A Composite Algorithm for the Team Orienteering Problem with Time Windows

Charalampos Konstantopoulos
Department of Informatics,
University of Piraeus,
Greece
Email: konstant@unipi.gr

Dimitrios Orfanos
Department of Informatics,
University of Piraeus,
Greece
Email: dorfanos85@gmail.com

***Abstract*—In this paper we propose a new heuristic algorithm for the Team Orienteering Problem with Time Windows (TOPTW). This problem is applicable to several real life problems. The route planning of the vehicles which belong to a company or the tourist trip design problem, are some of them. In the first problem, each customer is associated with a profit which can be obtained after visiting them. The objective is to derive a constant number of routes visiting customers such that the profit obtained overall is the maximum possible given that some constraints are satisfied. Specifically, the total cost of transportation of each route must be less than or equal to a certain time budget. In addition, the visit to each customer must occur within a specific time interval. This time interval is termed time window and can be different for each customer. In this sort of problems, the budget may represent the available time for serving customers and the number of routes represents the number of vehicles. In the second problem, the tourist trip design problem, there are some Points of Interest (POIs) which represent tourist sites. Each site is also associated with a profit indicating the interest of a tourist in this attraction. Tourists are looking for a constant number of routes (itineraries) so as to maximize the total profit gained out of the visit of sites included in these itineraries. The time budget and time window constraints are also exist. TOPTW is a generalization of the Orienteering problem. Hence, TOPTW belongs to NP-hard problems. The Iterated Local Search and Average Slack are heuristics that have been proposed in the literature for TOPTW. The proposed heuristic combines the aforementioned heuristics and takes advantage of the best features of both algorithms for providing solutions of higher overall profit.**

Keywords: Team Orienteering Problem with Time Windows, Metaheuristics, Vehicle Routing Problem, Tourist trip design Problem.

I. INTRODUCTION

Team Orienteering Problem with Time Windows (TOPTW) applies to real life problems like tourist guide application and vehicle routing optimization. In tourist guide application, tourists wish to visit the tourist sites of a city but they cannot visit all of them because the available time is limited. The input graph represents the city with tourist sites, Points of Interest (POIs), the number of tours represents the number of available days and time windows correspond to the opening and closing time of each site. Each site is associated with a profit which may express its significance or the level of interest of a specific tourist for this. In the special case where only one tour should be returned, the problem is known as Orienteering problem with time windows (OPTW).

In the problem of vehicle routing optimization there is a specific number of vehicles which visit some locations of a city. For example, the vehicles may belong to a company or organization and the locations may represent customers. The vehicles cannot visit all customers because of limited time. Consequently the company seeks to find a number of routes equal to the number of vehicles in order to visit as many customers as possible. Furthermore each customer may be related to a significance value, called profit, which is analogous to the amount of customer's order. In that case, the main objective is to visit those customers so that the overall profit is the maximum possible. Also the route planning must take into account some constraints which are the time budget for each vehicle and a given time interval for each node which models the time in which the visit must occur in order to be acceptable.

It has been proved that Orienteering problem without Time Windows belongs to NP-hard problems [5]. This means that TOPTW is also NP-hard due to the extra constraints. Thus, it is impossible that an exact solution will be found in polynomial time. Therefore, many heuristic algorithms have been proposed to solve the problem approximately. Some of them are ILS by Vansteenwegen et al.[1], Average Slack based on the SlackCSCRoutes in [3], GRASP-ELS by Labadie et al. [6], ACS algorithm by Montemanni et al. [7] and I3CH by Hu et al. [8].

GRASP_ELS heuristic by Labadie et al. [6] combines the greedy randomized adaptive search procedure (GRASP) with evolutionary local search (ELS) technique. In general, the algorithm applies GRASP to construct an initial solution and then inputs this solution to ELS. ELS improves the solution with iterated local search (ILS) which combines insertion steps with a perturbation step. ACS heuristic is based on ant colony optimization technique. GRASP-ELS, I3CH and ACS heuristics give high quality solutions but their computational time is large.

