ISSN 2277-3061

Roulette Wheel Selection based Heuristic Algorithm for the Orienteering Problem

Madhushi Verma*, Mukul Gupta, Bijeeta Pal, K. K. Shukla

Department of Computer Science and Engineering, IIT (BHU), Varanasi- 221005, India
madhushi.rs.cse@itbhu.ac.in
Department of Computer Science and Engineering, IIT (BHU), Varanasi- 221005, India
mukul.gupta.cse10@itbhu.ac.in
Department of Computer Science and Engineering, IIT (BHU), Varanasi- 221005, India
bijeeta.pal.cse10@itbhu.ac.in
Department of Computer Science and Engineering, IIT (BHU), Varanasi- 221005, India
kkshukla.cse@itbhu.ac.in

ABSTRACT

Orienteering problem (OP) is an NP-Hard graph problem. The nodes of the graph are associated with scores or rewards and the edges with time delays. The goal is to obtain a Hamiltonian path connecting the two necessary check points, i.e. the source and the target along with a set of control points such that the total collected score is maximized within a specified time limit. OP finds application in several fields like logistics, transportation networks, tourism industry, etc. Most of the existing algorithms for OP can only be applied on complete graphs that satisfy the triangle inequality. Real-life scenario does not guarantee that there exists a direct link between all control point pairs or the triangle inequality is satisfied. To provide a more practical solution, we propose a stochastic greedy algorithm (RWS_OP) that uses the roulette wheel selection method, does not require that the triangle inequality condition is satisfied and is capable of handling both complete as well as incomplete graphs. Based on several experiments on standard benchmark data we show that RWS_OP is faster, more efficient in terms of time budget utilization and achieves a better performance in terms of the total collected score as compared to a recently reported algorithm for incomplete graphs.

Keywords

Orienteering problem; complete graphs; incomplete graphs; roulette wheel selection method; Heuristic algorithm.

1 INTRODUCTION

Orienteering problem (OP) is a challenging NP-Hard combinatorial optimization graph problem that originated from a water sport where players need to visit a set of control points, starting at the source and reach the destination within a fixed time frame with an objective of collecting maximum possible rewards (associated with each control point). This problem is a combination of the travelling salesman problem (TSP) and the Knapsack problem (KP). The goal of maximizing the total collected score in OP is derived from KP and that of minimizing the total travel time from TSP. Unlike TSP, in OP it is not mandatory to visit each and every node of the network (Vansteenwegen et al., 2011). Many real-life situations can be modeled as OP like the transportation networks, home delivery system, telecommunication networks, logistics, trip planning in tourism industry, robot path planning, etc. In disaster management, a robot can be used with a map of its surrounding area to supply the necessary items like food, medicines, etc. to the affected locations. However, because the robot is battery operated, it cannot visit all the locations and the need here is to determine a path connecting the maximum number of locations (which are highly affected and requires immediate attention) that can be visited within the time bound (battery life of the robot) (Blum et al., 2003). Another example is that of a travelling salesman who does not have enough time to visit all the cities, but has some knowledge about the cities where maximum sales can take place. Therefore, the goal here is to maximize the total sales, but within the time constraint of a day or a week (Tsiligirides, 1984). In each of the stated applications, the task is to determine a solution, i.e. a path that collects the maximum rewards possible and also satisfies the time bound (Vansteenwegen et al., 2011; Schilde et al., 2009). Several variants of OP exist, which include team orienteering problem, orienteering and team orienteering with time window and the generalized orienteering problem (Vansteenwegen et al., 2011).

As stated earlier, OP is an NP-Hard problem so an exact algorithm is not practically feasible in terms of execution time for larger instances. However, a few exact algorithms were suggested by Laporte et al and Hayes et al based on the concepts of linear programming and dynamic programming (Laporte and Martello, 1990; Hayes and Norman, 1984). Other methods available in the literature include heuristic and approximation algorithms for OP. In 1984, one of the first heuristics for OP was proposed by Tsiligirides (1984). A four-phase heuristic and a centre-of-gravity heuristic was suggested by Ramesh and Brown and Golden et al respectively (Ramesh and Brown, 1991; Golden et al., 1987). Later, Fischetti et al stated a branch and cut heuristic for OP (Fischetti et al., 1998). Several other methods like artificial neural network, tabu search, pareto ant colony optimization, pareto variable neighbourhood search, etc. were introduced to deal with OP (Schilde et al., 2009; Wang et al., 1995; Gendreau et al., 1998). In 2013, Campos et al presented a Greedy Randomized Adaptive Search Procedure and the Path Relinking approach to solve OP (Campos et al., 2013). An approximation for the un-rooted version of OP was suggested (Awerbuch et al., 1999; Johnson et al., 2000) and Blum et al proposed an approximation algorithm for the rooted version of OP (Blum et al., 2003). Fomin et al, introduced an approximation algorithm for the time-dependent variant of OP (Fomin and Lingas, 2002). Most of the suggested algorithms for OP can be applied on complete graphs only but considering the practical applications of OP, it can be observed that many situations cannot be modeled only through complete graphs. The first genetic algorithm (GA) for OP was proposed by Tasgetiren (Tasgetiren, 2001) but Ostrowski et al presented a genetic algorithm for OP that can be solved for both complete as well as incomplete graphs (Ostrowski and Koszelew, 2011).

Here, we propose a stochastic greedy heuristic for OP (RWS_OP) that uses the roulette wheel selection method for determining the path that maximizes the total collected score within the specified time frame (T_{max}). The algorithm is guaranteed to reach the destination node (v_N) since the starting node (v_1) and the final node (v_N) are always the end points for all the generated paths. One necessary condition of OP is to ensure that a node is visited at most once, which in our algorithm is implemented by removing the explored nodes from the set of available nodes. This algorithm can be applied on both complete as well as incomplete graphs. We also compare our results with those reported by Ostrowski et al for large instances to show that RWS_OP executes more efficiently. Ostrowski et al proposed two algorithms one for incomplete graphs (IG) and the other that requires conversion of IG to complete graphs (CG) before OP can be solved. Few drawbacks of their paper are: (1) The paper does not categorically provide conclusive evidence about which of these algorithms is better. (2) Further, the authors allow each vertex to be visited more than once, but the reward is collected only on the first visit. This strategy is not only disadvantageous from the time budget point of view, but also produces invalid results (i.e. Non-Hamiltonian path). (3) In the second version of their algorithm, virtual edges need to be added using Dijkstra's algorithm, which results in unnecessary complication. (4) Their strategy requires application-specific complex crossover and mutation operations that often produce infeasible partial solutions that require additional correction/repair operations. In this paper, we propose a stochastic greedy algorithm that uses roulette wheel selection to avoid the search getting trapped in local maxima and removes all the disadvantages of Ostrowski's method mentioned above. It also outperforms Ostrowski's algorithm by improving the score and utilizing the time budget up to almost 99%. The objective function used in our method is motivated by greedy adaptive search procedure and path relinking (GRASP) technique suggested by Campos et al (Campos et al 2013). We show that roulette wheel selection with our new instance coding technique outperforms full GA implementation by Ostrowski et al in terms of both quality of solution and search time and space. In our method of candidate representation, we start with the shortest path between v_1 and v_N and using the roulette wheel proportionate selection scheme, keep adding nodes until T_{max} is reached. At each step, the probability of selecting a node is proportionate to its fitness defined in equation 9. Thus, without using crossover and mutation and without the need to maintain a population of possible solution, this algorithm is able to outperform GA based technique reported by Ostrowski et al. In section 2 of this paper, the problem formulation of OP is explained. Section 3 presents the various steps of the proposed algorithm. The observations based on the experiments performed on standard benchmarks are explained in section 4 and 5. Finally, the paper is concluded in section 6.

ISSN 2277-3061

2 PROBLEM FORMULATION

The orienteering problem can be modelled using a (complete or incomplete) weighted undirected graph $G(V,E)$ where $V = \{v_1, \dots\dots, v_N\}$ denotes the set of vertices and E denotes the set of edges. Let the time function on edges be $t: E \to \Re^+$ and a score function on nodes be $S: V \to \Re^+$. So if $V' \subseteq V$ then $S(V') = \sum_{v \in V'} S_v$ and if $E' \subseteq E$ then $t(E') = \sum_{e \in E'} t(e)$. The task is to compute a Hamiltonian path P, within the stated time bound (T_{max}) that connects the specified source (v_1), target (v_N) and also includes a subset (V') of V such that the total collected score is maximized (Vansteenwegen et al., 2011). To achieve the stated goal, here we use roulette wheel selection for exploring the various possible paths and return one that maximizes the total collected score within the specified time limit (T_{max}).

The OP can be presented as an integer programming problem and the formulation for the same is as follows (Vansteenwegen et al., 2011):

$$Max \sum_{i=1}^{N-1} \sum_{j=2}^{N} S_i x_{ij} \quad (1)$$

$$\sum_{j=2}^{N} x_{1j} = 1 \quad , \quad \sum_{i=1}^{N-1} x_{iN} = 1 \quad (2)$$

$$\sum_{i=1}^{N-1} x_{ik} \leq 1 \quad \forall k = 2, \dots\dots, N-1 \quad (3)$$

$$\sum_{j=2}^{N} x_{kj} \leq 1 \quad \forall k = 2, \dots\dots, N-1 \quad (4)$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max} \quad (5)$$

$$2 \leq u_i \leq N \quad \forall i = 2, \dots\dots, N \quad (6)$$

$$u_i - u_j + 1 \leq (N-1)(1 - x_{ij}) \quad \forall i,j = 2, \dots\dots\dots, N \quad (7)$$

$$x_{ij} \in \{0,1\} \quad \forall i,j = 1, \dots\dots, N \quad (8)$$

Here, the variable u_i denotes the position of vertex v_i in the path and if a vertex v_j is visited after vertex v_i , then x_{ij} = 1 else x_{ij} = 0. Equation 1 ensures that the objective of OP, which is maximization of the total collected score is fulfilled. The necessary condition that each path starts in v_1 and ends in v_N is ensured by equation 2. The constraint that each path remains connected and no vertex is visited more than once in a path is taken care by equations 3-4. The total time taken to traverse a path is within the specified time limit is ensured by equation 5. The requirement of eliminating sub tours is implemented by equation 6-7.

