d_p	0.01	Disturb probab.
			m_h	1	Mutation heuristic coeff.
α	90	Max. percentage of infeasible solutions	c_h	0.8	Crossover heuristic coeff.
			d_h	0,8	Disturb heuristic coeff.

5.2 Test Instances

TDOP benchmark instances were introduced by Verbeeck et al. [19]. These test instances model street traffic. The benchmark authors introduced 7 networks containing 21–102 vertices. Maximum travel times (T_{max}) were between 5 and 14 hours and all paths started at 7 in the morning in node 1 and ended in last node: N. There are three classes of benchmark networks:

– Class I contains the smallest networks (21–32 vertices). These are original TDOP instances which were solved optimally by benchmark authors.
– Class II contains all benchmark networks - they were slightly modified so that optimal results were the same as in static OP network. Details in [19].

– Class III instances were developed from previous classes by applying time discretization in order to compare author's algorithm with Gunawan's metaheuristics [20]. Original start time was 7 am and time limit 14 h.

5.3 Results

The author tested the evolutionary algorithm (EVO) on these instances and compared it to optimal solutions, ant-colony metaheuristic (ACS) [19] and Adaptive ILS [20]. Gaps were expressed in percent and computed as $100 \cdot (1 - \frac{res}{opt})$ where res is result obtained by a metaheuristic (average from 30 runs in case of EVO) and opt is profit of optimal solution. Execution time is given in seconds.

In Table 2 results of class I instances are given: performance of two evolutionary algorithm versions (P_{size} of 100 and 30) is presented and compared to ACS and optimal solutions. EVO100 obtains optimal solutions for all test cases and is on average 0.8 percent better than ACS. The biggest obtained gap is over 4 percent. EVO30 is only 0.1 percent worse than EVO100 and is much faster. Only in one case EVO30 is much worse than EVO100 (over 1.5%).

Table 2. Results for class I instances. N is network size.

N	T_{max}	EVO100		EVO30		ACS	Opt.	N	T_{max}	EVO100		EVO30		ACS	Opt.
		Time	Gap	Time	Gap	gap	result			Time	Gap	Time	Gap	gap	result
32	5	0.8	0	0.1	0	0	115	21	9	0.6	0	0.1	1.5	0	260
32	6	1.0	0	0.1	0	0	135	21	10	0.7	0	0.1	0	0	310
32	7	1.1	0	0.1	0	0	160	21	11	0.7	0	0.1	0	0	340
32	8	1.2	0	0.1	0	2.2	185	21	12	0.8	0	0.1	0.2	0	375
32	9	1.4	0	0.1	0	0.5	210	21	13	0.9	0	0.1	0	0	425
32	10	1.6	0	0.2	0	0.4	230	33	5.5	0.8	0	0.1	0	4.3	370
32	11	1.6	0	0.2	0	0	250	33	6.5	0.9	0	0.1	0	0	420
32	12	1.8	0	0.2	0	1.9	270	33	7.5	1.1	0	0.1	0	4	500
21	5	0.5	0	0.1	0	0	100	33	8.5	1.2	0	0.1	0	2.9	560
21	6	0.5	0	0.1	0	0	150	33	9.5	1.4	0	0.1	0	1.9	620
21	7	0.6	0	0.1	0	0	195	33	10.5	1.6	0	0.1	0	0.3	650
21	8	0.6	0	0.1	0.3	0	220	33	11.5	1.8	0	0.2	0	1.2	690
									avg.	**1.0**	**0**	**0.1**	**0.1**	**0.8**	**322.5**

In Table 3 results of class II instances are presented. EVO100 again achieves optimal solutions for most test cases and average gap is close to 0 percent. On average it is 1.4 percent better than ACS and for some instances the difference is more than 5 percent. EVO100 clearly outperforms ACS especially in case of some larger instances. EVO30 is on average only 0.1 percent worse than EVO100 but for some of largest test cases (100–102 nodes) the difference are larger (0.5–1.1 percent). Execution time of proposed methods is also acceptable.