The algorithm of TOPTW heuristic is likely to be applied to online applications which can be used by tourists in order to find the tours that they are going to visit. For this reason, it is important that the algorithm will run in real time.

The Iterated Local Search (ILS) algorithm by Vansteenwegen et al. [1] is a meta-heuristic, based on iterated local search technique. ILS is the most effective algorithm for TOPTW so far because it returns high profitable solutions in short time. However the disadvantage of ILS is that it can be trapped in local optimum solutions without taking into consideration some high profitable areas of the search space.

Average Slack is also a heuristic algorithm for the TOPTW. Average Slack is based on SlackCSCRoutes algorithm for Time-Dependent instances [3]. Average Slack is a modified version of SlackCSCRoutes in order to find solutions to instances where the cost of transportation from one node to another is constant and it is not dependent on the departure time.

In this paper we propose a new heuristic algorithm for the TOPTW which combines ILS with Average Slack. We compare our algorithm with ILS and Average Slack as regards the total profit collected by the solution and computation time.

II. Problem Definition

To define the Team Orienteering Problem with Time Windows, a complete graph $G=(V,E)$ is given where $V=\{v_0,v_1,...,v_{n+1}\}$ is the set of vertices (or nodes) and E is the set of edges. Vertices v_0, v_{n+1} correspond to the start and terminal node, respectively. We focus on problems in which v_0 and v_{n+1} are identical. Each node v_i has a non-negative profit p_i. The profit of v_0 ($\equiv v_{n+1}$) is zero. Every edge of the graph (v_i,v_j) is related with a positive number c_{ij} which is the cost of transportation from v_i to v_j. In addition, each node v_i is associated with a time interval $[R_i,D_i]$ called time window and the visit at node v_i must occur within its time window. Let *m* be a positive integer which corresponds to the number of routes and *B* be the budget of each route, that is the upper limit of the total cost of each route. Due to the fact that v_0 and v_{n+1} are identical, each route is actually a tour. The solution of the problem consists of *m* disjoint tours starting and ending at the same predefined vertex, so as to maximize the total profit collected by visited vertices in all tours. Each tour must have cost (or length) less than or equal to the budget *B* without violating time windows constraints.

III. Iterated Local Search

Iterated Local Search (ILS) algorithm combines an insertion step and a shake step. ILS starts with an initial solution and applies the insertion step until a local optimum solution is found. That means that there is not any other feasible insertion to the solution. Then, ILS applies the shake step where a number of consecutive nodes is removed from each tour of the solution. The insertion step is executed again on the derived solution after the shake step, until a new local optimum solution is reached. The new solution is compared with the previous one and the algorithm stores the solution with the highest profit. This computation ends when a fixed number of consecutive iterations without solution improvement have been executed.

A. Insertion Step

At this step, a new node is inserted into the solution provided that this insertion does not violate the time windows of the nodes which follow in the tour. In order to effectively estimate whether an insertion is feasible, some extra information is associated with each node belonging to the solution, namely the *Wait* and *MaxShift* parameters. *Wait* represents the waiting time in case the arrival at a node occurs before its time window. *Wait* is zero if the arrival takes place after the opening time of the time window.

Pseudo-code of Insertion Step

For each node not included in the solution
  Find the feasible position for insertion having the smallest shift
  Calculate ratio($=\frac{p^2}{shift}$)
End For
Insert node with the highest ratio (say v_j) to the position with the smallest shift
Calculate $wait_j$, $arrive_j$ and $start_j$
For each node v_i in tour after v_j
  Update $wait_i$, $arrive_i$, $start_i$ and $MaxShift_i$
End For
Update $MaxShift_j$
For each node v_i in tour before v_j
  Update $MaxShift_i$
End For

Let $[R_i, D_i]$ denote the time window of node v_i and $arrive_i$ denote the arrival time at this node. Then

$$Wait_i = \max(0, R_i - arrive_i)$$

MaxShift is defined as the maximum time that the start of the visit can be prolonged without violating time constraints of the tour. *MaxShift* is defined recursively as follows: if v_{n+1} is the last node of the tour, then $MaxShift_{n+1} = B - arrive_{n+1}$, where B is the time budget of the tour. For each node v_i, $MaxShift_i = \min(D_i - s_i, Wait_{i+1} + MaxShift_{i+1})$ where s_i denotes the starting time of the visit at node v_i.