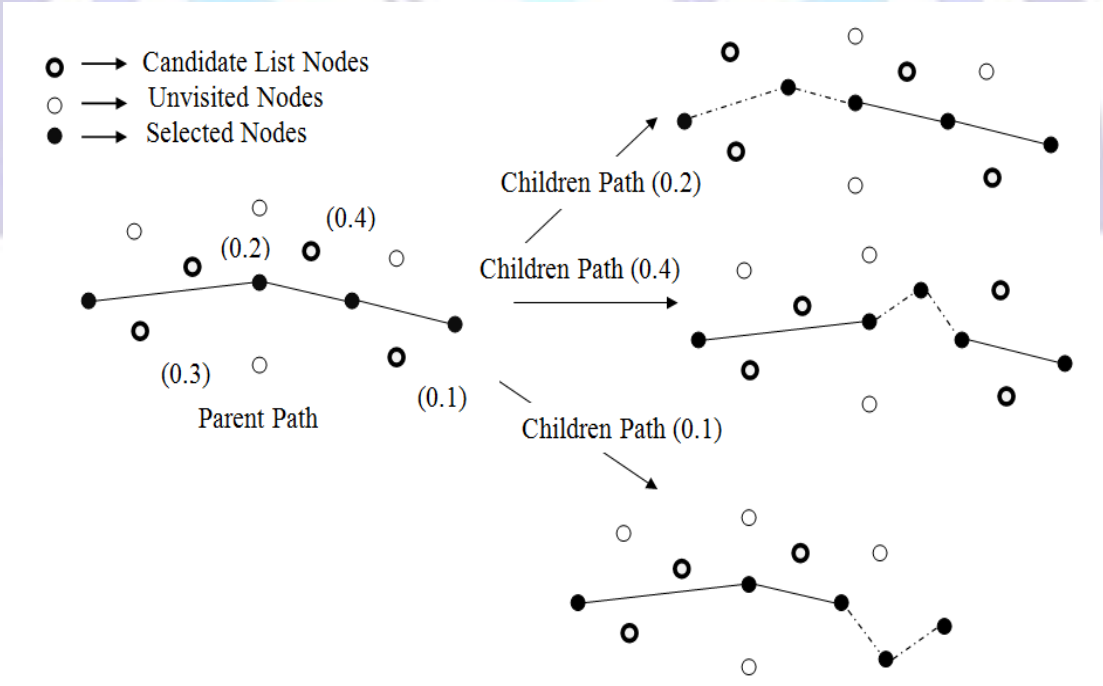


Fig 1: The process of selecting a path using roulette wheel selection function where the number in () denotes the probability of node selection.

ISSN 2277-3061

3 ALGORITHM RWS_OP

Input: A graph $G(V,E)$ with t_{ij}(time taken to traverse) value of each edge (e_{ij}) connecting vertex v_i and $v_j \in V$, S_i (Score) value of each vertex $v_i \in V$ and the $PathListSize$(maximum number of paths considered at each level).

Output: A Hamiltonian path with the highest possible collected score such that total travel time is within the specified time budget.

$Path$ is an array of nodes or vertices, which is a sequence connecting the source and the target.

$PathList$, $NewPathList$ and $ChildPathList$, are all queues and each of its element is a $Path$. $PathListSize$ is a constant whose value defines the maximum number of paths to be considered after each iteration.

$RWS\_OP(G, PathListSize, T_{max})$

1. ***Create*** $PathList$; // Array of paths which is initially empty.
2. $PathList \leftarrow \emptyset$;
3. $Path$ P $\leftarrow Dijkstra(v_1, v_n)$; // Shortest path between source (v_1) and target (v_N).
4. ***Insert*** P *to* $PathList$;
5. ***return*** $Generation(PathList)$;

$Generation(PathList)$

1. ***Create*** $NewPathList$,$Childpath List$; // Queues storing paths.
   $NewPathList \leftarrow \emptyset$;
   $ChildPathList \leftarrow \emptyset$;
2. ***for*** $i \leftarrow 0$ *to* $|PathList|$
   $a.Selection(PathList[i], ChildPathList)$; // $ChildPathList$ will contain children generated from each path in $PathList$.
3. ***for*** $i \leftarrow 0$ *to* $PathListSize$ // Selecting best $PathListSize$ children for next generation.
   $a.ChildPath = BestPath(ChildPathList)$; // The path with the maximum total score.
   Remove $ChildPath$ *from* $ChildPathList$;
   Insert $ChildPath$ *to* $NewPathList$;
   $b.$***if*** $|ChildPathList| == 0$
   break;
4. ***if*** $NewPathList == PathList$ // Terminate if no new child is generated and return the $BestPath$.
   return $BestPath(PathList)$;
5. ***return*** $Generation(NewPathList)$;

$Selection(Path\ P, ChildPathList)$ //$Path$ is an array of nodes that forms a path.

1. $q_{max} \leftarrow 0$;
2. ***for*** $i \leftarrow 0$ *to* $|V|$
3. $a.$***if*** $i \in P$ // If a node is already present in the path then ignore it.
   continue;
   $b.$***Calculate*** Δt_i ; // The time increment due to insertion of v_i at its best position.

$$c.q_i = \begin{cases} S_i/|\Delta t_i|, & \Delta t_i \geq 1 \\ S_i, & -1 \leq \Delta t_i < 1 \\ S_i * |\Delta t_i|, & \Delta t_i < -1 \end{cases};$$

$d.$ ***if*** $q_{max} \leq q_i$

$q_{max} \leftarrow q_i$

4. $Create\ CandidateList;$ // Array of candidate nodes used for roulette wheel selection.
5. $CandidateList \leftarrow \emptyset;$
6. ***for*** $i \leftarrow 0$ *to* $|V|$
7. $a.$***if*** $i \in P$

   continue;

   $b.$***if*** $(q_i \geq \alpha\ q_{max})\&\&\ (t_{parent} + \Delta t_i \leq T_{max})$ // α is the greediness parameter that decides which node should participate in roulette wheel selection.

   $c.$***Insert*** v_i *to* $CandidateList;$
8. ***if*** $|CandidateList| == 0$

   Insert P *to* $ChildPathList;$ // If no new nodes are added then insert the parent path P.

   Return;
9. ***for*** $i \leftarrow 0$ *to* $PathListSize$ // Generates $PathListSize$ number of children paths from path P.
10. $a.ChildNode \leftarrow RouleteWheel(CandidateList)$

    Remove $ChildNode\ from\ CandidateList;$

Insert $ChildNode$ *to Path P and* ***Insert*** *Path P to ChildPathList.*

Remove $ChildNode$ *From Path P;*

$b.$***if*** $|CandidateList| = 0$

break;

$RouleteWheel(CandidateList)$

1. $sum \leftarrow 0;$
2. ***for*** $i \leftarrow 0$ *to* $|CandidateList|$

   $sum += fitness(i);$ //$fitness(i) \leftarrow q_i$, where $q_i = \begin{cases} S_i/|\Delta t_i|, & \Delta t_i \geq 1 \\ S_i, & -1 \leq \Delta t_i < 1 \\ S_i * |\Delta t_i|, & \Delta t_i < -1 \end{cases}$
3. $RandomNumber \leftarrow Rand()\%(sum + 1)$
4. $PartialSum \leftarrow 0;$
5. $Selected \leftarrow 0;$
6. ***while*** $(RandomNumber > PartialSum)$

   $a.Selected++;$

   $b.PartialSum += fitness(selected);$
7. ***return*** $selected;$

$BestPath(PathList)$

1. $max \leftarrow 0; bestpath \leftarrow 0;$

2. $for\ i \leftarrow 0\ to\ |PathList|$

   a. $if\ \left(Score\left(PathList(i)\right)\right) > maxScore$ // *Score* ($Path\ P$) is the sum of the rewards associated with each node of

   $Path\ P$.

   $b. maxScore \leftarrow Score\left(PathList(i)\right); bestpath \leftarrow i;$

3. $return\ PathList(bestpath);$

The main aim of RWS_OP algorithm is to connect the source and the destination vertex and in between visit as many nodes as possible to collect the highest possible score within the given time bound. In lines 1-5 of $RWS\_OP()$, Dijkstra's algorithm is applied to find the shortest path connecting the source and target, and the path obtained is stored in $Path$. A queue of $Paths$, i.e. $PathList$ is maintained that contains the list of $Paths$ obtained after each iteration and is initialized with the shortest path P.

