A Stochastic Greedy Heuristic Algorithm for the Orienteering Problem

Madhushi Verma*
Department of Computer Science and Engineering
IIT (BHU)
Varanasi, India
madhushi.rs.cse@itbhu.ac.in
(*Corresponding Author)

Bijeeta Pal
Department of Computer Science and Engineering
IIT (BHU)
Varanasi, India
bijeeta.pal.cse10@itbhu.ac.in

Mukul Gupta
Department of Computer Science and Engineering
IIT (BHU)
Varanasi, India
mukul.gupta.cse10@itbhu.ac.in

K. K. Shukla
Department of Computer Science and Engineering
IIT (BHU)
Varanasi, India
kkshukla.cse@itbhu.ac.in

Abstract— **In this paper we introduce a new stochastic greedy heuristic algorithm for the orienteering problem (OP) which is an NP-Hard combinatorial optimization graph problem. The goal of OP is to determine a Hamiltonian path that connects the specified source and target, includes a set of control points and achieves the best possible total collected score within the fixed time frame. This problem finds application in several fields like logistics, telecommunication networks, tourism industry etc. In each of these applications, it is necessary to provide a practical modelling and the best way to tackle this situation is to use incomplete graphs. However, most of the techniques available in the literature can only be applied on complete graphs. Unlike other algorithms, our algorithm can be applied on both complete and incomplete graphs. We have implemented our algorithm using four different selection procedures and evaluated their performance on standard benchmarks. We find that the algorithm works best with the roulette wheel selection.**

Keywords- Orienteering problem, selection methods, incomplete graphs.

I. INTRODUCTION

Orienteering Problem (OP) which is a combination of two well-known problems i.e. travelling salesman problem (TSP) and Knapsack problem (KP) originated from a water sport where the goal is to determine a path connecting the source and the target and at the same time explore a set of control points. Not all control points can be visited because of the limit on available time (or total distance travelled) [1]. OP is an NP-Hard combinatorial optimization problem and finds application in various fields like the tourism industry, robot path planning in disaster management, home delivery system, telecommunication networks, logistics, transportation networks etc. Several variants of OP like team orienteering, orienteering and team orienteering with time window, generalized orienteering are defined in the literature which models different real life situations [1].

In 1990, Laporte et al suggested an exact algorithm for OP using the concept of linear programming [2]. Another exact algorithm was introduced by Hayes et al based on the concept of dynamic programming [3]. However, as OP is NP-Hard, it is not feasible in terms of execution time to implement exact algorithms for large instances. So, the efficient technique to solve OP is to use some heuristic or approximation algorithm and the first heuristic for OP was proposed by Tsiligirides in 1984 [4]. In 1991, a four-phase heuristic was suggested by Ramesh and Brown [5]. Golden et al introduced the centre-of-gravity heuristic in 1987 [6]. A well-known branch and cut method to solve OP was presented by Fischetti et al [7]. Later, methods like tabu search and artificial neural networks were proposed to solve OP [8-9]. In 2009, pareto ant colony optimization and pareto variable neighbourhood search techniques were suggested by Schilde et al to deal with OP [10]. Campos et al in 2013, introduced a method called Greedy Randomized Adaptive Search Procedure (GRASP) which is an efficient heuristic for tackling the orienteering problem [11]. For the time dependent variant of OP, an approximation algorithm was suggested by Fomin et al [12]. Other approximation algorithms for the un-rooted version and rooted version of OP were proposed by Awerbuch et al., Johnson et al and Blum et al respectively [13-15]. As stated earlier, OP finds several real life applications and most of the practical situations cannot be modelled through complete graphs only. A more realistic picture can be presented by modelling such situations using incomplete graphs. However, most of the algorithms available in the literature for OP can be applied on complete graphs only. Tasgetiren proposed the first genetic algorithm for OP but in 2011, Ostrowski et al presented another genetic algorithm that can be applied on both complete as well as incomplete graphs [16-17].

In this paper, we present a heuristic for OP (SEL_OP) which can be implemented on both complete as well as incomplete graphs. In our algorithm, we use different selection procedures and compare the results to determine the selection method that helps in achieving the best possible score for the orienteering problem.