Every insertion in the tour yields an increment in the tour time. Assume now that we are about to insert node v_j between nodes v_i and v_k in a tour. $Shift_j$ represents the tour time increment due to insertion of v_j and is defined as $Shift_j = c_{ij} + Wait_j + u_j + c_{jk} - c_{ik}$ where u_j is the visiting time at node v_j. Also, the feasibility of the insertion of node v_j between nodes v_i and v_k in a tour can be checked by the following formula: $Shift_j \le Wait_k + MaxShift_k$.

It also has to be ensured that the visit at node v_j takes place within its time window $[R_j, D_j]$. Hence, the inequality $s_i + u_i + c_{ij} \le D_j$ must be valid.

Then, for every node v_j that does not belong to the solution and for all the possible positions, the position with the smallest shift ($shift_j$) is determined and next the $ratio_j = \frac{p_j^2}{shift_j}$ is computed. The node with the highest ratio is inserted to the corresponding position. After the insertion, the parameters of each node in the tour must be updated. For the nodes following the insertion, *Wait*, *arrive*, *Start* and *MaxShift* have to be updated. For the nodes before the insertion only *MaxShift* needs update.

B. Shake Step

In order to avoid being trapped in a local optimal solution, shake step is applied. At this step one or more nodes are removed from each tour of the solution. Every shake step takes two integers as input. The first input integer (R) represents the number of consecutive nodes which are going to be removed. The second integer (S) represents the position in the tour from where the removal will start. If the removal meets the last node of the tour, then it continues from the start of the tour.

IV. The Average slack algorithm

The Average Slack algorithm also combines an insertion step with a shake step. The key difference between Average Slack and ILS is the different insertion criterion. Specifically, Average Slack uses a global criterion at the insertion step. This criterion takes into account the effect of a candidate insertion into a tour, on all the nodes belonging to the tour. For each node v_i of a tour, we define the variable $slack_i$ as $slack_i = MaxStart_i - arrive_i$, where $MaxStart_i$ denotes the maximum time that the visit at v_i can be delayed so as not to violate time windows constraints for nodes after v_i in the tour. As can be seen from definitions of *MaxStart* and *MaxShift* the following equality must holds:

$$MaxStart_i = s_i + MaxShift_i$$

MaxStart is defined recursively as follows:

$$MaxStart_i = \min\{D_i, MaxStart_{next(i)} - c_{i,next(i)} - u_i\}$$

where $next(i)$ is the first node after v_i in the tour and u_i the visiting time of node v_i. For the last node in the tour, it holds $MaxStart_{n+1} = D_{n+1}$. Obviously, as $slack_i$ goes closer to zero, it is more difficult to find a feasible insertion between v_i and its previous node in the tour.

Now assume that v_k is a candidate node for insertion between consecutive nodes v_i and v_{i+1} in a tour of the solution. The insertion of v_k will probably increase the arrival time at some nodes following v_k in the tour, because the cost of transportation to v_k and the visiting time u_k should be added to the total cost of the tour. For nodes before the insertion, *MaxStart* will also change. As a result, the slack variable will change for all the nodes in the tour.

Thus, the Average Slack of a tour after the insertion of a candidate node v_k between the nodes v_i and v_{i+1} in the tour is given by the following formula:

$$AvSlack_k^i = \frac{\sum_{j=1}^{i} slack_j^k + \sum_{j=i+1}^{n} slack_j^k + slack_k}{n+1},$$

where $slack_j^k$ for $j = 1,\ldots,n$ are the new slacks after insertion of v_k between v_i and v_{i+1}.