Lines 1-5 of $Generation()$ denotes that in each iteration (i.e. each run of), $Generation$ function inserts a node into the $Path$ obtained by the previous iteration. So, an iteration of $Generation$ function accepts a list of $Paths$ in the form of $PathList$ and for each $Path$ in the $PathList$, it calls the $Selection$ function. Another queue of $Paths$, i.e. $ChildPathList$ is maintained, which is initially empty. It stores the child paths generated using the $Selection$ function for each $Path$ of $PathList$. After each iteration, we consider only $PathListSize$ number of $Paths$ at a time. So $PathListSize$ number of $BestPaths$($BestPath$ is the one which has the highest value for total collected score) from the $ChildPathList$ are inserted into another queue of $Paths$, i.e. $NewPathList$. Then the termination condition is checked to determine whether a new child is generated by the $Generation$ function or not. If no new child is generated (i.e. all paths in the queue, $NewPathList$ is the same as that of $PathList$), then the $Generation$ function will return the $BestPath$ from the $PathList$ else, the $Generation$ function is recursively called for $NewPathList$.

The $Selection$ function is used for generating $ChildPaths$ from $Path\ P$. This function generates a $ChildPath$ which contains one node more than that already present in $Path$ P. This function decides the node to be inserted and its location in $Path\ P$. As shown in line 3 of $Selection()$, for each vertex that has not yet been inserted in $Path\ P$, its best position (one that leads to smallest increment in time, Δt_i) is evaluated and then the ratio q_i is computed. The ratio q_{max} is calculated in the following way:

$$(q_{max}) = max_{i \in (V \setminus P)}(q_i)$$

In lines 4-7 of $Selection()$, another list of nodes, named $CandidateList$ is created which is initially empty. All those nodes which satisfy the following inequality are inserted in the $CandidateList$:

$$CandidateList = \{i \in (V \setminus P) : (q_i \geq \alpha\ q_{\max}) \ \&\& \ (t_{parent} + \Delta t_i \leq T_{max})\}$$

Here α is the greediness parameter. Greater value of α denotes a more greedy solution and smaller value of α denotes a more random solution. As the value of α decreases, more nodes will be selected for the $CandidateList$. As stated in lines 8-10 of $Selection()$, if the $CandidateList$ is empty, $Path\ P$ is inserted in the queue $ChildPathList$, else $RouleteWheel$ selection method is used to choose a node from the $CandidateList$. The chosen node is removed from the $CandidateList$ and inserted at its best position in $Path\ P$ to generate a $ChildPath$ which is then en-queued into $ChildPathList$. This process of extracting nodes from the $CandidateList$ is repeated until a constant ($PathListSize$) number of $ChildPaths$ are generated or $CandidateList$ becomes empty.

Roulette wheel selection is a population-based selection method used in genetic algorithms that stochastically picks out a node based on their fitness value i.e. a node having greater fitness value has more chances of getting selected, although nodes with lower fitness also have a nonzero probability of selection. This assists the search in escaping local maxima. It is conceptually equal to giving each individual option, a portion of a circular roulette wheel proportional in area to the individual's fitness value (Zhang et al., 2012). As it can be seen, lines 1-7 of the $RouleteWheel\ ()$ selects a node from the $CandidateList$ using $FitnessProportionateSelection$ approach where an element is randomly chosen, but the probability of choosing an element with higher fitness is greater than choosing an element with lower fitness value. In our selection process, the fitness of an element of $CandidateList$ is computed using the following equation:

$$fitness(i) \leftarrow q_i, \text{ where } q_i = \begin{cases} S_i/|\Delta t_i|, & \Delta t_i \geq 1 \\ S_i, & -1 \leq \Delta t_i < 1 \\ S_i * |\Delta t_i|, & \Delta t_i < -1 \end{cases} \quad (9)$$

Therefore, nodes having a higher value of S_i and lower value of time increment (Δt), have a greater probability of getting selected and also the nodes having low S_i and high (Δt) values can be selected but with a lesser probability (> 0).

The time complexity of $Dijkstra's(v_1, v_N)$ algorithm is $O(|V|^2)$ where $|V|$ is the number of vertices in the Graph. In the $RouleteWheel$ function, there exists an iterative loop that runs $|CandidateList|$ times and because the $CandidateList$ can contain at most $(|V|-2)$ nodes, therefore the time complexity of $RouleteWheel$ will be $O(|V|)$.

In the selection function, line 2 is an iterative loop running for all the vertices of G i.e. $|V|$times and in each iteration, the best position of node i in the current $Path\ P$ is found which takes $O(|V|)$ time. Line 6 represents another iterative loop, which takes $O(|V|)$ time as it also runs for all the vertices of G and $Insert$ operation in the $CandidateList$ is similar to simple insertion in an array and takes $O(1)$time. $Remove$ operation in $CandidateList$ will take $O(|CandidateList|)$ or $O(|V|)$ time. $Remove$ operation in $ChildPathList$ takes $O(|V|)$ time as $O(|V|)$ time is required to bring the element to the front of $ChildPathList$ queue and $O(1)$ to dequeue it. $Insert$operation in $Path\ P$ takes $O(|V|)$ time. The iteration of line 9 takes $O(PathListSize * |V|)$ time as it runs for $PathListSize$ times, and each iteration takes $O(|V|)$ time. So the overall time complexity of $Selection$ function is $\left(O(|V|^2) + O(|V|) + O(PathListSize * |V|)\right)$, which is equivalent to $O(|V|^2)$ as $PathListSize \leq |V|$.

For the $Generation$ function, time complexity of line 2 which is an iterative loop is $O(PathListSize * |V|^2)$ as it runs for all the paths in $PathList$ and there can be at most $PathListSize\ Paths$ in this queue. In each iteration, it calls the $Selection$ function that takes $O(|V|^2)$ time. The $BestPath$ function will take $O(PathListSize)$ time. Since it is a recursive function, after each recursion, the $Paths$ contained in $PathList$ will increase by 1. The function will terminate when no new node is available that can be inserted in the $Path$. So, the recurrence relation can be written as:

$$T(M) = T(M-1) + PathListSize * |V|^2$$

Here, $T(M)$ is the time complexity for the $Generation$ function where, M is the number of nodes that can be added to the $Paths$ contained in $PathList$ and after one iteration the function calls itself with $M-1$ number of nodes that can be added to the $Paths$ of $NewPathList$. Solving the above recurrence, we get the overall complexity as $O(|V|^3)$.

The main function, i.e. RWS_OP uses the Dijkstra's algorithm ($O(|V|^2)$ time), $Selection$ function ($O(|V|^2)$ time) and $Generation$ function ($O(|V|^3)$ time). Therefore, the overall time complexity of RWS_OP is $O(|V|^3)$. The memory consumption of RWS_OP is $|PathListSize|^2 * N$ as opposed to Ostrowski_CG and Ostrowski_IG methods that occupies $PopulationSize * N$ memory where $N = Number\ of\ nodes\ in\ the\ graph$.

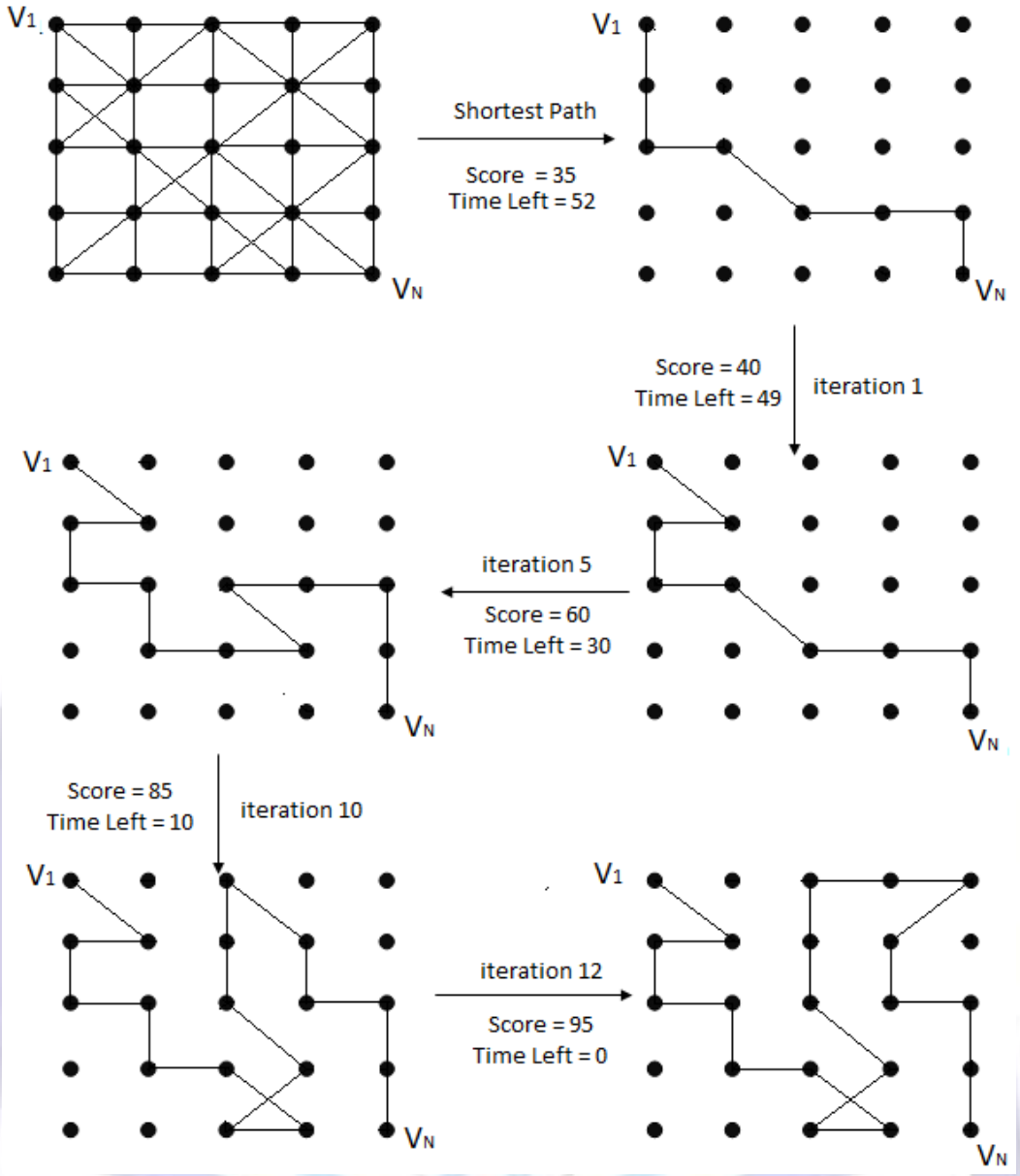


Fig 2: Progression of RWS_OP algorithm for a graph with 25 nodes with source $(V_1) = 1$, destination $(V_N) = 25$ and $T_{max} = 70$.

