TSP and Two-Approximation Algorithm

Muhammad Abbas

MS(Computer Science)-MS110400080-Fall-2012, Virtual University of Pakistan
ms110400080@vu.edu.pk

Abstract— **the purpose of this work to find out the relative optimize solution for Travelling Sale Man Problem (TSP). Since the TSP belong to the NP-hard category of problems and there is no an algorithm exits that compute the TSP in polynomial time. If a problem has been declared as HARD problem then we try to find out some approximation solutions with imposing the restrictions on the actual HARD problem. In the first part of the work, we described some basic terms and definitions that are frequently used in the whole paper. In the middle part, history of TSP and three versions of TSP have been explained. At the end of this paper, 2-approximation algorithm for restricted version of TSP (Matrix TSP) is presented. Improving approximation for TSP has been delighted area of research for the computer scientists.**

Keywords— **TSP, Ham-Cycle (Hamiltonian Cycle), MST (Minimum Spanning Tree), Triangular Inequality, Matrix TSP, Euclidean TSP, DFS, OPT (Optimal)**

1) INTRODUCTION

In this section of the paper, we presented the overview of some important terms that are often being used in the various parts of the paper. This section establishes the grounds for the actual work of the paper.

1.1 TSP [Travelling Sale Man Problem]: This problem was first studied formally in 1930 and became the hotcake in the optimization problems. In combinatorial optimization studies TSP is an NP-hard problem in operations research and theoretical computer science. TSP is known as benchmark in the studies of optimization category.

TSP is: to find out shortest possible rout in the given list of cities/locations with their cost pairs (distance between cities) in such a way that starting from origin and visiting every city exactly once and back to starting point (city).

There are several applications of TSP in our practical life, such that:

- Circuit designing
- Manufacturing of microchips
- Planning and logistics
- Sequencing of DNA
- etc

1.2 Graph: In mathematic, the graph is set of objects which are connected together by links and these links are called edges and linked objects are knows as vertices. More formally, a graph G with vertices V and edges E is represented as:

$$G = (V, E)$$

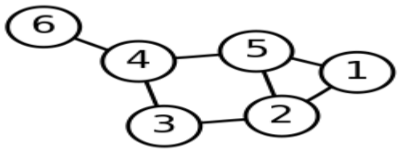

Fig. 1 A labelled graph with 6 vertices and 7 edges

1.3 Connected Graph: Let G be a graph such that G=(V,E) and there are where 'V' are the vertices or nodes and 'E' are the edges. A graph is connected if and only if there is an edge between the vertices. In simpers word, a graph is connected if there is a path from any vertex to the any other vertex in the G. The following are the some examples of connected graphs:

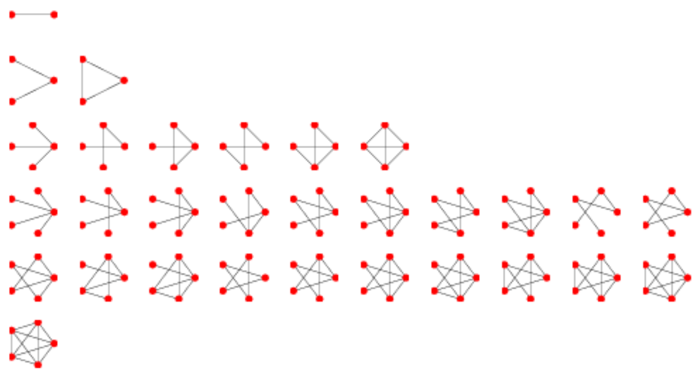

Fig. 2 Connected Graphs [mathworld.wolfram.com]

1.4 Complete Graph: A complete graph is a graph in which every vertex connected with every other vertex. Further, the edges in this graph support the triangular inequality.

