An Efficient Metaheuristic for the Time-Dependent Team Orienteering Problem with Time Windows

Krzysztof Ostrowski(✉)

Faculty of Computer Science and Telecommunication,
Bialystok University of Technology, ul. Wiejska 45A, 15-001 Bialystok, Poland
k.ostrowski@pb.edu.pl

Abstract. The Time-Dependent Team Orienteering Problem with Time Windows (TDTOPTW) is a combinatorial optimization problem defined on graphs. The goal is to find most profitable set of paths in time-dependent graphs, where travel times (weights) between vertices varies with time. Its real life applications include tourist trip planning in transport networks. The paper presents an evolutionary algorithm with local search operators solving the problem. The algorithm was tested on public transport network of Athens and clearly outperformed other published methods achieving results close to optimal in short execution times.

Keywords: Time-Dependent Team Orienteering Problem with Time Windows · Evolutionary algorithm · Local search · Public transport network

1 Introduction

The Time-Dependent Team Orienteering Problem with Time Windows (TDTO PTW) belongs to the family of the Orienteering Problem (OP). The classic OP is defined on a weighted graph with nonnegative profits associated to vertices and nonnegative costs associated to edges. The goal of the OP is to find a path between two given vertices, limited by total cost of visited edges and maximizing total profit of visited vertices. The OP solution does not have to contain all vertices (usually it is impossible because of total cost constraint) and each vertex can be visited only once.

The Time-Dependent Orienteering Problem (TDOP) [12] is a generalization of the classic OP and is defined on time-dependent graphs. In such graphs edge costs (weights) are identified with travel times between vertices. More importantly, travel time between vertices depends on a moment of travel start (weights

The research was carried out as part of the research work number WZ/WI/1/19 at the Bialystok University of Technology, financed from a subsidy provided by the Minister of Science and Higher Education.

K. Saeed and J. Dvorský (Eds.): CISIM 2020, LNCS 12133, pp. 402–414, 2020.
https://doi.org/10.1007/978-3-030-47679-3_34

are functions of time). Public transport networks are good examples of time-dependent graphs (travel time determined by time-table). The purpose of the TDOP is to find most profitable path between given two vertices (starting at a given time) limited by total travel time.

The TDTOPTW [7] is an extension of the TDOP. The goal of the TDTOPTW is to find a set of paths (of a given cardinality) maximizing total collected profit. Each path has the same limit of total travel time and the same start/end vertices. Any vertex can be included only once in a multi-path solution. Additionally each vertex has some visit time as well as time window, which determines when a given vertex can be visited. Arriving too late makes it impossible to visit a vertex while arriving too early means waiting for its opening. It should be noted that time of edge traversals as well as time needed to visit vertices and waiting time are all included in total travel time.

Problems from the OP family have many practical applications including tourist trip planning [6,17], transport logistics and even DNA sequencing [3]. In tourist trip planning each attraction (point of interest - POI) has some profit (dependent on its popularity) and a time-window (opening hours). Finding an attractive multi-day tour (of a limited duration) in a time-dependent transport network (time-dependency: timetables and traffic) is equivalent to solving the TDTOPTW.

1.1 Literature Review

Problems from the OP family are NP-hard [9] and exact algorithms can be very time-consuming for larger graphs. For this reason most papers are devoted to metaheuristics. Various approaches for the OP were based i.a. on greedy and randomized construction of solutions [2], local search methods [4,20], tabu search [8], ant-colony optimization [18] and genetic algorithms [19].

Most papers about the Time-Dependent versions of the OP are associated with practical applications of the problem (trip planning in public transport networks). LI [12] published the first article about the classic TDOP and solved it with an exact dynamic programming algorithm. He obtained results for small test instances.

Garcia et al. [6] presented the first paper describing application of the TDTOPTW in POI and public transport network of San Sebastian. To solve the problem the authors proposed Iterated Local Search method (ILS). However, they performed computations on average daily travel times and assumed periodicity of public transport timetables.