4 EXPERIMENTAL ANALYSIS

Most of the standard benchmark instances available for the orienteering problem include Set 1, Set 2 and Set 3 given by Tsiligirides, Set 64 and Set 66 given by Chao etc. (www.mech.kuleuven.be/en/cib/op/). In each of the available instances, only scores and coordinates of each vertex are specified leading to the formation of complete graphs satisfying the triangular inequality. However, in the real world, there is no guarantee that two nodes will be connected through a direct path. To deal with such cases, graphs are usually complemented with fictional edges by running Dijkstra's algorithm for every node before applying the classic OP algorithms available for complete graphs, but this results in a considerable increase in the search space size. Our method, works on complete as well as incomplete graphs without the need to insert fictional edges.

We have implemented our code in C++ using CodeBlocks on an Intel Core i5 650 at 2.20 GHz. As stated before, RWS_OP is capable of handling both complete as well as incomplete graphs, and here we present a few observations by applying RWS_OP on a real road network data based on 160 and 306 cities of Poland (http://piwonska.pl/p/research/). It consists of two text files -cities.txt and distances.txt. The file cities.txt specifies the names of the cities and the score of each, which is assigned based on the number of inhabitants in that city, i.e. $score = inhabitants/10000$. The distances.txt file was created from the real map of Poland representing the roads as edges in the graph stating for a particular city, its adjacent cities and their corresponding edge lengths.

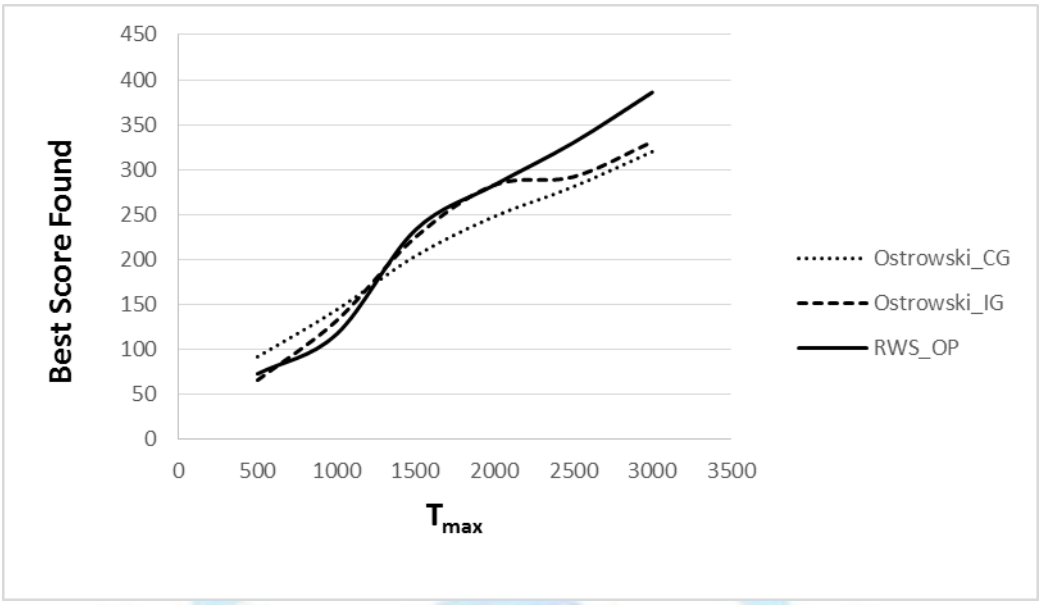
ISSN 2277-3061

5 DISCUSSION

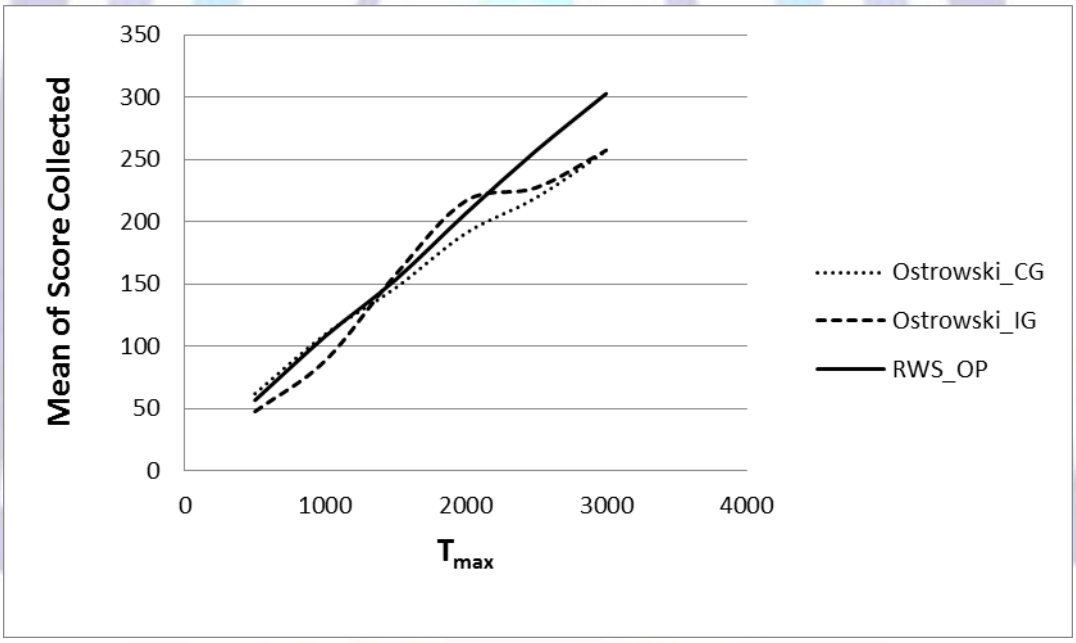
The following tables and plots show a comparison of our results with those obtained by applying the genetic algorithm suggested by Ostrowski et al on the same instances (Ostrowski and Koszelew, 2011). As can be seen in Table 1 and Fig. 3, it is observed that RWS_OP performs better for larger T_{max} values (taking the same first and last node i.e. source = destination) than the one proposed by Ostrowski et al because the highest total collected score attained for a graph with 306 nodes is greater in our case. Furthermore, the mean score and 95% confidence interval (CI) of mean for 30 runs $\left(i.e\ CI = \frac{1.96 * standard\ deviation\ of\ the\ number\ of\ runs}{\sqrt{number\ of\ runs}}\right)$ are higher in our case when compared to the values obtained by the genetic algorithm. Fig. 4 shows that for the same T_{max} value, there is a significant decrease in the execution time of RWS_OP as compared to the CG (Complete Graph version) and IG (Incomplete Graph version) of the genetic algorithm proposed by Ostrowski et al. It is also seen that the execution time increases linearly with the increase in T_{max} because increase in T_{max} leads to the exploration of more number of nodes. Fig. 5 presents the effect on scores by varying the value of the greediness parameter (α), which balances the degree of randomness and greediness. Increasing the value of the greediness parameter (α) makes the algorithm more greedy and the value of score obtained as a result of several executions of our code is the same most of the times i.e. lesser variation in the score, whereas decreasing the value of (α) leads to a situation with greater randomness, i.e. more variation in the scores obtained as shown through smaller and larger boxes respectively. It is also observed that the maximum score achieved increases with the increase in (α) however, for a large number of executions, a greater maximum score can be obtained even for smaller values of (α). RWS_OP is also efficient in terms of the time utilization (as shown in Table 2) as almost 99% of the specified time budget (T_{max}) is utilized, which helps in determining a better path that covers almost 70% of the cities, thus leading to greater total collected score. As the algorithm progresses, one node is added to the final path after each iteration, which results in an increase in the total collected score and decrease in the given time budget as shown in Fig. 6. Most of the experiments are performed at $\alpha = 0.6$ because at 0.6, both randomness and greediness come into play and as stated before, decreasing the value of α makes the algorithm more random and increasing it makes RWS_OP greedier. RWS_OP uses roulette wheel selection for choosing a node to be added in the path so different runs of the same algorithm with the same input parameters may result in selection of different nodes for being added in the final path and the trend in the utilization of time budget and an increase in the total collected score for three different runs at $\alpha = 0.2$ and $T_{max} = 1500$ for 160 cities instance is shown in Fig. 7. Fig. 8 shows how the time budget is utilized, and the total collected score increases with each iteration of RWS_OP for different values of α at $T_{max} = 1500$. The proposed heuristic is capable of exploring almost 70% of the search space as shown in Fig. 9 (a) and 9 (b) is a box plot showing the percentage of nodes explored and unexplored at $T_{max} = 7000$ for different values of α. As can be observed in 9 (b), the percentage of nodes explored (NE) is higher than the percentage of nodes unexplored (NU). Furthermore, the percentage of score collected for the explored nodes is higher than the percentage of score left out (score that could not be collected) for lower values of α, as a lower value of α induces more randomness into RWS_OP. RWS_OP is efficient both in terms of time and space complexity when compared to Ostrowski_CG and Ostrowski_IG methods, therefore, RWS_OP can be implemented for larger values of T_{max} which helps in achieving a higher total collected score for the considered instances as shown in Table 3 and Fig. 10.