A complete graph is denoted by Kn, where 'n' is the number of vertices. The number of undirected edges in complete graph can be calculated by the following formula:

$$\frac{n(n-1)}{2}$$

Following are the some examples of complete graphs:

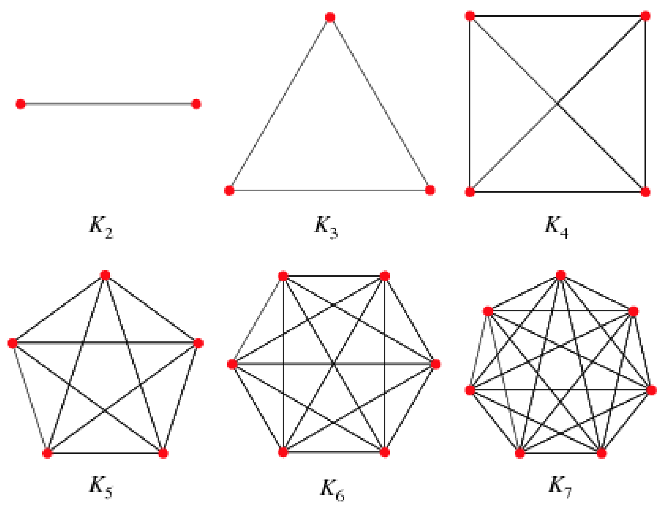

Fig. 4 Complete Graphs

1.5 Undirected Graph: When nodes/vertices of graph are connected with undirected edges (edge without arrow) then graph is called undirected graph. Simply "graph" is taken as undirected graph.

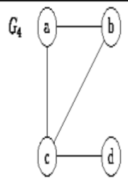

Fig. 5 Undirected graph with 4 vertices

1.6 Acyclic Graph: If there is no cycle in the underlying graph then this graph is called acyclic graph.

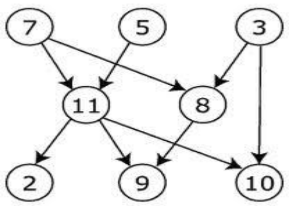

Fig. 6 Acyclic Graph

1.7 Hamiltonian's graph: In this paper, Hamiltonian graph and tour will be used interchangeably.

A graph "G" is said to be Hamiltonian if every vertex will traversed/visited exactly once.

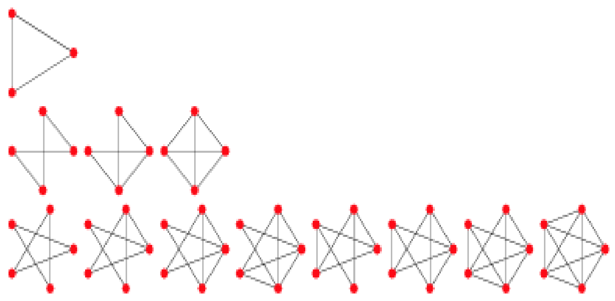

Fig. 7 Images of Hamiltonian graphs

1.8 Planner Graph: A graph that is embedded in a plane in such a way that its edges do not cross the vertices.

1.9 Spanning Tree: Spanning tree in a connected graph consist of all the vertices in such a way that they make no cycle.

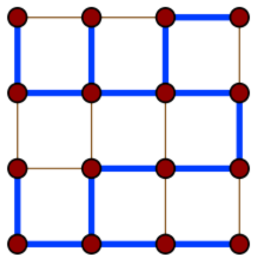

Fig. 8 A spaning tree

1.10 Minimum Spanning Tree (MST): If a graph has more than spanning trees then the MST will be a spanning tree which has minimum weight from start vertex to end vertex.