Table 3. Results for class II instances.

N	T_{max}	EVO100		EVO30		ACS	Opt.	N	T_{max}	EVO100		EVO30		ACS	Opt.
		Time	Gap	Time	Gap	gap	result			Time	Gap	Time	Gap	gap	result
32	5	0.9	0	0.1	0	0	135	100	5	2.3	0	0.3	0.5	0.8	486
32	6	1.0	0	0.1	0	0	165	100	6	2.7	0	0.3	0.1	1.7	590
32	7	1.3	0	0.1	0	0	185	100	7	4.5	0.2	0.4	0.3	2.5	679
32	8	1.3	0	0.1	0	0	210	100	8	4.0	0	0.5	0.4	4.5	771
32	9	1.5	0	0.1	0	0	240	100	9	5.0	0	0.7	0.6	4.2	853
32	10	1.7	0	0.1	0	1.5	260	100	10	6.1	0.02	0.8	1.1	3	932
32	11	1.7	0	0.2	0	0	275	100	11	6.4	0	0.9	0.2	0.9	1007
32	12	1.9	0	0.2	0	0	285	100	12	8.8	0	1.1	0.1	3	1083
32	13	1.9	0	0.2	0	0	285	100	13	7.9	0	1.2	0.1	1.7	1147
21	5	0.6	0	0.1	0	0	165	100	14	9.9	0	1.2	0.6	5.3	1198
21	6	0.7	0	0.1	0	0	200	64	6.5	2.6	0	0.3	0	0.3	870
21	7	0.6	0	0.1	0	0	225	64	7	2.9	0	0.4	0.04	0.6	930
21	8	0.7	0	0.1	0	0	275	64	8	3.4	0	0.4	0.1	0.9	1056
21	9	0.7	0	0.1	0	0	315	64	9	3.9	0	0.4	0	0.8	1152
21	10	0.8	0	0.1	0	0	375	64	10	4.5	0	0.5	0.02	1.1	1236
21	11	0.8	0	0.1	0	0	415	64	11	6.1	0	0.8	0.4	1.8	1308
21	12	0.8	0	0.1	0	0	440	64	12	6.2	0	0.7	0.01	1.5	1344
21	13	0.9	0	0.1	0	0	450	64	13	6.0	0	0.6	0	0	1344
33	5.5	0.9	0	0.1	0	0	430	64	14	6.1	0	0.6	0	0	1344
33	6.5	1.0	0	0.1	0	0	490	102	4	2.3	0	0.3	0	5.3	532
33	7.5	1.3	0	0.1	0	0	550	102	5	2.9	0	0.4	0	6.1	648
33	8.5	1.3	0	0.1	0	0	590	102	6	3.9	0	0.5	0.2	1.1	774
33	9.5	1.5	0	0.2	0	0	630	102	7	5.0	0	0.6	0	1.8	884
33	10.5	1.7	0	0.1	0	0	680	102	8	6.5	0	0.8	0.1	6.1	994
33	11.5	1.8	0	0.2	0	0	730	102	9	7.2	0	1	0.1	2.5	1090
33	12.5	1.9	0	0.2	0	1	770	102	10	7.7	0	1.1	0.01	3.1	1175
33	13.5	2.0	0	0.2	0	0.8	800	102	11	8.6	0	1.4	0	4.6	1251
66	5.5	1.5	0	0.2	0	0.3	580	102	12	9.6	0	1.3	0	3.2	1317
66	6	1.8	0.03	0.2	0.2	2.3	650	102	13	15.5	0.02	2	0.5	5.6	1368
66	7	2.5	0.04	0.3	0.3	1.6	770		**avg.**	**3.5**	**0.01**	**0.4**	**0.1**	**1.4**	**710.7**

In Table 4 results for class III instances are compared. The proposed method was compared with Gunawan's Adaptive ILS method (which achieved generally the best results among heuristics tested by them). Both versions of EVO are on average 1.7–2 percent better than Adaptive ILS and for most instances achieve optimal results or close to best known solutions (average gap 0.2–0.5 percent). EVO30 performance is generally slightly worse than EVO100 but it is about 1–1.5 percent worse than EVO100 for a few larger networks. Gunawan's heuristics had execution time limit of 1 second (this limit was also used in case of EVO30).