Gavalas et al. [7] proposed an approach to the TDTOPTW which uses real time-dependent travel times in a transport network of Athens. The authors introduced two fast heuristics (TD_CSCR and TDSlCSCR), which based on ILS and vertex clustering, and made comparisons of a few methods. The authors created 20 topologies and 100 tourist preferences combining into 2000 different test cases.

Verbeeck et al. [21] developed new benchmark instances for the TDOP, which model street traffic. The authors proposed an ant-colony approach, which achieved high quality results in a short execution time. Gunawan et al. [10] modified Verbeeck's benchmarks (discretization of time) and compared a few approaches (adaptive ILS proved to be the most effective of them).

The author's previous papers were devoted to metaheuristics for various problems from the OP family. Methods developed by the author (composition of evolutionary algorithms and local search heuristics) proved successful on the OP [14,15] as well as on TDOP benchmark instances [16] and on TDOPTW public transport and POI network of Bialystok [17]. The algorithms achieved results close to optimal and outperformed other methods: GRASP [2] and GLS [20] (OP), ACS [21] and Adaptive ILS [10] (TDOP).

Recently a metaheuristic (tabu search+nonlinear programming) was proposed to tackle a new variant of the problem: Orienteering Problem with Service Time-Dependent Profits (OPSTP) [22]. In this variant vertex profits are not constant (as in all previous problems) but change with time (in a non-linear way), which is another aspect of time-dependency.

2 Problem Definition

Let $G = (V, E)$ be a directed, weighted graph. In time-dependent graphs 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, nonnegative visit time τ_i and time windows (opening time to_i and closing time tc_i) indicating its period of availability.

Given the time-dependent graph, moment of start t_0, time limit T_{max}, start and end vertices (s and e) the purpose of Time-Dependent Team Orienteering Problem with Time-Windows (TDTOPTW) is to find a set of m paths from s to e starting at time t_0 which maximizes total profit of visited vertices and total travel time of each path is limited by T_{max}. Each vertex (except s and e) can be included only once and only in one path. For simplicity let's assume that vertices s and e (start and end point of the tour) have no profits, no visit time and no windows (this is usually the case in practical applications).

TDTOPTW can be formulated as Mixed Integer Programming (MIP) problem. Let x_{ij} is 1 iff a solution contains direct travel from i to j and 0 otherwise. Let ta_i and tl_i be arrival and leave time at/from vertex i included in a solution. The purpose of TDOP is to maximize formula 1 while satisfying Eqs. 2–7:

$$max \sum_{i \in V} \sum_{j \in V} (p_i \cdot x_{ij}) \tag{1}$$

$$\sum_{i \in V} x_{si} = \sum_{i \in V} x_{ie} = m \tag{2}$$

$$\underset{i \in V \setminus \{s,e\}}{\forall} (\sum_{j \in V} x_{ij} = \sum_{j \in V} x_{ji} \leq 1) \tag{3}$$

$$tl_s = t_0 \tag{4}$$

$$\underset{i\in V, j\in V\setminus\{e\}}{\forall}(x_{ij}\cdot ta_j = x_{ij}\cdot(tl_i + w_{ij}(tl_i))) \tag{5}$$

$$\underset{i\in V}{\forall}(x_{ie}\cdot(tl_i + w_{ie}(tl_i)) \leq x_{ie}\cdot(t_0 + T_{max})) \tag{6}$$

$$\underset{i\in V\setminus\{s,e\}, j\in V}{\forall}(x_{ij}\cdot ta_i \leq x_{ij}\cdot tc_i \wedge x_{ij}\cdot tl_i = x_{ij}\cdot(max(ta_i, to_i) + \tau_i)) \tag{7}$$

Equation 2 guarantees that every path in a solution starts at vertex s and ends at vertex e. Formula 3 indicates each vertex can be visited at most once and no path ends in vertices other than s and e. Equation 4 guarantees that every path start at time t_0 while formula 5 guarantees that leave/arrival times of subsequent vertices are consistent with time-dependent weights. Constraint 6 means that total travel time of every path cannot be more than T_{max} while constraint 7 guarantees that time-windows are not violated.