Table 1. Comparison of maximum, mean and confidence Interval (CI) for mean of scores obtained by RWS_OP (keeping $v_1 = v_N$ i.e. $v_1 = v_N = 1$) with those obtained by executing the Ostrowski'salgorithm (Please refer Ostrowski and Koszelew, 2011, their Table 5 for Ostrowski_CG and Table 7 for Ostrowski_IG) on Real Road Network database with 306 cities of Poland.

T_{max}	RWS_OP(α=0.6)			Ostrowski_CG (highest fitness/travelTime)			Ostrowski_IG (fitness-Gain^2/travelTimeIncrease)		
	Mean	CI for Mean	Maximum	Mean	CI for Mean	Maximum	Mean	CI for Mean	Maximum
500	56.56	±3.5	73	61.9	±3.5	92	47.5	±3.3	66
1000	107.9	±2.64	117	109.5	±5.4	144	88.3	±6.0	132
1500	153.3	±9.29	233	146.7	±8.8	204	157.4	±13.5	225
2000	206.5	±13.1	283	190.6	±9.7	248	216.6	±12.8	283
2500	256.9	±13.07	330	219.1	±10.7	281	227.2	±13.9	292
3000	302.7	±13.4	386	256.9	±11.4	320	257.3	±15.2	331



(a)



(b)

Fig 3: Comparison of (a) maximum score and (b) mean score of each method with respect to time budget (T_{max}) based on 30 runs at $\alpha = 0.6$ for Real Road Network database with 306 cities of Poland.

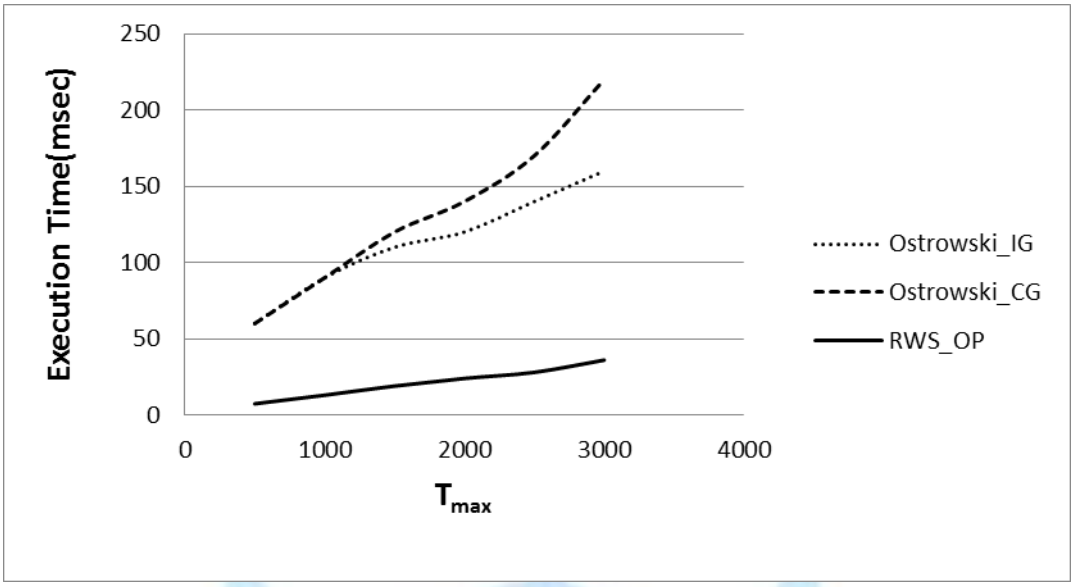
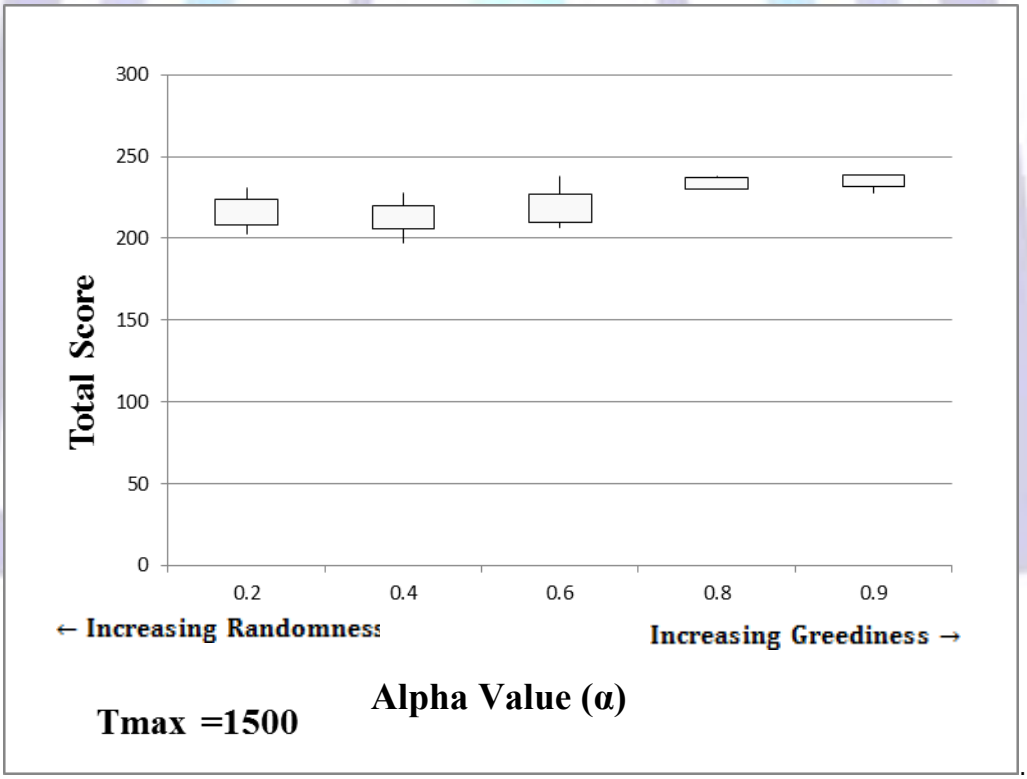
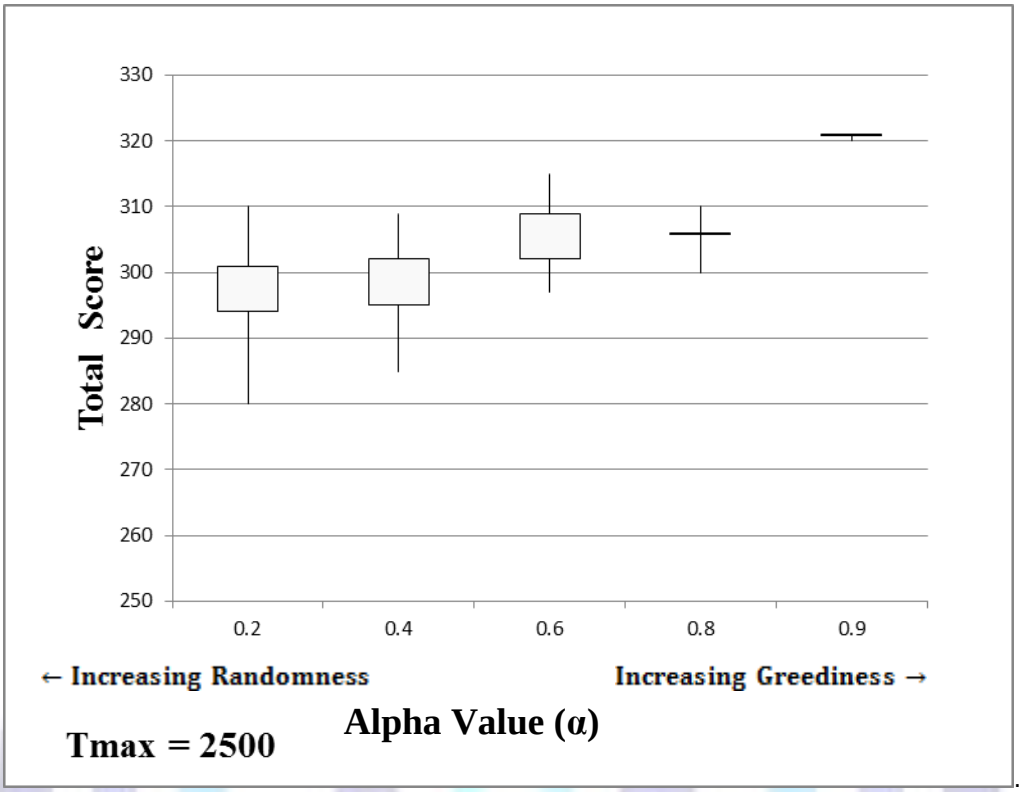


Fig 4: Comparison of execution time of each method with respect to time budget (T_{max}) based on 30 runs at $\alpha = 0.6$ for Real Road Network database with 306 cities of Poland.