If $AvSlack_k^i$ has a large value, this means that after the insertion of node v_k, it is highly probable that further insertions before or after v_k in the tour are still feasible.

Overall, at each insertion step, for each node v_k not belonging to the solution and for each feasible position in the solution, the Average Slack is computed. Let $\max AvSlack_k$ be the largest Average Slack value for a node v_k among all feasible positions. In order to decide which node will be inserted at the current insertion step, the weight $slackWeight_k = p_k^2 \cdot \max AvSlack_k$ is calculated for each node v_k. Finally the node with the largest weight is inserted into the position corresponding to the $\max AvSlack_k$ value. Shake step is applied in the same way as in ILS heuristic.

V. The composite algorithm

In this section, we propose a new heuristic algorithm for TOPTW and we compare it with ILS and Average Slack, in order to test its efficiency. The algorithm executes a number of insertion and shake steps but now at each insertion step it combines the logic of the ILS and the Average Slack heuristic and returns good approximate solutions for the TOPTW.

The basic idea is that for tours incurring large waiting time at most of their nodes, Average Slack obtains good solutions, while for tours with small waiting time at nodes, ILS is more effective. If a tour is associated with large waiting time, this means that there exist large slacks in several segments of the tour. Hence Average Slack performs well. When the waiting time at nodes is short, slacks are accordingly small and Average Slack is not able to insert many new nodes. In this case, ILS performs better because of its different insertion criterion which is not global as that of Average Slack. In the proposed algorithm, Average Slack is applied first and derives solutions with small total waiting time at the tours. Then, ILS is executed for further improving the solution since ILS is more "aggressive" in finding areas with high node density when the waiting time at each node is relatively small.

The algorithm takes a threshold value as an input which defines how large the waiting time of the tour should be in order to switch from one algorithm to the other. The threshold variable assumes values between 0 and 1. When the threshold value is 1, we essentially get the ILS and when that value is equal to zero, the Average Slack algorithm is obtained.

A. *Insertion Step of the Composite Algorithm*

We separate the *m* tours into two sets, namely R1 and R2. R1 contains tours with low total sum of the waiting times at their nodes and R2 contains the remaining tours. The limit of total waiting time per tour which defines the separation into the sets R1 and R2 is determined by the threshold value, mentioned above. In particular, for each tour, we calculate the ratio of total waiting time in the tour to the maximum time limit (budget). If the aforementioned ratio is less than or equal to threshold, the tour belongs to the set R1, otherwise, to the set R2. It is worth mentioning that the total waiting time of a tour includes also the difference of tour's budget minus the total time (length) of the tour. After the separation of tours, the insertion step of ILS is applied to tours belonging to R1, and the insertion step of Average Slack is executed on the tours belonging to R2. Hence, for each tour in R1 the algorithm searches for a node v_i not belonging to the current solution and possessing the largest value in the ILS criterion, i.e. the $ratio_i = \frac{p_i^2}{shift_i}$ of node v_i is the largest one.

Now for each tour in R2, the algorithm searches for the node v_j with the highest value in the Average Slack criterion, as described in section IV. If R1 is empty, node v_j will be inserted into the solution in the tour and the specific position indicated by the Average Slack.

Pseudo-code of Insertion Step for the Composite Algorithm

tb: maximum time budget per tour
twt_i : total waiting time in tour i (*)
th: a real number $\in [0,1]$
R1: the set of tours r_i such that $\frac{twt_{r_i}}{tb} \le th$
R2: the set of tours not belong to R1
bestRatio←0
bestSlackWeight←0
For each node $\notin$ the solution Do
  For each tour $r \in$ R1 Do
    Find the position with the smallest shift
    Calculate ratio= $p^2 / shift$
    If ratio>bestRatio then
      bestRatio←ratio
      Let v_i and r_i be the node and the tour respectively that correspond to bestRatio
    End If
  End For
  For each tour $r \in$ R2 Do
    Find the maximum average of slacks (*AvgSl*)
    Calculate slackWeight= $p^2 \cdot AvgSl$
    If slackWeight>bestSlackWeight then
      bestSlackWeight←slackWeight
      Let v_j and r_j be the node and the tour respectively that correspond to bestSlackWeight
    End If
  End For
End For
If R2 is empty then
  Insert v_i in tour r_i
Else If R1 is empty
  Insert v_j in tour r_j
Else
  Insert the node with the highest profit between v_i and v_j
End If
(* twt = the total waiting time of nodes in a tour plus the difference between budget and current duration of the tour)