3 Algorithm Description

To solve the TDTOPTW the author proposed an evolutionary algorithm with local search methods embedded, which was developed from the method solving the TDOP [16]. It uses both random and local search operators, 2-point heuristic crossover, disturb operator and deterministic crowding as selection mechanism. Path representation was used: subsequent genes indicate vertices visited. The algorithm starts with a random population of feasible solutions.

3.1 Evaluation

Fitness function of a solution is the sum of profits of its paths. Contrary to the author's TDOP algorithm, infeasible solutions are not present in the population. This is due to nature of the problem: additional time-window constraints and presence of multiple paths in a single solution. Genetic operators don't allow solutions to violate time-windows and T_{max} constraints. In the future the author may propose a fitness function for infeasible solutions, which takes into account TDTOPTW specificity.

3.2 Crossover

In each iteration $c_p \cdot P_{size}$ individuals are selected and arranged in random pairs (c_p - crossover probability, P_{size} - population size). The basic procedure of heuristic 2-point crossover (working on two single paths) as well as random parents selection was based on TDOP algorithm. Heuristic crossover tries to exchange fragments between successive common vertices of both paths in order to maximize fitness of the better child.

Each TDTOPTW solution contains m paths (instead of one) and for this reason an adaptation was needed. For each path from parent A the algorithm applies crossover with each path from parent B (m^2 single-path crossovers in total). After each procedure the crossed paths are inserted back into parents and duplicate vertices are removed (if needed). From all options the algorithm chooses the one which maximizes fitness of the better of two modified solutions (children). The procedure is explained in Fig. 1. On the other hand, random crossover version selects sub-paths and exchanged fragments randomly. Crossover specificity is determined by c_h parameter - it's the probability of using heuristic crossover (with $1 - c_h$ the probability of random version).

parent A:
A1: (1, 5, 4, 3, 8, 10, 2, 12, 7)
A2: (1, 6, 9, 14, 16, 11, 7)

parent B:
B1: (1, 17, 16, 3, 14, 5, 10, 17, 7)
B2: (1, 8, 11, 15, 6, 20, 19, 7)

single crossover (A1 and B1):
A1: (**1**, 5, 4, **3**, 8, **10**, 2, 12, **7**)
B1: (**1**, 17, 16, **3**, 14, 5, **10**, 18, **7**)

exemplary result (1...3 fragments exchange):
An: (**1**, 17, $\cancel{16}$, **3**, 8, **10**, 2, 12, **7**)
Bn: (**1**, 5, 4, **3**, 14, $\cancel{5}$, **10**, 18, **7**)

child A:
An: (1, 17, 3, 8, 10, 2, 12, 7)
A2: (1, 6, 9, 14, 16, 11, 7)

child B:
Bn: (1, 5, 4, 3, 14, 10, 18, 7)
B2: (1, 8, 11, 15, 6, 20, 19, 7)

Fig. 1. Exemplary crossover of two solutions (each consists of $m = 2$ paths). In the example first path of parent A (A1) is crossed with first path of parent B (B1). There are three possible fragment exchanges: between 1 and 3, between 3 and 10, between 10 and 7. First of them is presented and new paths (An and Bn) are created. Vertices 5 and 16 are removed from newly created paths due to vertices duplicates in newly created solutions. Afterwards new paths are inserted into original parents and children are formed. Heuristic crossover checks all crossing combinations between various paths and all fragments exchanges.

3.3 Selection

After crossover children compete with their own parents for a place in the population (survivor selection in the form of deterministic crowding [13]). For $m = 1$ edit distance function used to determine pairs is the same as in [16]. For $m > 1$ a modified solution compete with its original version (they have $m - 1$ paths in common and are similar). Such selection approach maintains diverse population for longer enabling a more effective search before convergence.