978-1-4799-6758-2/14/$31.00 ©2014 IEEE

II. PRE-REQUISITES

A. Problem Definition

In OP the aim is to determine a Hamiltonian Path P that connects the stated source (v_1) and target (v_N), includes a subset (V') of the vertex set V such that the total collected score can be maximized within the stated time budget (T_{max}). OP can be represented by an undirected weighted graph (complete or incomplete) $G(V, E)$ where V is the set of vertices and E is the set of edges. We associate two functions i.e. a time function and a score function to the edges and vertices respectively. Let $t: E \rightarrow \Re^+$ denote the time function and $S: V \rightarrow \Re^+$ signify the score function. So, for a subset $V' of\ V$ and $E' of\ E$, we have $S(V') = \sum_{v \in V'} S_v$ and $t(E') = \sum_{e \in E'} t(e)$ respectively [1].

The OP can be stated as an integer programming problem which is as follows [1]:

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

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

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

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

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

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

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

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

Equation 1 represents the objective function of OP i.e. maximization of the total collected score. Few other constraints like a path should have v_1 as its starting and v_N as its ending node is taken care by equation 2 and no vertex is visited more than once and the path remains connected is ensured by equation 3-4. The important condition that the path satisfies the time bound (T_{max}) is represented by equation 5. The need to eliminate sub tours is executed by equations 6 and 7. Variable u_i signifies the position of vertex v_i in the path and x_{ij} = 1 if v_j is visited after v_i otherwise, x_{ij} = 0.

B. Selection Methods

1) Tournament Selection

In this selection technique, one individual is chosen from a population of individuals. Several tournaments are run among the randomly selected individuals from a population and the best ones are copied to the next generation (those that win the tournament i.e. have the best fitness value). The process is repeatedly performed several times to saturate the population. The tournament size is denoted by q and the most common tournament size is q=2. By altering the tournament size, the selection pressure can be easily adjusted. In case the tournament size is large then weak individuals have a lesser chance of getting selected. Some advantages of this method are (1) it does not require any sorting mechanism and therefore can be implemented in O(n) time and (2) it supports parallel architecture [18-20].

2) (μ, λ) Selection

This selection technique was formulated with the idea of reducing the offspring population generated as a result of recombination and mutation. In this case, the offspring population of size $\lambda \geq \mu$ is reduced by selecting μ best offspring individuals as new parents for the next generation. It is an easy to implement technique, takes less time and practically follows the greedy approach therefore generates good results and helps in most of the cases [18-20].

3) Roulette Wheel Selection

This selection method is also called proportional selection method and uses the fitness proportionate approach. It selects an element randomly from a list on the basis of the fitness function. The elements having a higher fitness value have a greater probability of getting selected, however, even the elements with a lesser fitness value has a non-zero probability of getting selected [18-20].

4) Random Selection

It is a non-probabilistic and the simplest selection method. In this technique, any node is randomly selected from the set of available candidate nodes i.e. a node is selected only because it is readily available and can be conveniently used for further operation without performing any other processing. This method can explore a large search space and provides a wide range of answers. Also, if the technique is allowed to run large number of times, it may generate the best possible solution [18-20].

III. ALGORITHM

Input: A graph $G(V, E)$ with t_{ij} (time taken to traverse) value and S_i (Score) value of each edge and each vertex respectively. A constant $PathListSize$ value which denotes the maximum number of paths that are considered at each level.

Output: A Hamiltonian path with the best possible total collected score such that total travel time is within the stated time bound.

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

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

$Generation(PathList)$

1. $\boldsymbol{Create}\ NewPathList\ , ChildpathList;$
   // Queues storing paths.
   $NewPathList \leftarrow \emptyset;$
   $ChildPathList \leftarrow \emptyset;$
2. $\boldsymbol{for}\ i \leftarrow 0\ to\ |PathList|$

$a.Selection(PathList[i], ChildPathList);$
//$ChildPathList$ will contain children generated from each path in $PathList$.