When R2 is empty, node v_i from ILS will be inserted into the solution in the previously determined tour and position. If both R1 and R2 are not empty, the algorithm inserts the node with the highest profit between v_i and v_j. The insertion step is applied until the solution reaches a local optimum which means that no further feasible node insertions are possible in the current solution.

In order to reduce the execution time of the algorithm, we have also applied an acceleration technique for calculating the average of slacks and the total waiting time of a tour. The idea is to store the average slack and the total waiting time of a tour after each insertion and calculate them incrementally in the next insertion step.

B. *Shake Step of Composite Algorithm*

When the solution reaches a local optimum, the shake step as described in section III.B, is executed. Finally, all the steps of the Composite Algorithm are given in detail in the following pseudo-code:

Pseudo-code of the Composite Algorithm

bestSol←null
bestProf←0
Initialization of currentSol(m tours starting and ending at the same node)
maxNumberToRemove←NumberOfPOI/(3*m)
notImproved←0
S←1, R←1
While notImproved< 150 Do
  While currentSol not local optimum Do
    Insertion Step
  End While
  currentProf←Profit of currentSol
  If currentProf>bestProf then
    bestSol←currentSol
    bestProf←currentProf
    R←1
    notImproved←0
  Else
    notImproved←notImproved+1
  End If
  Shake(R, S)
  S←S+R
  R←R+1
  smallestSize←the length of the tour with the least number of nodes
  If S ≥ smallestSize then
    S←S-smallestSize
  End If
  If R is equal to maxNumberToRemove then
    R←1
  End If
End While
Return bestSol

VI. EXPERIMENTAL RESULTS

The evaluation of the composite algorithm is done using some published instances which are useful to test the algorithm in theoretical and practical level. The first group of instances are Solomon's (c*,r*,rc*) and instances of Cordeau (pr*). The instances of Solomon include 100 POIs. The c1*, r1*, rc1* instances have smaller budget and tighter time windows than c2*, r2*, rc2* instances. Cordeau's instances pr* include 44-288 nodes. The time budget is 1000 minutes. The instances pr01-pr10 have 135 minutes average length of time windows and pr11-pr20 269 minutes. Solomon's and Cordeau's instances are tested for number of tours *m*=1, 2, 3, 4.

A main disadvantage of these instances is that they do not correspond to realistic conditions as regards the tourist trip design problem. In a realistic setting, POIs have large time windows which may be different for each visiting day. The isolated POIs are few and usually POIs are close to each other in specific areas. Also, in practice, the profit of each POI is correlated with the visiting time at the POI. In addition, the time budget is commonly about few hours per visiting day in contrast to the aforementioned instances which have large budget.

For these reasons, 100 instances (t*) have been constructed. Each instance includes the number of tours of the problem (*m*=1, 2 or 3). The number of POIs varies between 100 and 200, 80% of POIs are located in one virtual area and the number of virtual areas is in the range of [1,..., 10]. The profit of each point depends on the visiting time duration. Finally, the time budget is 5 hours for t2* instances and about 10 hours for t1* instances.