3.4 Mutation

In each iteration $m_p \cdot P_{size}$ individuals are chosen for mutation (m_p - mutation probability). Mutation include a few operators. Compared to TDOP solution, new operator was added (*move*) and others were modified to operate on multi-path solutions. Initially all m paths of the selected individual undergo 2-opt procedure, which exchanges two edges (connections) present in a path with another two edges in order to reduce total travel time (Fig. 2). Afterwards *move* operator tries to move a single vertex from one path to another (within the chosen solution) in order to reduce the total travel time of the solution as much as possible. All vertices are considered by *move* and if no option reduces the travel time the solution is not modified. This operator helps to balance routes and is presented in Fig. 3. Afterwards a vertex insertion or vertex deletion (heuristic or random) is carried out: (probability of insertion/deletion is 0.5 each). The goal of heuristic insert is to find a non-included vertex maximizing profit to travel time increase ratio: all vertices, all insertion places and all sub-paths are considered. An example of heuristic insert is given in Fig. 4. Heuristic deletion works a bit analogically: it deletes a vertex minimizing profit to travel time decrease ratio. Specificity of operators (random/heuristic) is steered by parameter m_h, which is equal to the probability of usage heuristic insertion/deletion (analogically to c_h in crossover).

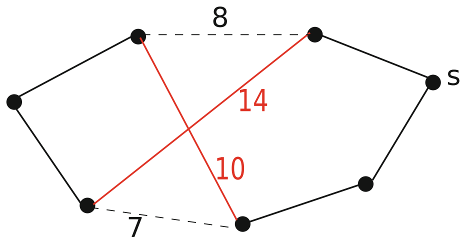

Fig. 2. Exemplary 2-opt operator on Euclidean plane. A solution is a cycle which starts and ends in vertex s. Elimination of edge intersection in the cycle (exchanging red edges for dashed edges) will reduce total length of the cycle by 9. The proposed algorithm operates on time-dependent travel times (not distances) but this example was used for simplicity. (Color figure online)

3.5 Disturb

In each iteration $d_p \cdot P_{size}$ individuals are chosen for disturb (d_p - disturb probability). Disturb is a different kind of mutation. It is executed way less often than standard mutation but it can cause larger changes in individuals. The operator removes a path fragment (consisted of up to 10% of vertices) in a random or heuristic way (it is steered by d_h parameter). The operator is destructive in nature and should be used rarely but it's usage can help escape local optima and slightly improve results.

3.6 Operators Optimization

In time-dependent paths each modification (i.e. insertion of new vertex) requires travel time recalculation for a path fragment after the modification point.

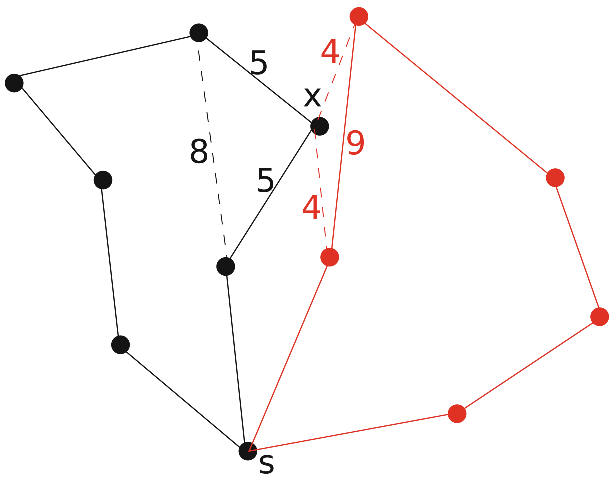

Fig. 3. Exemplary move operator on Euclidean plane. A solution contains two paths (black and red), both start and end in vertex s. Edge distances are marked near edges. It can be seen that moving vertex x from black path to red path results in reduction of total distance of two paths (reduction in black path (2) is larger than increase in red path (1)). The proposed algorithm operates on time-dependent travel times (not distances) but this example was used for simplicity. (Color figure online)