3. $\boldsymbol{for}\ i \leftarrow 0\ to\ PathListSize$
// Selecting best $PathListSize$ children for next generation.
$a.ChildPath = BestPath(ChildPathList);$
// The path with the maximum total score.
$\boldsymbol{Remove}\ ChildPath\ from\ ChildPathList;$
$\boldsymbol{Insert}\ ChildPath\ to\ NewPathList;$
$b.\boldsymbol{if}\ |ChildPathList| == 0$
$break;$
4. $\boldsymbol{if}\ NewPathList == PathList$
// Terminate if no new child is generated and return the $BestPath$.
$\boldsymbol{return}\ BestPath(PathList);$
5. $\boldsymbol{return}\ Generation(NewPathList);$

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

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

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

$d.\boldsymbol{if} q_{max} \leq q_i$
$q_{max} \leftarrow q_i$
4. $Create\ CandidateList\ ;$
// Array of candidate nodes used for selection.
5. $CandidateList \leftarrow \emptyset;$
6. $\boldsymbol{for}\ i \leftarrow 0\ to\ |V|$
7. $a.\boldsymbol{if}\ i \in P$
$\boldsymbol{continue};$
$b.\boldsymbol{if}\ (q_i \geq \alpha\ q_{max})\&\&\ (t_{parent} + \Delta t_i \leq T_{max})$
// α is the greediness parameter that decides which node should participate in selection process.
$c.\boldsymbol{Insert}\ v_i\ to\ CandidateList\ ;$
8. $\boldsymbol{if}|CandidateList| == 0$
$\boldsymbol{Insert}\ P\ to\ ChildPathList;$
// If no new nodes are added then insert the parent path P.
$\boldsymbol{Return};$
9. $\boldsymbol{for}\ i \leftarrow 0\ to\ PathListSize$
// Generates $PathListSize$ number of children paths from path P.
10. $a.ChildNode \leftarrow Compute(CandidateList)$

// Compute 1 using tournament selection.
// Compute 2 using (μ, λ) selection.
// Compute 3 using roulette wheel selection.
// Compute 4 using random selection.

$\boldsymbol{Remove}\ ChildNode\ from\ CandidateList;$
$\boldsymbol{Insert}\ ChildNode\ to\ Path\ P\ and\ \boldsymbol{Insert}\ Path\ P\ to\ ChildPathList.$
$\boldsymbol{Remove}\ ChildNode\ From\ Path\ P;$
$b.\boldsymbol{if}\ |CandidateList| = 0$
$\boldsymbol{break};$

$BestPath(PathList)$

1. $max \leftarrow 0; bestpath \leftarrow 0;$
2. $\boldsymbol{for}\ i \leftarrow 0\ to\ |PathList|$
a.$\boldsymbol{if}\ \big(Score\big(PathList(i)\big)\big) > maxScore$
// *Score* ($Path\ P$) is the sum of the rewards associated with each node of $ath\ P$.
$b.maxScore \leftarrow Score\big(PathList(i)\big);$
$bestpath \leftarrow i;$
3. $return\ PathList(bestpath);$

The aim of above stated algorithm is to determine a path that connects the source (v_1) and target (v_N) and a subset of vertices that maximize the total collected score and obeys the time limit. This algorithm can be implemented on both complete as well as incomplete graphs. To ensure that in a path, source and target is connected we implement the Dijkstra's algorithm as shown in step 3 of $SEL\_OP()$. To take care of the other constraint that no vertex is visited more than once, the explored nodes are removed from the set of available nodes that forms the candidate list for the selection process. In the $Selection\ ()$, four selection techniques are used and a comparison of their results is presented in the next section. The time complexity of SEL_OP is $O(|V|^3)$.

IV. Experimental Analysis

Our code was implemented in C++ and compiled using CodeBlocks on an Intel Core i5 650 running at 2.20 GHz. SEL_OP has the capability to tackle both complete as well as incomplete graphs and here we report the results for incomplete graphs by running the code on some real data. We considered two instances i.e. a real road network data of 160 and 306 cities of Poland. Each instance is associated with two files i.e. cities.txt and distances.txt. The names of the various cities and their scores are specified in cities.txt. The score of each city is calculated on the basis of the number of inhabitants using the formula $score = inhabitants/10000$. The other file distances.txt represents for each city, its adjacent city and their respective edge lengths. These edges of the graph correspond to the roads of the real map of Poland.