(a)



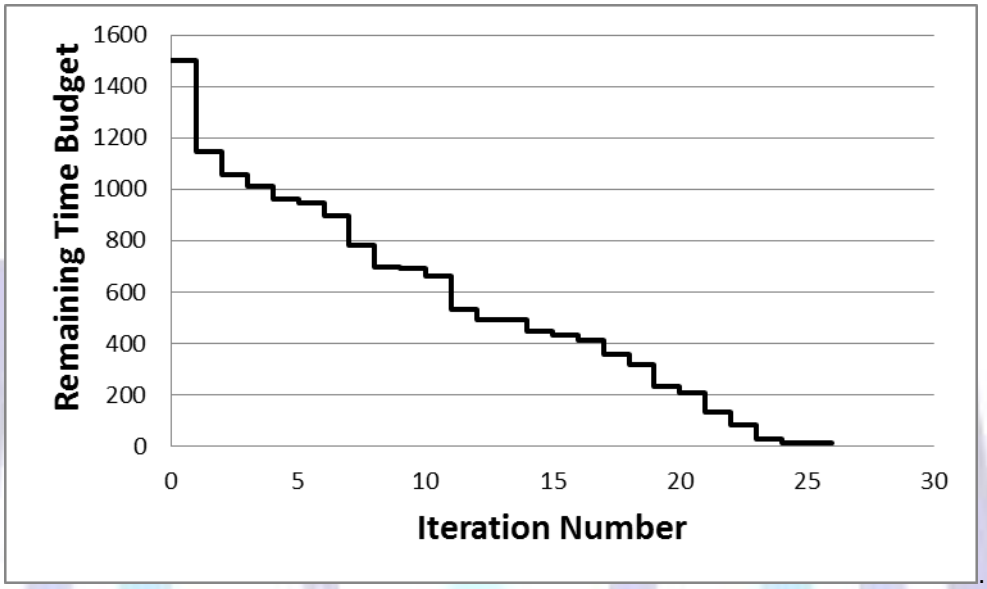
(b)

Fig 5: Comparison of score with respect to α for (a) $T_{max} = 1500$ and (b) $T_{max} = 2500$ for a Real Road Network database with 306 cities of Poland.

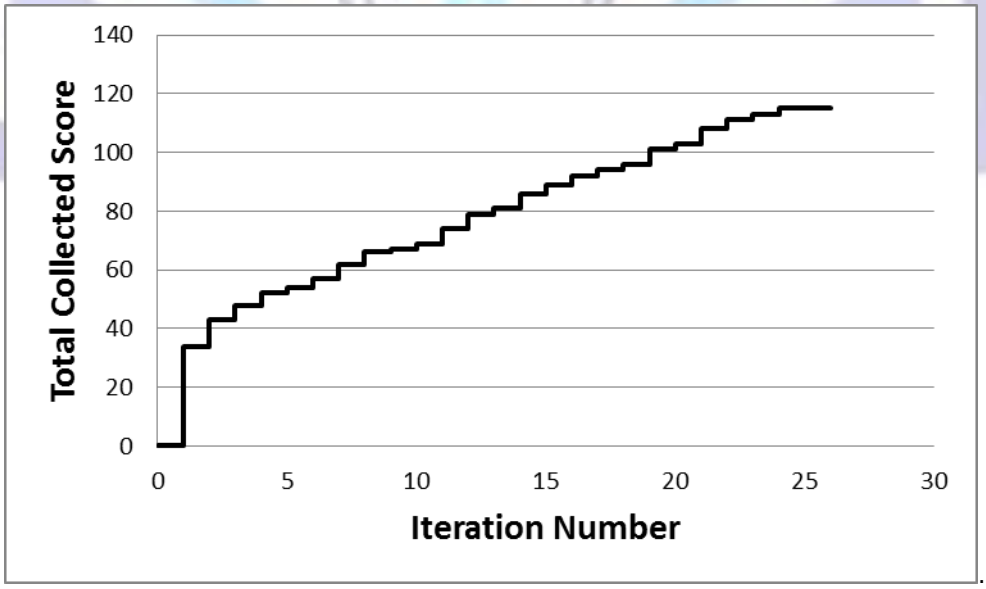
Table 2. The Highest Score Collected, Mean of Score Collected, Mean Time to Traverse the Path and % of Time Budget Utilized values obtained by RWS_OP at $\alpha = 0.6$ (keeping $v_1 \neq v_N$ i.e. $v_1 = 1$ and $v_N = 306$ for 306 cities and $v_1 = 1$ and $v_N = 160$ for 160 cities) when implemented on a Real Road Network database with 306 cities and 160 cities of Poland.

T_{max}	306 cities (α=0.6)				160 cities (α=0.6)			
	Highest Score Collected	Mean of Score Collected	Mean Time to Traverse the Path	% of Time Budget Utilized	Highest Score Collected	Mean of Score Collected	Mean Time to Traverse the Path	% of Time Budget Utilized
500	122	120.1	495.1	99.02	49	48.83	496.67	99.34
750	154	150.3	743.4	99.12	67	65.1	747.07	99.6
1000	177	174.3	995.1	99.51	88	76.8	990.8	99.08
1250	210	195.1	1246.53	99.72	104	102.3	1230.3	98.42
1500	243	220.4	1495.33	99.69	117	115	1488.37	99.25
1750	261	243.6	1742.72	99.58	129	127.9	1739.1	99.38
2000	286	270	1993.4	99.67	145	142.6	1993.63	99.68
2250	310	291	2244.8	99.77	162	156.1	2242.4	99.66
2500	324	312.4	2493.76	99.75	185	176.4	2485.8	99.43
2750	345	332.5	2744.8	99.81	202	193.2	2736.4	99.5

3000	371	349.6	2993.1	99.77	214	205.1	2980.3	99.34
3500	411	392.1	3495.47	99.87	250	243.6	3488.9	99.68
4000	451	436.9	3995	99.88	271	265.1	3982.7	99.57
4500	502	487.1	4494.3	99.87	306	290.23	4479.4	99.54
5000	537	524.5	4995.2	99.90	314	307.53	4960.9	99.22
5500	574	561	5491.7	99.85	322	316.9	5319.9	96.73
6000	620	593.3	5990.7	99.85	322	314.7	5336	88.93

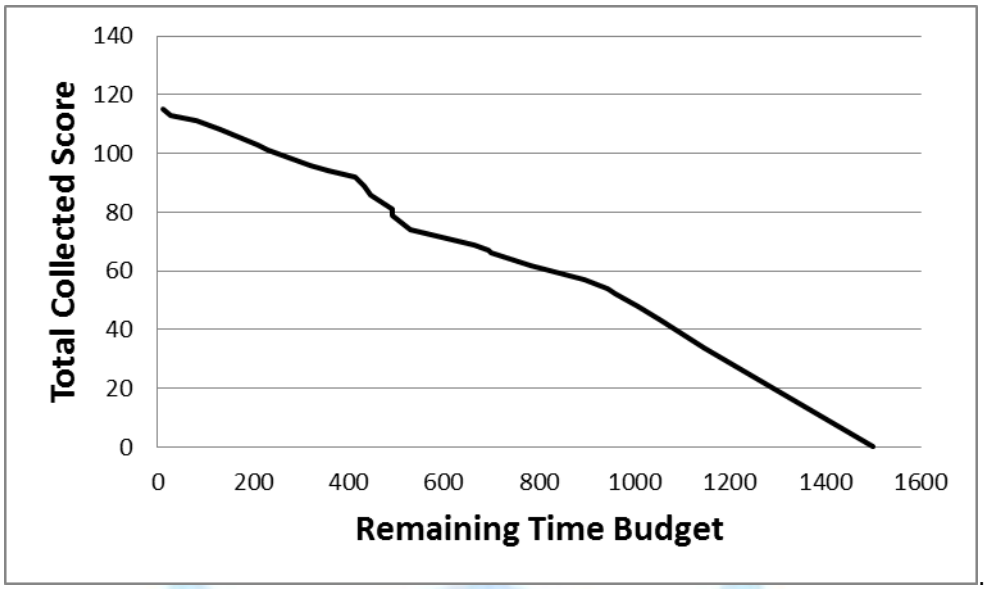


(a)



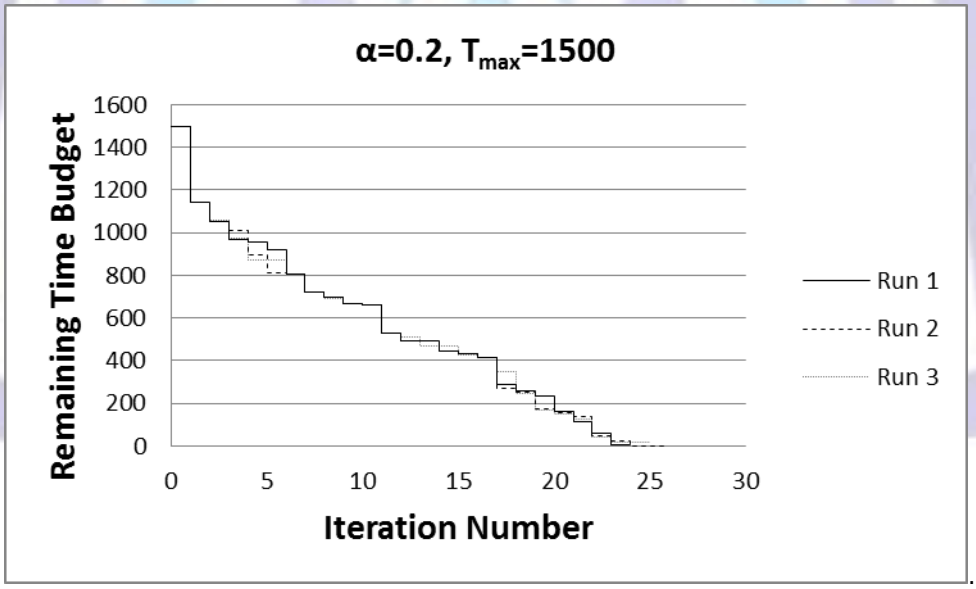
(b)