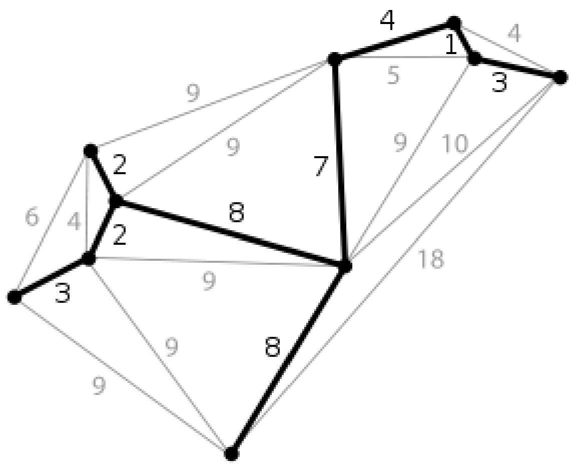

Fig. 9 MST with weighted edge

2) History of TSP

TSP problem is extremely rich problem. It is very well studied problem in the history of NP complete problems because it has been favourite filed of study for computer scientists. This problem has also practical implementation in our daily life. This problem has very long and interesting history. This was the first problem for which Jack Adams argued that this problem has not a polynomial time algorithm. He conjectured in some sense that:

$$P \neq NP$$

Since the theory of NP completeness was not developed at time. While working on this problem, Jack looked that there is some thing difficult to solve this problem in polynomial time and thereof he argued on the base of his knowledge that this problem has no polynomial time algorithm.

3) Why TSP is hard?

Let G be a graph with n vertices and let's say it is a complete graph i.e every vertex is connected to every other vertex and now every edge has a cost (weight) associated with it which is a positive integer. We want to find out the cheapest path (Hamiltonian) from all possible Hamiltonian paths (cycles) present in the given graph G to visit all the vertices in the graph.

There is a question that how many possible tours (visits) are there in the given graph, the answer of this proved by combinatorial mathematics that there are (n-1)! visits. We can order the vertices in acyclic arrangement in (n-1)! ways. Since it is a complete graph and there will be (n-1)! tours/visits in the graph.

If we try to enumerate/visit all the possible tours and try to calculate their costs, keeping track of smallest cost of the tour, the problem is that there will be too many tours [(n-1)!] and it will take too much time. We know that the factorial function is not a polynomial time function because it grows very fast as the value of 'n' (number of vertices) increases.

In one way to think about NP completeness is perhaps to see the space we are searching is very large enough and however this is not the fact. This point can be elaborated by the help of an other problem about the satisfiability.

Suppose, there is a Boolean formula on 'n' variables and we are asked to argue that the given formula is satisfiable or not?. The realization is that there are 2n assignments of the given Boolean formula over 'n' variables. Therefore, there are exponentially many assignments. The question is that is there a single assignment that make the whole class true? Infact we are looking for one particular assignment that satisfy the whole family of that class. Since the space is exponential and if we look one by one assignment, that would be not a polynomial time algorithm. In worst case this algorithm take exponential time. Therefore, there is not possible to come with an algorithm that enumerate these assignments in polynomial time algorithm.

Therefore, in the above mentioned two problems [TSP & Boolean assignments] there is commonality that the search space is exponential.

The common thing in all NP complete problems is that there is exponential time search space.

There is a question; is it true that every problem that has exponential search space is NP complete?

The answer is no. We have many problems that have exponential search space and we can enumerate these problems in polynomial time. Minimum Spanning Tree (MST) can have exponential search space. If the given graph is complete graph then by using clever combinatorial methods that the number of spanning trees in complete graph is n(n-2). There are too many spanning trees in the underlying complete graph. MST is the one possible cheapest spanning tree out of n(n-2) spanning trees. Therefore, the search space is exponential in this case. The exhaustive search uses the enumeration of all spanning trees and find out the MST and this will result into exponential time because there are exponential number of spanning trees in the complete graph.

Suppose a connected graph and undirected graph G=(V,E), where 'V' are the vertices and 'E' are the edges of the graph 'G'. For this graph 'G' a spanning tree is an acyclic subset of edges $S \subseteq E$ that joins all the vertices together. Let each edge (u,v) in the underlying graph G, there is a cost 'w' associated with them which is denoted as:

$$w(u,v) \qquad : \qquad w \geq 0$$

The cost of spaning tree 'S' can be calculated as follows:

$$w(S) = \sum_{(u,v)\in S} w(u,v)$$

But Kruscal's algorithm and Prim's algorithm calculate the minimum spaning tree in polynomial time in connected weighted graph. Although the search space in these cases is exponential, this fact does not mean that the problem is NP-complete. In shortest path algorithms, the search space may also be exponential but there are efficient algorithms like Dijikstra's algorithm that find shortest path in polynomial time.