The algorithm was implemented with C++ and executed on Intel(R) Xeon(R) 2.5GHz, 16GB RAM. We evaluate our algorithm as regards the profit of the obtained solutions and the execution time measured in milliseconds. Ideally, the objective is to find a solution of maximum profit as fast as possible. For this reason, we compare the solutions obtained by the composite algorithm with those of ILS and Average Slack heuristics. For each instance, the composite algorithm is applied for all the values of the threshold parameter in the range [0,1]. Specifically, the threshold value is gradually increased in steps of 0.01 starting from 0 until it reaches 1. As has already been mentioned threshold=0 corresponds to Average Slack algorithm and threshold=1 to ILS.

A. Profit Evaluation

In Table I, we present the average profit improvement of the composite algorithm relative to ILS. For each instance, the best profitable solution corresponding to a threshold value between 0.01 and 0.99 is compared with the profit of the ILS solution.

It is clear that for an appropriate threshold value, the composite algorithm outperforms ILS with respect to the profit. The disadvantage of ILS is that even when the waiting time at nodes is relatively large, sometimes it does not explore all the search space because either it is trapped at remote nodes with high profit or it does not explore remote areas with high density of low-profit nodes due to the large cost of transportation and the small profit of these nodes. This is more obvious in instances with high number of POIs or with tight time windows and low time budget.

TABLE I. AVERAGE PERCENT OF IMPROVEMENT

Composite vs ILS profit improvement (%)				
	Number Of Tours			
Topology Name	1	2	3	4
c1	0.6	1.17	1.12	1.13
c2	1.64	1.13	1.5	0
r1	1.62	1.11	1.43	1.48
r2	2.61	2.08	0.13	0
rc1	3.45	3.15	2.08	2.06
rc2	2.04	3.26	0.81	0
Pr	4.29	5.26	4.39	4.35
t1	2.98			
t2	4.22			

On the other hand, Average Slack has a more general criterion than that of ILS. ILS takes into account the node profits and the shift of a potential insertion. Since the shift variable is related to the cost of transportation, if the transportation cost is high, the shift is also high and the value of the ILS insertion criterion is decreased accordingly. Average Slack is seeking for nodes that give large average of slacks in the tours affected by their potential insertion. Aside from the profit, Average Slack takes implicitly into account the shift, in order to avoid overly increasing the arrival times for the nodes located after the insertion point along the affected tour and in order to keep almost the same *MaxStart* for nodes being before the insertion point. Therefore, Average Slack algorithm has a global insertion criterion and considers the effect of a candidate insertion on the whole tour. Thus, at the first steps of the Composite algorithm, where the average slack of the so far constructed tours is relatively large, the insertion of nodes belonging to remote areas with high density of nodes is likely and thus the execution of the Average Slack in this setting is able to insert a lot of new nodes in the existing tours. As a result, the overall profit obtained by visiting these areas is high due to the large number of POIs inserted into the solution.

Table I shows that for t2* instances, composite algorithm gives solutions with high profit compared to ILS (4.22% more profitable on average). Those instances have low budget compared to other instances and high number of POIs. t1* instances have also high number of POIs which explains the high average improvement (2.98%). Compared to t2* instances, t1* have higher budget. The fact that the composite algorithm gives highly profitable solutions for t* instances is important because these instances correspond to realistic conditions in tourist trip design problem. For the instances of Cordeau, the average improvement is also high, because these instances include a high number of POIs and their time windows are small in comparison to the c1* instances.

TABLE II. AVERAGE PERCENT OF IMPROVEMENT

Composite vs AvgSlack profit improvement (%)				
	Number Of Tours			
Topology Name	1	2	3	4
c1	1.97	2.29	1.98	2.68
c2	0.98	1.65	2.5	0
r1	2.43	2.51	2.63	3.2
r2	3.16	1.82	0.2	0
rc1	2.58	2.28	2.41	2.51
rc2	4.69	2,72	0.58	0
pr	6.52	3,62	2.33	2.62
t1	2.14			
t2	2.77			

Rc1* instances have low time budget per tour and tight time windows. In c2*, r2* and rc2* instances, the average improvement is small for the case of 3 and 4 tours because both algorithms (composite and ILS) find the best solution which includes all POIs.