The algorithm SEL_OP was executed using four different selection methods defined in section II (B). In Table 1, the maximum and mean score value achieved by each selection technique for different T_{max} values is presented and in Figure 1, these values are plotted for the first instance with 160 cities. It was observed that roulette wheel selection method performs better than the other three

techniques in most of the cases and helps in obtaining a better total collected score when compared to the other techniques.

It was also seen that when the same algorithm was executed for the 306 cities instance, only the random selection method suffers and rest of the selection methods achieves the same maximum total collected score most of the times as shown in Table 2 and Figure 2. The possible reason for this observation might be the existence of big clusters in the 306 cities instance which is absent in case of the 160 cities instance as shown in Figure 3 (a) and 3 (b). Therefore, the selection techniques have more possible options available to be explored in case of 160 cities instance and do not get stuck within a cluster. So, it can be stated here that roulette wheel selection procedure performs better than the other methods when executed by SEL_OP algorithm.

V. CONCLUSION

In this paper, we introduce a new stochastic greedy heuristic approach (SEL_OP algorithm) for the orienteering problem which can be applied for both complete as well as incomplete graphs. It was implemented using four different selection methods and the results thus obtained were compared on standard benchmarks. We find that roulette wheel selection method performs better than the other three selection techniques and helps in achieving a better total collected score for the specified time budget.

ACKNOWLEDGMENT

The first author would like to acknowledge the financial support by IIT (BHU) in terms of teaching assistantship.

REFERENCES

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

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

[3] M. Hayes, and J. M. Norman, "Dynamic Programming in Orienteering: Route Choice and the Siting of Controls", Journal of the Operational Research Society, vol. 35 (9), pp. 791-796, 1984.

[4] T. Tsiligirides, "Heuristic methods applied to orienteering", Journal of the Operational Research Society, vol. 35, pp. 797–809, 1984.

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

[6] B. Golden, L. Levy, and R. Vohra, "The orienteering problem", Naval Research Logistics, vol. 34, pp. 307–318, 1987.

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

[8] Q. Wang, X. Sun, B. L. Golden, and J. Jia, "Using artificial neural networks to solve the orienteering problem", Annals of Operations Research, vol. 61, pp. 111–120, 1995.

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

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

[11] V. Campos, R. Marti, J. Sanchez-Oro, and A. Duarte, "GRASP with Path Relinking for the Orienteering Problem", Journal of the Operational Research Society (2013), pp. 1–14, 2013.

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

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

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

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

[16] M. Tasgetiren, "A genetic algorithm with an adaptive penalty function for the orienteering problem", Journal of Economic and Social Research, vol. 4(2), pp. 1–26, 2001.

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

[18] T. Back, "Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms", Proceedings of the First IEEE Conference on Evolutionary Computation, 1994, pp. 57-62.

[19] N. M. Razali, and J. Geraghty, "Genetic Algorithm Performance with Different Selection Strategies in Solving TSP", Proceedings of the World Congress on Engineering 2011, vol II WCE 2011, July 6 - 8, 2011, London, U.K. pp. 1134-1139.

[20] R.Sivaraj, and T.Ravichandran, "A Review of Selection Methods in Genetic Algorithm", International Journal of Engineering Science and Technology (IJEST), vol. 3(5), pp. 3792-3797, 2011.

TABLE I. Comparison of the mean and maximum value of the total collected score obtained by SEL_OP when executed with four different selection procedures for 160 cities.

T_{max}	(μ,λ)		Random		Tournament		Roulette Wheel	
	Max	Mean	Max	Mean	Max	Mean	Max	Mean
500	49	49	49	48	49	48	49	48
750	67	67	67	64	67	66	67	65
1000	83	83	88	77	88	84	88	79
1250	103	103	103	99	103	101	102	100
1500	118	118	116	114	119	116	116	114
1750	135	135	130	128	135	128	130	128
2000	145	145	145	143	145	143	148	143
2250	157	157	160	154	157	155	162	155
2500	179	179	184	175	179	176	187	174
2750	190	190	203	193	191	189	203	193
3000	206	206	214	205	206	202	214	206
3250	234	234	233	223	234	228	233	224
3500	248	248	249	242	249	245	251	244
3750	255	255	261	255	257	255	263	256
4000	269	269	271	264	272	268	270	267

TABLE II. Comparison of the mean and maximum value of the total collected score obtained by SEL_OP when executed with four different selection procedures for 306 cities.