ISSN 2277-3061



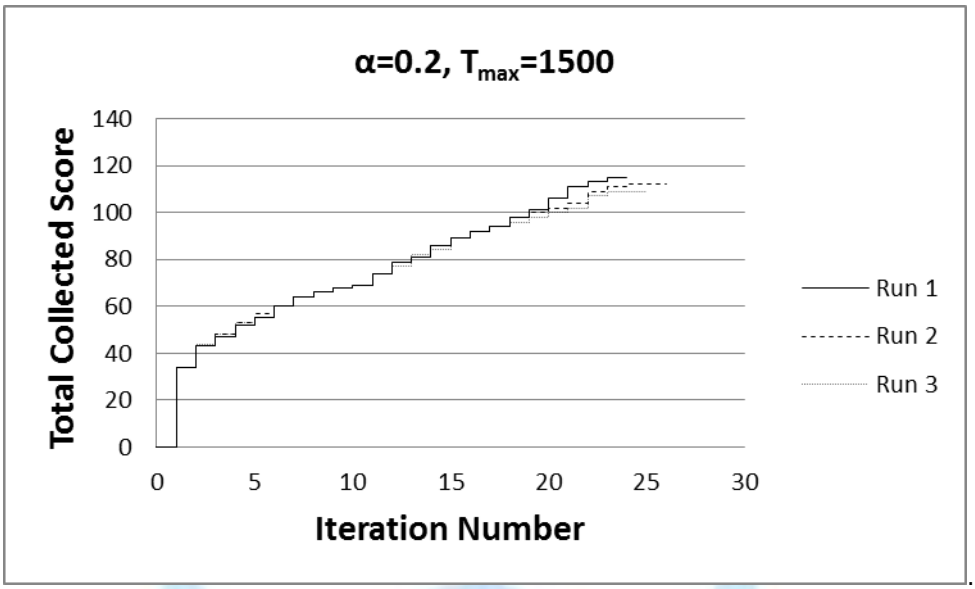
(c)

Fig 6: Plots showing (a) utilization of the time budget and (b) increase in the total collected score at $\alpha = 0.6$ and $T_{max} = 1500$ for a Real Road Network database with 160 cities of Poland. As the RWS_OP algorithm progresses, with each iteration, a node is added to the final path which results in an increase in the total collected score and decrease in the time budget as shown in (c).



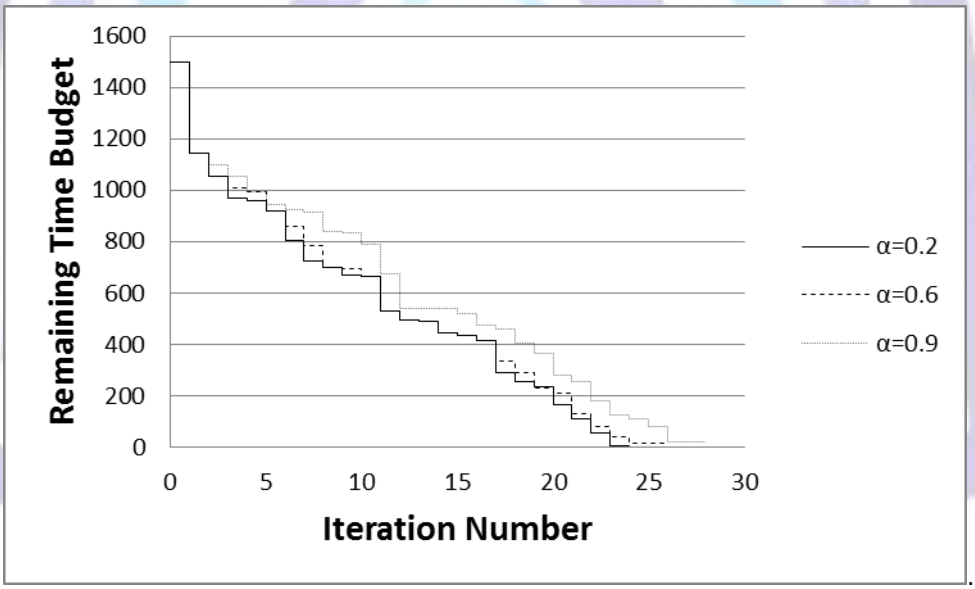
(a)

ISSN 2277-3061



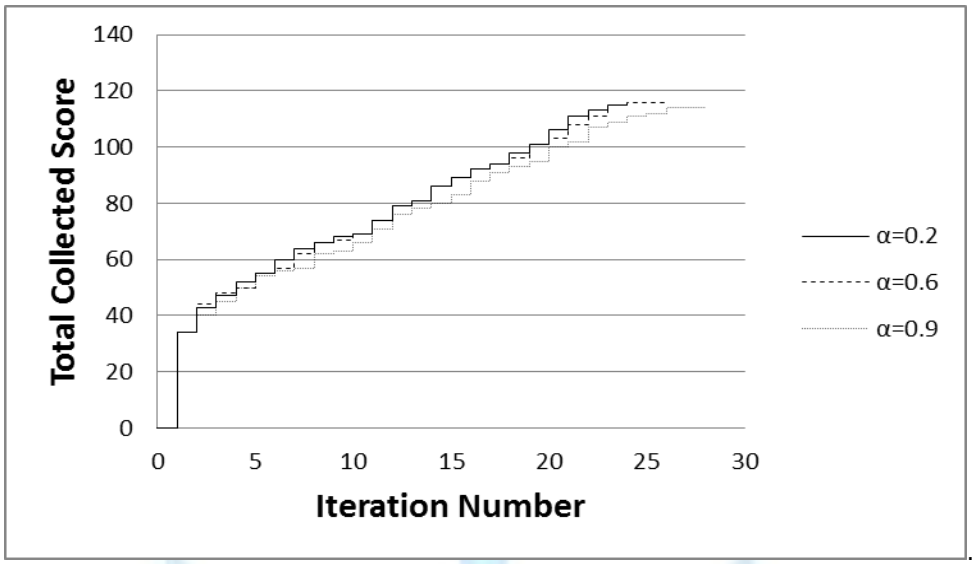
(b)

Fig 7: Plots showing the observation of three different runs of RWS_OP with $\alpha = 0.2$ and $T_{max} = 1500$ for a Real Road Network database with 160 cities of Poland. As the algorithm progresses, it results in (a) decrease in the time budget and (b) increase in the total collected score as shown above.



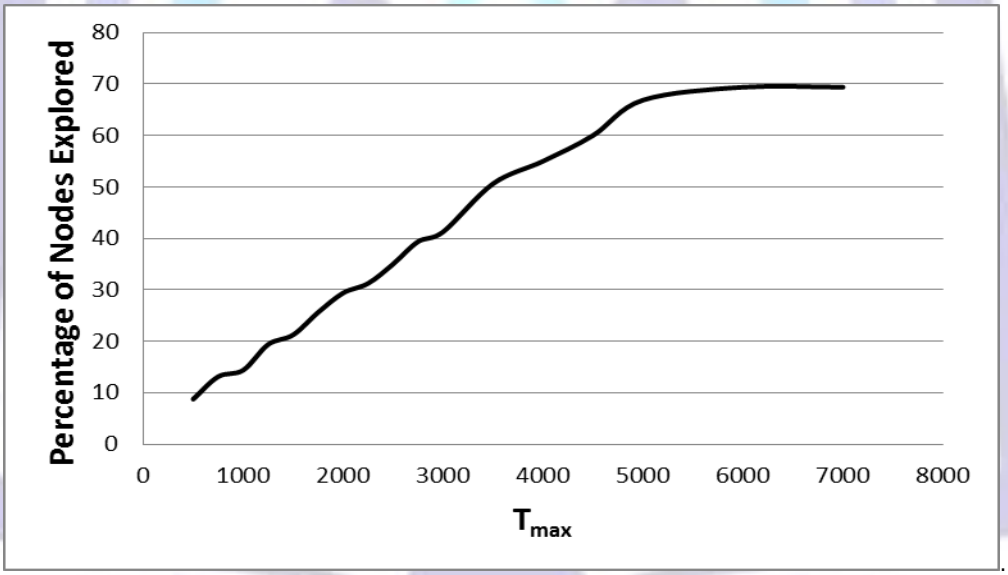
(a)

ISSN 2277-3061

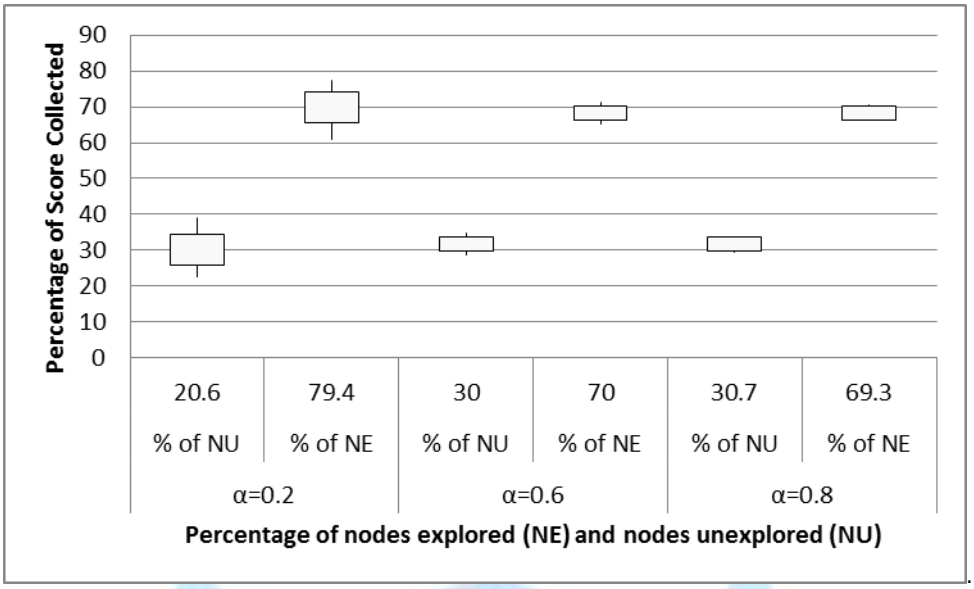


(b)

Fig 8: Plots showing (a) utilization of the time budget and (b) increase in the total collected score for three different α values at $T_{max} = 1500$ for a Real Road Network database with 160 cities of Poland.