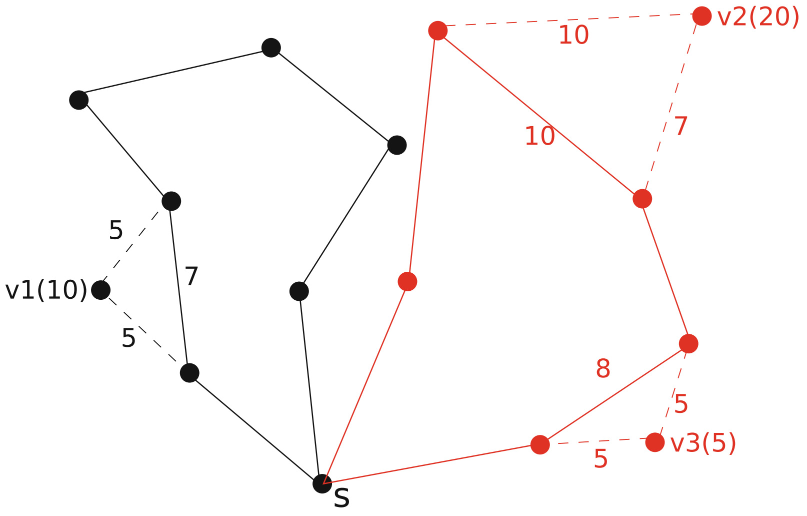

Fig. 4. Exemplary heuristic insertion operator on Euclidean plane. The presented solution consists of two paths (black and red) starting and ending in s. Vertex profits are in parenthesis and edge costs are marked near edges. The goal is to find new vertex and insertion point (in any path), which maximize ratio of profit to cost increase. For $v1$ the ratio is 10/3, for $v2$ it's 20/7 and for $v3$ it's 5/2. Vertex $v1$ will be included in the black path by the operator. The proposed algorithm operates on time-dependent travel times (not distances) but this example was used for simplicity. (Color figure online)

In case of insert operator (which checks all possible insertion places for a new vertex) naive searching would require n^2 time complexity (n - path size). However, determining best insertion place (in terms of travel time increase) for a given vertex can be done in linear time with just one loop over a path. Let's assume that the path includes vertices $1, 2, ..., n$ and let $t_1, t_2, ..., t_n$ be vertex arrival times for this path. We want to insert new vertex x in the best place (minimizing total travel time after insertion). This problem is equivalent of finding earliest arrival time to vertex n (assuming that x was included). This can be calculated recursively. Let $EAT(k)$ be earliest arrival time to vertex k assuming that vertex x was included in a path somewhere before k. Here are formulas (for their simplicity there are no time-windows and visit times):

$$EAT(2) = t_1 + w_{1x}(t_1) + w_{x2}(t_1 + w_{1x}(t_1)) \tag{8}$$

$$EAT(k) = MIN \left\{ \begin{matrix} EAT(k-1) + w_{k-1k}(EAT(k-1)) \\ t_{k-1} + w_{k-1x}(t_{k-1}) + w_{xk}(t_{k-1} + w_{k-1x}(t_{k-1})) \end{matrix} \right\} \tag{9}$$

The optimal solution of insertion x before vertex k is either:

1) inserting x optimally before vertex $k-1$ and connect $k-1$ and k directly

2) inserting x directly before vertex k (and after $k-1$)

Out of these two options we choose one with earliest arrival time at vertex k. Calculating it in one loop (from vertex 2 to n) we can obtain $EAK(n)$ and optimal insertion point in linear execution time.

Additional optimizations were also performed on time-consuming 2-opt operator. Precomputation of reversed path fragments is done to estimate fast if a given 2-opt move is promising to perform. Details were presented in [16].