T_{max}	(μ,λ)		Random		Tournament		Roulette Wheel	
	Max	Mean	Max	Mean	Max	Mean	Max	Mean
500	127	121	127	117	127	121	127	118
750	155	149	155	147	155	148	155	147
1000	178	173	177	168	178	172	178	169
1250	211	173	208	190	211	172	211	193
1500	236	213	234	218	236	219	236	221
1750	263	241	263	253	263	254	263	251
2000	282	280	279	273	282	277	282	273
2250	307	300	299	291	307	292	307	281
2500	325	323	323	307	325	309	325	306
2750	349	336	333	325	349	327	349	329
3000	366	355	366	346	366	345	366	344
3250	396	339	386	366	396	371	396	369
3500	420	394	414	384	420	393	420	390
3750	435	408	428	410	435	411	435	400
4000	459	430	448	429	459	437	459	440

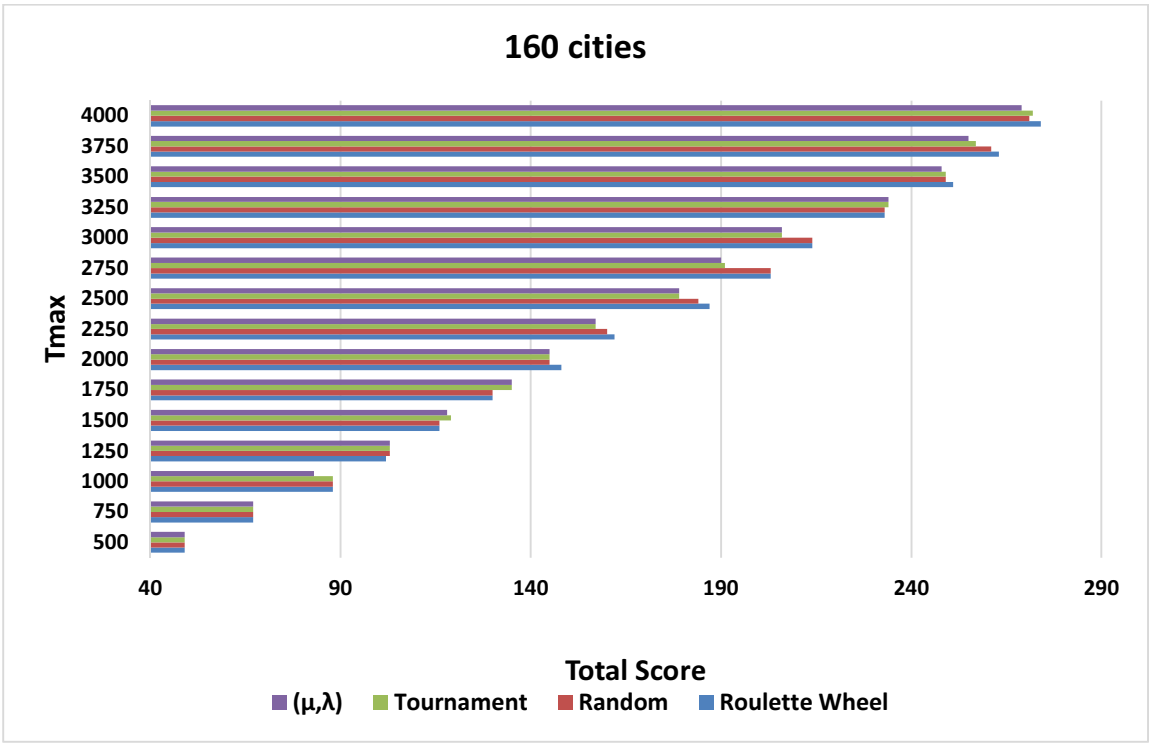


Figure 1. Comparison of the maximum value of the total collected score obtained by four different selection methods for different T_{max} values (160 cities).