Composite algorithm has also been compared with Average Slack. Table II presents the average percent of profit improvement of the Composite algorithm against the Average Slack. Clearly, the Composite algorithm gives better solutions than Average Slack as regards profit for an appropriate threshold value.

B. Time Evaluation

It is important that the algorithm can be executed in real time in order to be suitable for online tourist applications. An important feature of ILS is that it runs in real time. Now, we compare our algorithm with ILS as regards the execution time and the results are listed in Table III. For each group of instances, the average difference in the execution time is computed in milliseconds. When the average difference is positive, the Composite algorithm is faster than ILS.

ILS is faster for most groups of instances. However, the average gap in the execution time of the two algorithms is at most 1.6 sec in the worst case. This means that the Composite algorithm can also be executed in real time.

Finally in Table IV, the composite algorithm is compared to the Average Slack. For most of the instances, the Composite algorithm exhibits shorter execution time on average in comparison to the Average Slack.

VII. CONCLUSIONS

In this paper we presented a new heuristic algorithm for the Team Orienteering Problem with Time Windows. The algorithm combines the ILS algorithm by Vansteenwegen et al. [1] and a modified version of SlackCSCRoutes algorithm in [3] called Average Slack in this paper. The experimental results demonstrate that the composite algorithm returns solutions with higher profit than those returned by the two other algorithms.

TABLE III. TIME GAP (MS) COMPOSITE VS ILS

	Number Of Tours			
Topology Name	1	2	3	4
c1	1.44	4.77	162.3	98.6
c2	49	160.7	156.5	0.125
r1	-9.4	3.67	-0.75	-178
r2	-225.1	-161.1	-20.45	7.3
rc1	12.4	-7.87	-4.125	132.25
rc2	-189.6	-89	-78.125	2.5
pr	-102.55	-893.8	-1202	-1581
t1	-155			
t2	-41			

TABLE IV. TIME GAP (MS) COMPOSITE VS AVGSL

	Number Of Tours			
Topology Name	1	2	3	4
c1	-3.1	-3.5	216	546
c2	142.75	477	108	27
r1	2.91	117	188	76
r2	87	200	116	72
rc1	12.375	-43.6	37	-38
rc2	-83.75	585	151	70
pr	-199	-358	-584	-594
t1	303			
t2	5			

Despite the increase in the execution time, the composite algorithm is still suitable for real time applications.

ACKNOWLEDGEMENT

This work has been partly supported by the University of Piraeus Research Center.

REFERENCES

[1] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden, "Iterated Local Search for the team orienteering problem with time windows," Computers & Operations Research, 36: 3281-3290, 2009.

[2] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, Y. Tasoulas, "Cluster-Based Heuristics for the Team Orienteering Problem with Time Windows," in *Proceedings of 12th International Symposium on Experimental Algorithms* (SEA 13), pages 390-401, 2013.

[3] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou and N. Vathis, "Heuristics for the Time Dependent Team Orienteering Problem: Application to Tourist Route Planning," Computers & Operations Research, 62: 36-50, 2015.

[4] S. Boussier, D. Feillet, and M. Gendreau, "An exact algorithm for team orienteering problems," 4OR: A Quarterly Journal of Operations Research, 5:211-230.

[5] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, "Approximation Algorithms for Orienteering and Discounted-Reward TSP", In Foundations of Computer Science, 2003. Proceedings 44th Annual IEEE Symposium on, pages 46-55, oct 2003.

[6] Labadie N., Melechovsky J., Wolfler Calvo, R., "Hybridized evolutionary local search algorithm for the team orienteering problem with time windows", J. Heuristics 17,729-753, (2011).

[7] Montemanni, R., Gambardella, L.M., "An ant colony system for team orienteering problems with time windows", Found Comput. Decis. Sci. 34(4), 287-306 (2009).

[8] Hu, Q., Lim, A., "An iterative three-component heuristic for the team orienteering problem with time windows", Eur. J. Oper. Res. 232(2), 276-286 (2014).