(a)



(b)

Fig 9: Plots showing (a) the percentage of nodes explored with the increase in T_{max} values at $\alpha = 0.6$ and (b) percentage of nodes explored and unexplored for different values of α at $T_{max} = 7000$ for a Real Road Network database with 160 cities of Poland for 30 runs.

Table 3. The Highest Score Collected, Mean of Score Collected and confidence interval (CI) for Mean of Score Collected obtained by RWS_OP when implemented on a Real Road Network database with 306 cities of Poland for different T_{max} values at $\alpha = 0.6$ (keeping $v_1 = v_N$ i.e. $v_1 = v_N = 1$).

T_{max}	RWS_OP(α=0.6)		
	Mean of Score Collected	CI for Mean of Score Collected	Highest Score Collected
500	56.56	±3.5	73
1000	107.9	±2.64	117
s1500	153.3	±9.29	233
2000	206.5	±13.1	283
2500	256.9	±13.07	330
3000	302.7	±13.4	386
3500	353.16	±14.9	430
4000	427.8	±12.54	460
4500	466.7	±13.5	508
5000	506.1	±12.2	548
5500	553.2	±11.12	593
6000	595.2	±6.4	645
7000	653.1	±6.64	686
8000	718.2	±4.23	743
9000	767.8	±4.75	784
10000	769	±8.04	792
11000	769	±7.33	793

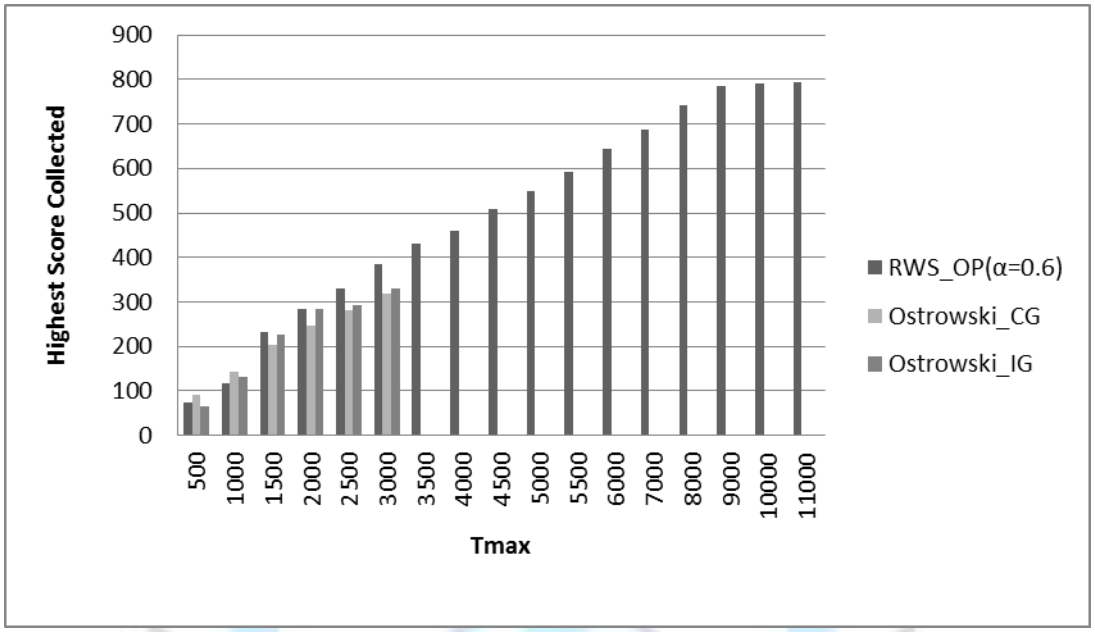


Fig 10: Plot showing that RWS_OP can achieve higher total collected score for larger T_{max} values as compared to Ostrowski_CG and Ostrowski_IG methods when implemented on a Real Road Network database with 306 cities of Poland at $\alpha = 0.6$.

6 CONCLUSION

In this paper, we considered the orienteering problem which is a NP-Hard combinatorial optimization problem and suggested a heuristic that uses the roulette wheel selection process for obtaining a Hamiltonian path that satisfies the time bound and helps in maximizing the total collected score. RWS_OP differs from the other techniques available in the literature as it can be applied on both complete as well as incomplete graphs whereas most of the existing algorithms can only be applied on complete graphs. Through experimental analysis, we showed that RWS_OP is more efficient than the previously suggested method by Ostrowski et al for incomplete graphs in terms of execution time. For a particular time bound, the proposed heuristic (RWS_OP) achieves a higher total collected score than the genetic algorithm of Ostrowski et al, utilizes almost 99% of the given time budget and is capable of exploring 70% of the considered search space. It is expected that when our algorithm is augmented with path relinking meta-heuristic (Glover and Laguna, 2000) its effectiveness will be further enhanced. In the future, we plan to integrate elite sub-paths found at earlier stages of the algorithm to produce new near optimal solutions. It will also be interesting to study the effect of incorporating adaptive or incremental beam search with the heuristic used in this paper.

REFERENCES

[1] Awerbuch, B., Azar, Y., Blum, A. and Vempala, S. 1999. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen, *Siam J. Computing,* Vol. 28, pp. 254–262.

[2] Blum, A., Chawla, S., Karger, D. R., Lane, T., Meyerson, A. and Minkoff, M. 2003. Approximation Algorithms for Orienteering and Discounted-Reward TSP, *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pp. 1-10.

[3] Campos, V., Marti, R., Sanchez-Oro, J. and Duarte, A. 2013. GRASP with Path Relinking for the Orienteering Problem. http://www.uv.es/rmarti/paper/docs/routing7.pdf.

[4] Fischetti, M., Salazar, J. and Toth, P. 1998. Solving the orienteering problem through branch-and-cut, *INFORMS Journal on Computing*, Vol. 10, pp. 133–148.

[5] Fomin, F. V. and Lingas, A. 2002. Approximation algorithms for time-dependent orienteering, *Information Processing Letters*, Vol. 83, pp. 57–62.

[6] Gendreau, M., Laporte, G. and Semet, F. 1998. A tabu search heuristic for the undirected selective travelling salesman problem, *European Journal of Operational Research*, Vol. 106, pp. 539–545.

[7] Glover, F. and Laguna, M. 2000. Fundamentals of scatter search and path relinking, *Control Cybern*, Vol. 29 No. 3, pp. 653–684.

[8] Golden, B., Levy, L. and Vohra, R. 1987. The orienteering problem, *Naval Research Logistics*, Vol. 34, pp. 307–318.

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

[10] Johnson, D., Minkoff, M. and Phillips. S. 2000. The prize collecting steiner tree problem: Theory and practice, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 760–769.

[11] Laporte, G. and Martello, S. 1990. The Selective Traveling Salesman Problem, *Discrete Applied Mathematics*, Vol. 26, pp. 193-207.

[12] Ostrowski, K. and Koszelew, J. 2011. The Comparison of Genetic Algorithms which Solve Orienteering Problem using Complete and Incomplete Graph, *Informatyka*, Vol. 8, pp. 61-77.

[13] Ramesh, R. and Brown, K. 1991. An efficient four-phase heuristic for the generalized orienteering problem, *Computers and Operations Research*, Vol. 18, pp. 151–165.

[14] Schilde, M., Doerner, K. F., Hartl, R. F. and Kiechle, G. 2009. Metaheuristics for the bi-objective orienteering problem, *Swarm Intelligence*, Vol. 3, pp. 179-201.

[15] Tasgetiren, M. 2001. A genetic algorithm with an adaptive penalty function for the orienteering problem, *Journal of Economic and Social Research*, Vol. 4 No. 2, pp. 1–26.

[16] Tsiligirides, T. 1984. Heuristic methods applied to orienteering, *Journal of the Operational Research Society*, Vol. 35, pp. 797–809.

[17] Vansteenwegen, P., Souffriau, W. and Oudheusden, D. V. 2011. The orienteering problem: A survey, *European Journal of Operational Research*, Vol. 209, pp. 1–10.

[18] Wang, Q., X. Sun, B. Golden, and J. Jia. 1995. Using artificial neural networks to solve the orienteering problem. *Annals of Operations Research* 61:111–120.

[19] Zhang, L., H. Chang, and R. Xu. 2012. Equal-width Partitioning Roulette Wheel Selection in Genetic Algorithm. In: *Proceedings of the Conference on Technologies and Applications of Artificial Intelligence*, pp. 62-67.