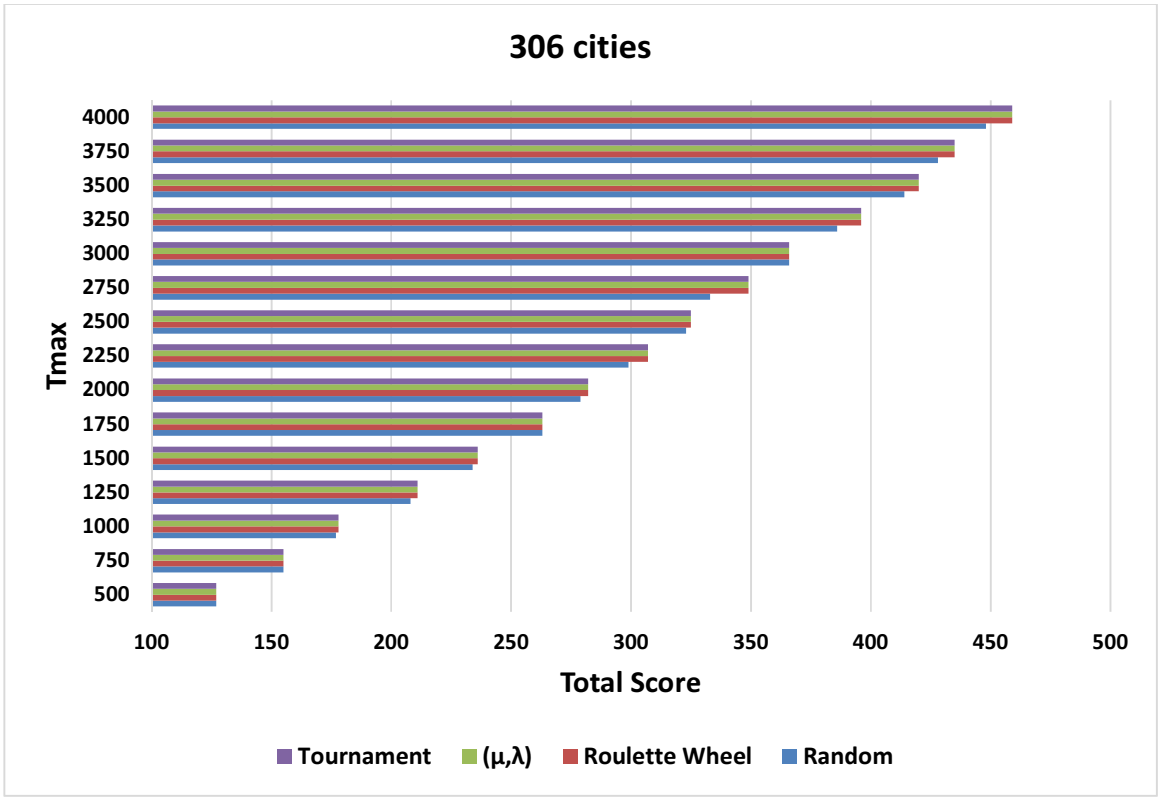
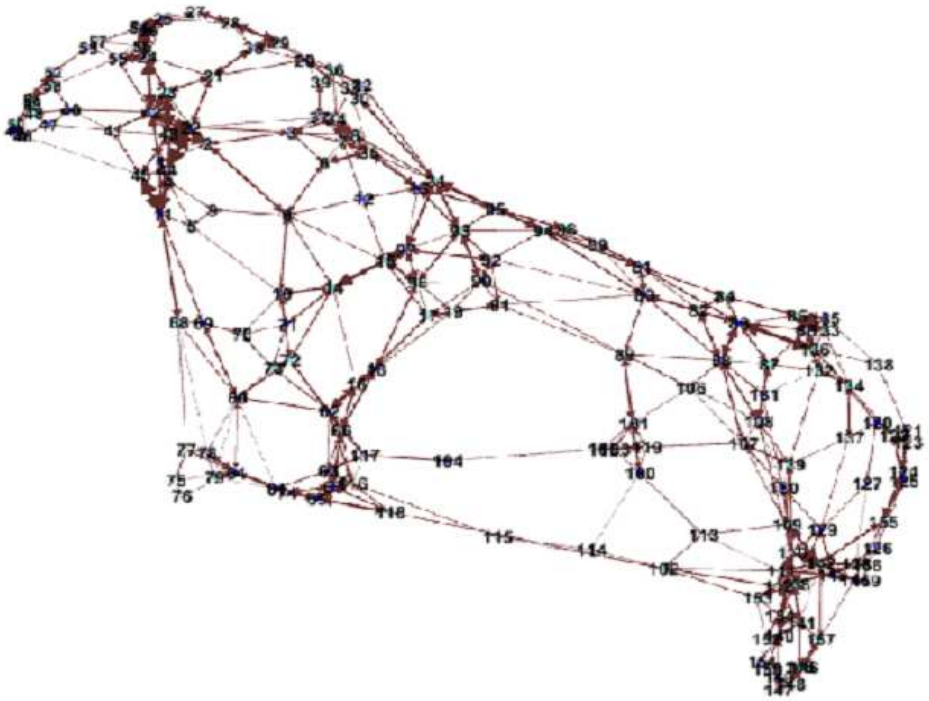
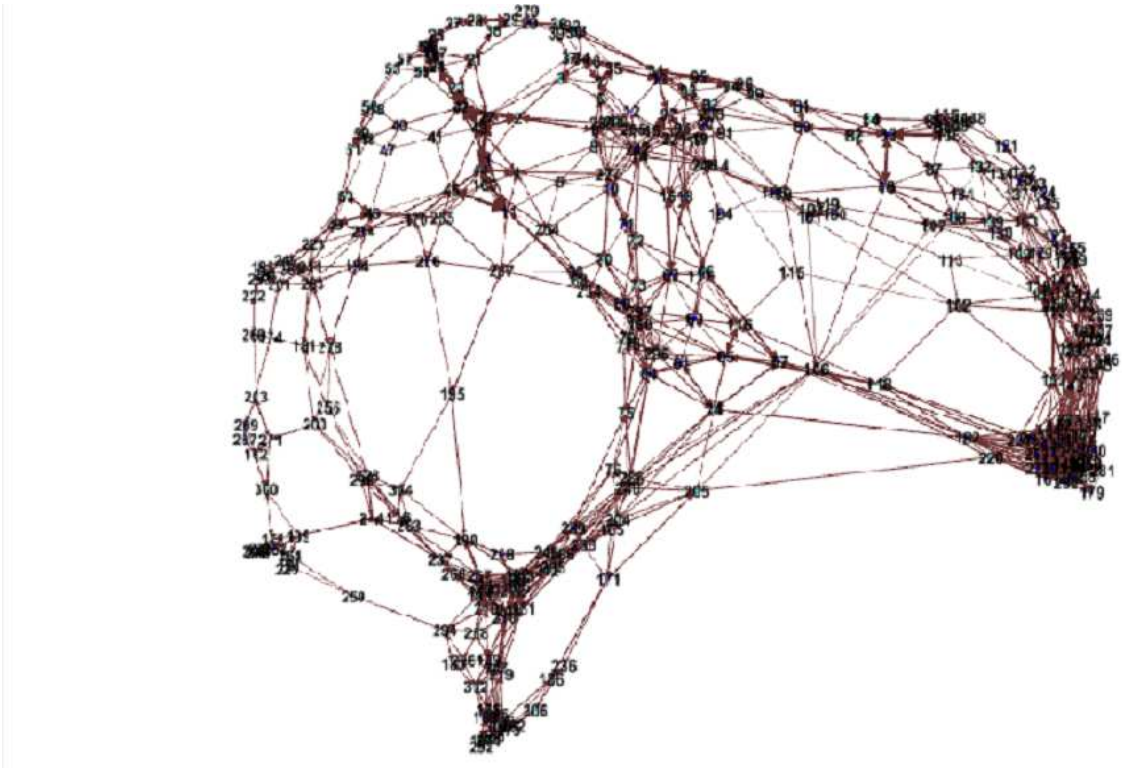


Figure 2. Comparison of the maximum value of the total collected score obtained by four different selection methods for different T_{max} values (306 cities).



(a)



(b)

Figure 3. Graph for (a) 160 cities and (b) 306 cities instance.