Such algorithms circumvent the whole search space in some tricky way and use the structure of the problem to find the solution in acceptable polynomial time.

Since the TSP is declared NP-hard problem and there is no polynomial time algorithm for its solution. Due to this fact the computer scientist then start looking the alternation ways such that approximation solutions of these problems.

4) Variants of TSP

To find the approximation solutions, we can consider the following three versions of TSP:

- *Usual TSP:* In this version, we are given an arbitrary graph G with weights on its edges and we are asked to find the cheapest Hamiltonian cycle in the given graph G. Weight function does not satisfies any constraint other than the fact the weights are non-negative. This version of TSP belong to NP-complete class.
- *The Metric TSP:* This restricted version of TSP shows that; given a complete graph that satisfies the following condition:

$$w(a,c) \leq w(a,b) + w(b,c)$$

This condition is called triangular inequality. Triangular inequality says that the cost of going from a→c is less or equal to sum of costs from a→b and then b→c. This is always true for triples a,b,c. This restriction of Metric TSP is very natural and always be satisfiable on planner situation. Most of the time Metric TSP is satisfiable in practical situations while travelling via some air line and choosing the cheapest rout from origin to destination. So the metric TSP is very natural restriction and many problems fall in this category. Thus the Metric inequality is:

> *"The underlying graph is a complete graph and edge weights of the graphs satisfy the triangular inequality"*

- *The Euclidean TSP:* Euclidean TSP is even simpler, some one may be asked to find out the cheapest path from the given 'n' number of points on the plane. The Euclidean TSP is more restricted; in this version, satisfiability is guaranteed because all the points are to be considered in Euclidean plan and this plan always satisfy the triangular inequality constraint. This restricted version of TSP also belong to NP-complete.

Since the above versions are the derivations of TSP and are NP-complete as that of their origin. In this situation, our focus is not to find the solution that equate the two classes P and NP or to find the solution of the known NP hard problem into polynomial time but to find such algorithms that solve NP hard problems in relatively less time. These algorithms are known as ***"approximation algorithms".***

In the problem of TSP, we can not find the cheapest path for tour because it is NP hard problem. But if someone, propose a path that is cheap but not cheapest then we have no other choice to adopt this cheap path among the exponentially large space. This solution is not complete but someone can rely on it to circumvence the arbitrary large search space. This is the main idea behind the approximation algorithms.

Formally the approximation algorithm is defined as:

"A fast (polynomial time) algorithms that does not find the cheapest path/tour in exponential search space but find the relatively cheap enough tour/path"

The idea behind the approximation algorithms is to apply more natural restrictions on the original problem that is in NP class so that the restricted version may be solved in polynomial time.

Here is our purpose is to deal with hard problems. Question is to solve our problem that falls in the NP Complete category. First of all we will try to find out polynomial time algorithm to tackle this problem and since we know that it is NP complete; we have to find out restricted version of the underlying problem. We have to take this restricted version and impose more restrictions on it so that it may fall in class P. The restrictions must be so natural that their implementation in real life should be obvious. This strategy will be implemented while starting from general version to restricted and more restricted and so on until the optimal (polynomial time) solution of the underlying problem can be found. This is the one approach.

Another approach is; to solve a problem that is in NP, pickup the problem 'P' and we know that problem belong to NP class. We want to find polynomial time algorithm for this problem. For this purpose, we have to settle the exponential time algorithm of underlying problem. As the exponential time algorithm take a lot of time and we want to device such algorithm in this category that may be relatively more efficient. In this approach, we focus on to bring the exponent down to reasonable choice.

The third approach is that we can apply on NP hard problems. This approach is not very popular with theoretical computer scientists but has been very fruitful. Since the problem is NP complete and to solve we have to go though the whole search space. In this approach we use some sort of heuristical idea to cut down on the search space. In our practical life, these instances of NP complete problems can solve in reasonably comparable time as we have enough powerful computers. This approach is mostly used in SAT Solver (Satisfibility Solver). Although the search space is exponential but we have powerful enough computers with sufficient resources, therefore many computer scientists are using these heuristics to solve the NP problems with thousands of variables.