4 Experimental Results

Experiments were conducted on a computer with Intel Core i7 3.5 GHz processor and the algorithm was implemented in C++. The algorithm was tested on public transport and POI network of Athens (tests prepared by Gavalas et al. [7]). The authors created 20 topologies and 100 tourist preferences combining into 2000 different test cases. Results presented in this section (profits and execution times) are averaged over execution runs for all test cases. Gaps are expressed in percent and illustrate relative differences between profits of EVO100 and other methods. The author's metaheuristic is compared to:

- Time-dependent heuristics (TD_CSCR, TDSlCSCR) by Gavalas et al. [7] and their version working on average travel times (AvgCSCR).
- ILS algorithm working on average travel times (AvgILS) by Garcia et al. [6] and its time-dependent version (TD_ILS).
- Exact algorithm (based on branch-and-bound and dynamic programming techniques) implemented by the author (marked as OPT).

Two versions of the evolutionary algorithm (with different population sizes) were tested: EVO100 and EVO20. Parameter values were derived from the TDOP algorithm [16]. Originally parameter values were computed by automatic tuning procedure - ParamILS [11,14]. Parameters N_g and C_g for EVO20 were scaled down to 1000 and 100 accordingly. Version with reduced population size (20) was chosen because its execution times were similar to other compared methods. Parameters are given in Table 1.

Table 1. Parameters of the evolutionary algorithm

Param.	Value	Description	Param.	Value	Description
P_{size}	100/20	Population size	m_p	1	Mutation probab.
N_g	5000/1000	Max. generations number	c_p	1	Crossover probab.
C_g	500/100	Max. generations number	d_p	0.01	Disturb probab.
		Without improvement	m_h	1	Mutation heuristic coeff.
d_h	0.8	Disturb heuristic coeff	c_h	0.8	Crossover heuristic coeff

Table 2. Experimental results for different numbers of paths (m). Gaps are expressed in percent and illustrate relative differences between profits of EVO100 and other methods. Execution times are given in seconds. Max. trips duration: 5 hours, start at 10:00.

Method	m = 1		m = 2		m = 3		m = 4		Exec.
	Profit	Gap	Profit	Gap	Profit	Gap	Profit	Gap	Times
AvgILS	298.5	13.4	561.9	16.0	819.7	13.1	1078.7	14.5	0.02–0.26*
TD_ILS	326.3	5.3	641.4	4.1	939.8	3.4	1219.2	3.3	0.02–0.38*
AvgCSCR	332.0	3.6	643.3	3.9	933.7	4.1	1209.3	4.1	0.03–0.2*
TDSlCSCR	342.1	0.7	657.9	1.7	946.5	2.8	1219.1	3.3	0.05–0.32*
TD_CSCR	337.8	1.9	654.3	2.2	948.1	2.6	1225.5	2.8	0.04–0.26*
EVO20	343.6	0.3	666.5	0.4	966.3	0.7	1248.9	1.0	0.04–0.36
EVO100	344.5	0.0	669.2	0.0	973.3	0.0	1261.1	0.0	0.66–7.6
OPT	344.6	-0.1	−0.1/−0.4**						

*Other methods were executed on a different computer - times given for informative purposes.
**Because of long execution times optimal solutions for $m = 2$ were computed only for preference number 205 (and all 20 topologies). Given gaps are differences between OPT and both EVO versions.

In Table 2 experimental results for all methods are given. One can see that the proposed evolutionary algorithm (both versions) clearly outperforms all other methods. On average EVO20 is 1.5 and 1.8% better than the best of remaining metaheuristics (TDSlCSCR and TD_CSCR) and the difference is about 2% for $m > 2$. Gaps of other methods are even larger (3–16%). What is more, EVO20 achieves results close to optimal (average gap of 0.4% for $m \leq 2$) in a very short execution time. Differences between the best algorithms gradually grow as m is increased. Larger solution space (larger m) is explored more effectively by the stronger version of the evolutionary algorithm (EVO100) and gaps between these two versions rise from 0.3 to 1.0%. Better results by EVO100 (nearly optimal) are achieved at the cost of increased execution time.