Table 4. Results for class III instances. Symbol d is discretization step (in minutes).

d	N	EVO100		EVO30		Ad.ILS	Best	d	N	EVO100		EVO30		Ad.ILS	Best
		Time	Gap	Time	Gap	gap	result			Time	Gap	Time	Gap	Gap	result
1	32	1.7	0	0.2	0.0	0.1	285	15	32	1.2	3.8	0.1	3.8	0	260
	21	0.8	0	0.1	0	0	450		21	0.6	0	0.1	0	2.5	405
	33	1.8	0	0.2	0	0.3	780		33	1.4	0.3	0.1	1.3	2.5	680
	100	16.4	0.1	1.0	0.6	8.4	1145		100	5.0	0.1	0.6	0.9	5.9	864
	66	6.7	0	0.8	0.2	1.0	1513		66	1.7	0	0.2	0	0.6	870
	64	7.9	0	0.8	0	0.5	1344		64	4.9	0	0.5	0.6	1.6	1254
	102	36.6	0.1	1.0	1.4	6.5	1372		102	9.3	0	1.0	0.5	4.5	1068
5	32	1.5	0	0.2	0	0	280	30	32	0.9	0	0.1	0.2	0	220
	21	0.7	0	0.1	0	2.2	450		21	0.5	0	0.0	0	0	355
	33	1.5	0	0.1	0	2.1	760		33	1.1	0	0.1	0	0	590
	100	9.3	0	1.0	0.6	7.6	1032		100	2.4	1.2	0.3	1.7	3.5	623
	66	3.2	0	0.4	0.3	1.0	1260		66	1.6	0	0.2	0	0.2	850
	64	6.1	0	0.6	0.5	2.9	1274		64	1.5	0	0.1	0	0	768
	102	14.4	0	1.0	1.5	6.8	1244		102	2.6	0.1	0.3	0.3	1.1	690
									avg.	**5.1**	**0.2**	**0.4**	**0.5**	**2.2**	**810.2**

6 Conclusions and Further Research

The experiments show that proposed evolutionary algorithm with memetic operators is an effective metaheuristic for the Time-Dependent Orienteering Problem (TDOP). The results obtained for TDOP benchmarks by the algorithm are optimal or close to optimal in most cases and are better than those obtained by other metaheuristics. Author's method, which was successfully used to tackle the classic Orienteering Problem (OP), achieved high quality results for TDOP benchmarks as well. Promising results suggest that proposed method should be adapted to other related problems. Therefore further research will concentrate on application of the algorithm for other problems from OP family i.e. variants with multiple paths (Team OP and Team TDOP) and with time-windows.

Acknowledgements. The author is grateful for support from the Polish Ministry of Science and Higher Education at the Bialystok University of Technology (S/WI/1/2014).

References

1. Tsiligirides, T.: Heuristic methods applied to orienteering. J. Oper. Res. Soc. **35**(9), 797–809 (1984)
2. Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Res. Logistics **34**, 307–318 (1987)
3. Mahfoud, S.W.: Crowding and preselection revisited. In: Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN II), Brussels, Belgium, 1992, pp. 27–36, Elsevier, Amsterdam (1992)