The final approach to deal with HARD problem is "Approximation Algorithms". The idea of this approach is find a good enough solution of the underlying problem.

The approximation algorithm for TSP is:

"Find out the polynomial time (not too much time) but not to find out the cheapest tour instead to try to find out cheap enough tour"

But the difference between the approximation algorithm and Heuristic is given below:

In Heuristic, we have no guaranty that the solution of the underlying problem is cheapest or cheaper enough.

Approximation approach gives us the bonafied guaranty that the outputted solution is close to optimal one and cheap enough.

The general TSP can not guaranty in any degree of approximation. We can not have a polynomial time algorithm under the hypothesis "$P \neq NP$" that give us the guaranty to 100 times the optimal tour. We can only convert this number to any polynomial time computable function. Using metric TSP, it is easy to designing 2-approximation algorithm.

We can also design 1.5-approximation algorithm using Krestof heuristic which guaranty a tour in metric TSP in 50% optimal.

5) 2-Approximation Algorithm

Let "G" is a graph with cheapest tour having size "t". Suppose "A" is an algorithm that gives the tour having size "αt" then we say it $\alpha - approximation$ algorithm.

In this scenario, we can say that $2 - approximation$ algorithm will give us answer that is at most twice the *"t"*. Similarly 1.5-appriximation and 1.01-approximation algorithms will give answer in 1.5 x *t* and 1.01x *t* (at most) respectively.

Out goal is to prove the following theorem:
"There is a 2-approximation algorithm for the metric TSP problem"

We are given a graph "G" which satisfies the triangular inequality. The approximation algorithm has three main step:

- Calculating MST: First we have to calculate the MST in this graph which can be done in polynomial time with the help of Kruscal's or Prim's algorithms because both these algorithms can find out MST in polynomial time. To calculate the Hamiltonian cycle, we juts calculate MST in this step.
- Applying DFS: In second step, we take this MST and visit all the vertices of this MST in the DFS technique, starting from any point. Writing down the visited vertices. In this way we start from a root node, visit all the vertices and come back to the starting point. Heuristically, this seems to be a nice way to tour the whole graph in cheapest way because MST cheapest way to connect the underlying graph.
- Removing the Duplicates: The only problem in this way (DFS) of traversing the graph is that we may have to visit one vertex more than once. Hence there will not be Hamiltonian cycle in the underlying graph. The simplest way to overcome this problem is to keep the record of visited vertices as mentioned above, if there is duplicated visited vertex, throw it out. In this way, the resultant MST will be Hamiltonian cycle.

The output of the above description of the algorithm is Hamiltonian Cycle in the graph. The observations of this algorithm are:

- It runs in polynomial time because MST can be calculated in polynomial time and DFS also take polynomial time. Removal of duplicates in the DFS is not difficult and can be done in polynomial time.
- It results the Hamiltonian cycle because steps of the algorithm satisfy the conditions of the Hamiltonian cycle [MST connects ***ALL*** the vertices, DFS traverse ***ALL*** the vertices and removal of duplicate vertices ensures the single occurrence of ***ALL*** the vertices].

The critical and important part of the algorithm comes into action. This part contains the analysis for the approximation algorithm. This analysis contains the fact that how does the Hamiltonian cycles be the cheapest cycle in the graph. In the underlying graphs we find out a Hamiltonian cycle and how we can compare this output result with the cheapest Ham-Cycle which exists somewhere in the graph and we do not know about it. This seems to be a contradictory thing as we are comparing our result with some thing that we can not find out. Actually, we do not need to find it out but we have to compare the properties of these two objects which come up with an approximation guarantee. For this analysis and observation, we have to prove the following facts:

"The cost of the MST is less than or equal to the cost of the cheapest tour"

Proof:

- We have a path with special kind of spanning tree (one edge is deleted from the Ham-cycle). According to the definition of MST, this path must have more cost that the MST.
- In a graph with the property of satisfying triangular inequality, the cost of the tour resulted by DFS will be at least twice the actual cost of the tree.