Table 3. Additional results (profits) obtained by EVO for longer trips (maximum duration: 8 h, start time: 9:00).

Method	m = 1	m = 2	m = 3	m = 4	Exec. times
EVO20	560.2	1069.4	1529.9	1957.6	0.07–0.7
EVO100	561.4	1074.8	1542.7	1976.2	0.9–16
OPT	561.5				

In Table 3 results for longer trips are presented. For $m = 1$ both algorithm versions achieve results very close to exact algorithm. As m gets larger EVO100 gains advantage over EVO20 (up to 1%). No optimal solutions are known for $m > 1$ but these results are presented for future comparisons.

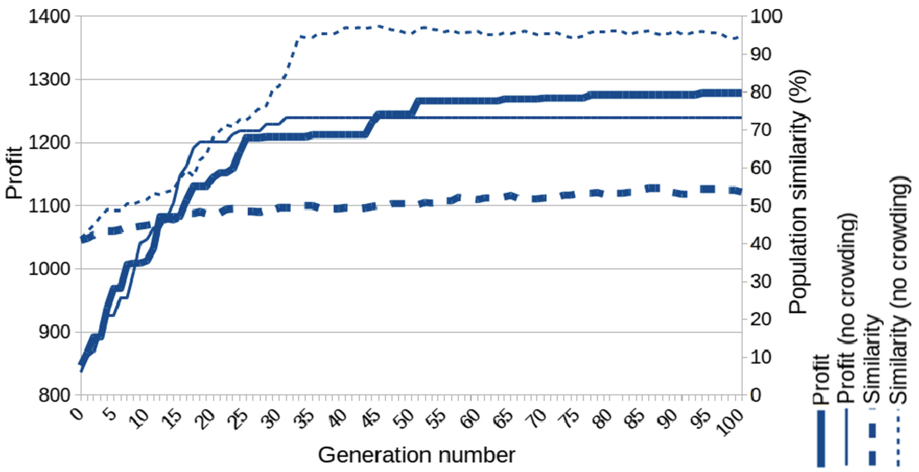

Fig. 5. Exemplary runs of two algorithm versions: with and without crowding. Profit of the best solution found so far and average population similarity are presented as a function of generation number. $P_{size} = 20$, $m = 4$, $topology = 1$, $preference = 408$.

In Fig. 5 there is a comparison of two algorithm runs. One of them uses deterministic crowding and the other uses random assignment of competition pairs during survivor selection. Population similarity (based on longest common subsequence metric) grows very fast without deterministic crowding - similarity close to 100% signals convergence around one solution. Deterministic crowding forces competition between more similar individuals, which enables to preserve population diversity for longer and improve results in later generations (as seen in the figure). Usage of crowding improved average results by 1.4% for $m = 4$. This method of selection proved to be very effective (compared to standard parent selection methods) in the classic OP as well [14].

Table 4. Comparison of average results for different values of parameters: heuristic crossover coefficient (c_h) and heuristic mutation coefficient (m_h). Relative profit losses (in percent) to the best configuration (in bold) are given. Popul. size was 20 and $m = 4$.

$\mathbf{c_h}$	$\mathbf{m_h}$					
	0.0	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
0.0	23.8	11.0	3.5	1.4	0.8	0.8
0.2	19.9	8.2	2.4	0.9	0.5	0.5
0.4	16.3	6.4	2.0	0.7	0.3	0.3
0.6	13.6	5.4	1.6	0.5	0.2	0.2
0.8	11.8	4.7	1.3	0.4	0.1	0.1
1.0	10.4	4.1	1.2	0.3	**0.0**	0.1

In Table 4 the algorithm performance for different values of parameters is presented. Heuristic crossover coefficient (c_h) is the probability that a given crossover will be heuristic (the probability of a random version of crossover is $1 - c_h$). Parameter m_h plays analogical role for mutation. It can be seen that usage of heuristic crossover and local search during mutation has a very good influence on results quality. The best profits are achieved for high values of m_h and medium/high values of c_h. The algorithm isn't very sensitive to changing parameter values: almost half of parameter configurations in the table is less than 1% worse than the best configuration.