4. Chao, I., Golden, B., Wasil, E.: Theory and methodology - a fast and effective heuristic for the orienteering problem. Eur. J. Oper. Res. **88**, 475–489 (1996)
5. Gendreau, M., Laporte, G., Semet, F.: A branch-and-cut algorithm for the undirected selective traveling salesman problem. Networks **32**(4), 263–273 (1998)
6. Gendreau, M., Laporte, G., Semet, F.: A tabu search heuristic for the undirected selective travelling salesman problem. Eur. J. Oper. Res. **106**, 539–545 (1998)
7. Fischetti, M., Salazar, J., Toth, P.: Solving the orienteering problem through branch-and-cut. Informs J. Comput. **10**, 133–148 (1998)
8. Tasgetiren, M.: A genetic algorithm with an adaptive penalty function for the orienteering problem. J. Econ. Soc. Res. **4**(2), 1–26 (2001)
9. Fomin, F.V., Lingas, A.: Approximation algorithms for time-dependent orienteering. Inf. Process. Lett. **83**, 57–62 (2002)
10. Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Oudheusden, D.V.: A guided local search metaheuristic for the team orienteering problem. Eur. J. Oper. Res. **196**(1), 118–127 (2009)
11. Schilde, M., Doerner, K., Hartl, R., Kiechle, G.: Metaheuristics for the biobjective orienteering problem. Swarm Intell. **3**, 179–201 (2009)
12. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stutzle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
13. Li, J.: Model and algorithm for time-dependent team orienteering problem. Commun. Comput. Inf. Sci. **175**, 1–7 (2011)
14. Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: The city trip planner: an expert system for tourists. Expert Syst. Appl. **38**(6), 6540–6546 (2011)
15. Abbaspour, R.A., Samadzadegan, F.: Time-dependent personal tour planning and scheduling in metropolises. Expert Syst. Appl. **38**(10), 12439–12452 (2011)
16. Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaza, M.T.: Integrating public transportation in personalised electronic tourist guides. Comput. Oper. Res. **40**(3), 758–774 (2013)
17. Campos, V., Marti, R., Sanchez-Oro, J., Duarte, A.: Grasp with path relinking for the orienteering problem. J. Oper. Res. Soc. **156**, 1–14 (2013)
18. Koszelew, J., Ostrowski, K.: A genetic algorithm with multiple mutation which solves orienteering problem in large networks. In: Bădică, C., Nguyen, N.T., Brezovan, M. (eds.) ICCCI 2013. LNCS, vol. 8083, pp. 356–366. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40495-5_36
19. Verbeeck, C., Sörensen, K., Aghezzaf, E.H., Vansteenwegen, P.: A fast solution method for the time-dependent orienteering problem. Eur. J. Oper. Res. **236**(2), 419–432 (2014)
20. Gunawan, A., Yuan, Z., Lau, H.C.: A mathematical model and metaheuristics for time dependent orienteering problem. Angewandte Mathematik und Optimierung Schriftenreihe AMOS 2014 (2014)
21. Ostrowski, K.: Comparison of different graph weights representations used to solve the time-dependent orienteering problem. Trends in Contemporary Computer Science, Podlasie 2014, pp. 144–154. Bialystok University of Technology Publishing Office (2014)
22. Zabielski, P., Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K.: A genetic algorithm with grouping selection and searching operators for the orienteering problem. In: Nguyen, N.T., Trawiński, B., Kosala, R. (eds.) ACIIDS 2015. LNCS, vol. 9012, pp. 31–40. Springer, Cham (2015). doi:10.1007/978-3-319-15705-4_4

23. Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Vathis, N.: Heuristics for the time dependent team orienteering problem: application to tourist route planning. Comput. Oper. Res. **62**, 36–50 (2015)
24. Ostrowski, K.: Parameters tuning of evolutionary algorithm for the orienteering problem. Adv. Comput. Sci. Res. **12**, 53–78 (2015)
25. Ostrowski, K., Karbowska-Chilinska, J., Koszelew, J., Zabielski, P.: Evolution-inspired local improvement algorithm solving orienteering problem. Ann. Oper. Res. (2016). doi:10.1007/s10479-016-2278-1