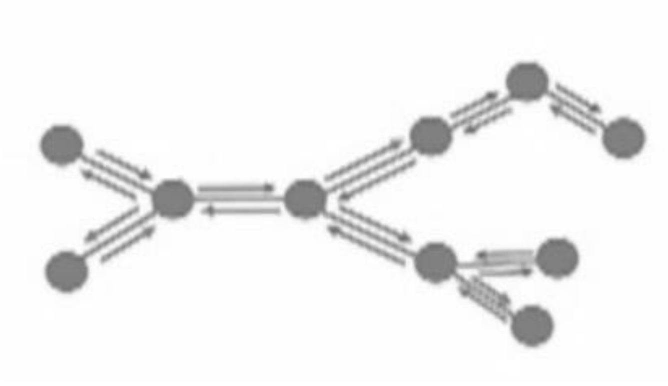

Fig. 10 DFS with duplicate edges

- The above figure shows that it will not result in Ham-cycle because some vertices are traversed more than once which is contradiction to the Ham-cycle property. The solution of this problem is to delete the every vertex which is visited before. This fig. 10 after applying this shortcut will look like as:

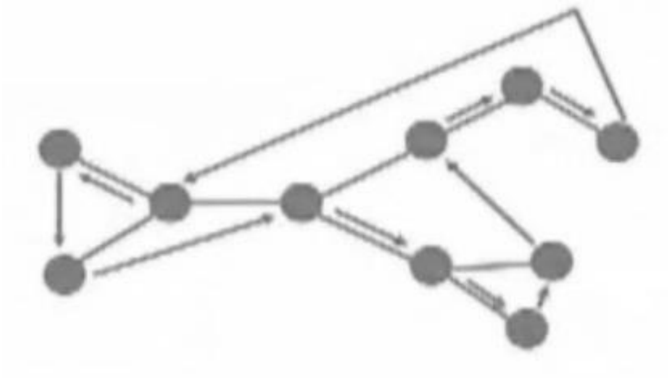

Fig. 11 DFS after deleting duplicate edges

1. The important point is that if the underlying graph satisfies the triangular inequality the applying shortcut does not increase its cost. Therefore, this Ham-cycle does not have the cost more than the MST which is resulted with DFS. It may be twice at most to the Ham-path in the graph. If there are $v_1, v_2, v_3, \ldots v_k$ vertices by applying short cut we will have v_1, v_k . According to triangular inequality,:

$w(v_1, v_k) \leq w(v_1, v_2) + ...w(v_k - 1, v_k)$

This fact shows that with the satisfaction of the above property, this graph remains at most 2-aproximation.

- If we applying the above property on general or complete graph, the consequence may be in the form of non-Hamiltonian graph or greater cost than twice of the MST. Matrix TSP is well defined under this property, therefore, it can result into 2-aproximation algorithm.

The above discussion can be summarized as follows:

1. Input a complete graph with positive edge weights that satisfy the triangular inequality.
2. Compute the MST of the given graph with the guarantee that $\cos t \leq OPT$ i.e cost of MST is less or equal to the optimal cost of the tour(Ham-cycle).
3. Apply DFS and output the encountered vertices during the DFS. The result of DFS will be:
   $COST\ with\ repeated\ tour \leq 2MST \leq 2OPT$
4. Obtaining result of all the visited vertices after removal of duplicate vertices which cost will always be $Cost \leq 2OPT$