5 Conclusions and Further Research

In this paper an effective algorithm solving the Time-Dependent Team Orienteering Problem with Time Windows was presented. The described method (evolutionary algorithm with local search heuristics) proved to be very effective compared to other metaheuristics. It was confirmed that the presented approach is efficient for various problems from the OP family. Test were conducted on public transport and POI network of Athens and high-quality solutions were achieved in a very short execution time. This signals potential application of the algorithm in e-tourism. Further research will concentrate on adaptation of the method to related problems: the Orienteering Problem with Time-Dependent Profits [22], the Orienteering Problem with Hotel Selection [5] and the Stochastic Orienteering Problem [1].

References

1. Campbell, A.M., Gendreau, M., Barrett, W.T.: The orienteering problem with stochastic travel and service times. Ann. Oper. Res. **186**(1), 61–81 (2011)
2. 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)
3. Caserta, M., Voss, S.: A hybrid algorithm for the DNA sequencing problem. Discrete Appl. Math. **163**(1), 87–99 (2014)
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. Divsalar, A., Sorensen, K., Vansteenwegen, P., Cattrysse, D.: A memetic algorithm for the orienteering problem with hotel selection. Eur. J. Oper. Res. **237**(1), 29–49 (2014)
6. 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)
7. 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)
8. 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)
9. Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Res. Logist. **34**, 307–318 (1987)
10. Gunawan, A., Yuan, Z., Lau, H.C.: A Mathematical Model and Metaheuristics for Time Dependent Orienteering Problem. Angewandte Mathematik und Optimierung Schriftenreihe AMOS 14 (2014)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stutzle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
12. Li, J.: Model and algorithm for Time-Dependent Team Orienteering Problem. Commun. Comput. Inf. Sci. **175**, 1–7 (2011)
13. Mahfoud, S.W.: Crowding and preselection revisited. In: Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN II), Brussels, Belgium, pp. 27–36. Elsevier, Amsterdam (1992)
14. Ostrowski, K.: Parameters tuning of evolutionary algorithm for the orienteering problem. Adv. Comput. Sci. Res. **12**, 53–78 (2015)
15. Ostrowski, K., Karbowska-Chilinska, J., Koszelew, J., Zabielski, P.: Evolution-inspired local improvement algorithm solving orienteering problem. Ann. Oper. Res. **253**(1), 519–543 (2017)
16. Ostrowski, K.: Evolutionary algorithm for the Time-Dependent Orienteering Problem. Lect. Notes Comput. Sci. **10244**, 50–62 (2017)
17. Ostrowski, K.: An effective metaheuristic for tourist trip planning in public transport networks. Appl. Comput. Sci. **12**(2) (2018)
18. Schilde, M., Doerner, K., Hartl, R., Kiechle, G.: Metaheuristics for the biobjective orienteering problem. Swarm Intell. **3**, 179–201 (2009)
19. Tasgetiren, M.: A genetic algorithm with an adaptive penalty function for the orienteering problem. J. Econ. Soc. Res. **4**(2), 1–26 (2001)
20. 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)

21. Verbeeck, C., Sorensen, K., Aghezzaf, E.H., Vansteenwegena, P.: A fast solution method for the time-dependent orienteering problem. Eur. J. Oper. Res. **236**(2), 419–432 (2014)
22. Yu, Q., Fang, K., Zhu, N., Ma, S.: A matheuristic approach to the orienteering problem with service time dependent profits. Eur. J. Oper. Res. **273**(2), 488–503 (2019)