6) Conclusions

In the above mentioned work, we presented the overview of most studies problem in the field of computation and graph theory. The underlying problem was Travelling Salesman Problem (TSP). The heart of this problem is Hamiltonian Cycle. TSP remains always be an open area of research to find out the optimal Ham-path in an arbitrary large domain of vertices/points. Generally when a problem belong to NP-complete family of domain, then we often do not exert force and time to find out its polynomial time solution because it will be the wastage of time and resources. In this situation, we cut down the domain of the problem and looks toward the restricted versions of the problem to find out the cheap enough solution with the help of approximation algorithm. Here is the same case with TSP. Generally, TSP belongs to NP-complete family of problems and we focus on restricted version of this problem to find out possible optimal solution. For the TSP, we considered Matrix TSP and find out its 2-approximation algorithm. In approximation algorithms, improving the approximation is a big challenge for the researcher and computer scientists. There is 1.5-approximation algorithm has also been worked out but improving it up to 1.499-approximation is still an open challenge in computer science.

Acknowledgment

I would like to thank those who consulted with me to explain the concepts of this article and most probably the great personalities, who are guiding lamp for me all the way. The naming of list of these great persons is referred below. Further, I want to pay my gratitude toward my teachers, who guide me all the way for this work.

References

[1] ^{1}Khaled Elbassioni, ^{2}Aleksei V. Fishin, ^{3}Rene Sitters, ^{1}Max-Planck-Institute fur Informatik, Germany, ^{2}University of Liverpool, UK, Department of Mathematics and Computer Science, ^{3}Eindhoven University of Technology, Netherlands, "Approximation Algorithms for the Euclidean Traveling Salesman Problem with Discrete and Continuous Neighborhoods," in proceedings of *ICALP 2005 and ISAAC 2006*.

[2] Kamal Jain, College of Computing, Georgia Institute of Technology, "A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem", March-1998.

[3] ^{1}Aaron Archer, ^{2}Muhammad Hossein Bateni, ^{1}Mohammad Taghi Hajiaghayi, ^{1}Howard Karloof, ^{1}AT & T Labs-Research, 180 Park Avenue, Florham Park, NJ 07932, ^{2}Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540, "Improved Approximation Algorithms for PRIZE-COLLECTING STEINER TREE AND TSP".

[4] Piotr Berman, Marek Karpinski, Dept. of Computer Science and Engineering, The Pennsylvania State University, Dept. of Computer Science, University of Bonn, "8/7-Approximation Algorithm for (1,2)-TSP (Extended Version)" in Proc. 17th ACM-SAIM SODA (2006).

[5] ^{1}Gautier Stauffer, ^{2}Guillaume Massonnet, ^{2}Christophe Raphine, ^{2}Jean-Philippe Gayon, ^{1}Bordeaux Institute of Mathematics, France, ^{2}Laboratoire G-SCOP, Grenoble INP, France, "A Simple and fast 2-approximation algorithm for the one-warehouse multi-retailer problem".

[6] ^{1}Rainer E. Burkard, ^{2}Vladimir G. Deineko, ^{3}Rene Van Dal, ^{4}Jack A. A. Van Veen, ^{1}Derhard J. Woeginger, ^{1}Institute of Fur Mathematics, TU Graz, Austria, ^{2}Warwick Business school, The University of Warwick, Coventry, UK, ^{3}Chalmers Tekniska Hogskola, Institution for Datavetenskap, Sweden, ^{4}Nijerrode University, The Netherlands Schools of Business, the Netherlands, "Well-Solvable Special Cases of the Traveling Salesman Problem: A Survey" Vol. 40, No.3, pp. 496-546, September 1998.

[7] Valerie King, University of Victoria, Canada, *Minimum Spanning Tree Verification in Linear Time Complexity,* Lecture handout-2009.

[8] Dr. Sohail Aslam, Dept. of Computer Science, Virtual University of Pakistan, *Design and Analysis of Algorithms,* Lecture Handouts, January-2004.

[9] Dr. Sarmad Abbasi, Dept. of Computer Science, Virtual University of Pakistan, *Theory of Computation,* Lecture Handouts.

[10] Gregory Gutin and Alexei Zverovitch, Department of Computer Science, Royal Hollway, University of London, *Exponential Neighborhoods and Domination Analysis for the TSP*, The Travelling Salesman Problem and its Vatiants, 223-256 (2002).