A NETWORK BRANCH AND BOUND APPROACH FOR THE TRAVELING SALESMAN MODEL

Elias Munapo

Graduate School of Business and Leadership, University of KwaZulu-Natal

Accepted: June 2012

Abstract

This paper presents a network branch and bound approach for solving the traveling salesman problem. The problem is broken into sub-problems, each of which is solved as a minimum spanning tree model. This is easier to solve than either the linear programming-based or assignment models.

Key words: NP hard, traveling salesman problem, spanning tree, branch and bound method

JEL: C610

1 Introduction

The traveling salesman problem (TSP) is one of the NP hard problems that are of concern to researchers. A salesman visits each of the given N cities or towns in such a way that each city is visited once, the total distance travelled is minimal and must return to his or her original base (Berman & Karpinski, 2006; Gutin & Punnen, 2006). Up to now we have been unaware of any effective and exact method for solving this problem.

There are several simple variations of the TSP that have originated from various real-life problems. These include the MAX TSP, bottleneck TSP, TSP with multiple visits (TSPM) and clustered TSP. Let $G=(V,E)$ be a graph (directed or undirected) and F be the family of all Hamiltonian cycles (tours) in G. For each edge $e \in E$ a cost (weight) c_e is prescribed. The matrix $C=(c_{ij})_{N\times N}$ is the distance or weight matrix, where the $(i,j)^{th}$ entry c_{ij} corresponds to the cost of the edge joining node i to node j in G. Let the node set be $V=\{1,2,...,N\}$. In other words, the TSP is to find a tour (Hamiltonian cycle) in G such that the sum of the costs of the edges of the tour is as low as possible.

In the MAX TSP, the objective is to find a tour in G where the total costs of edges of the tour is a maximum. This is solved as a TSP by replacing each edge costs with its additive inverse. In bottleneck TSP, the objective is to find a tour in G such that the highest cost of edges in the tours is as low as possible. A bottleneck TSP can be formulated as a TSP with exponentially high edge costs. In the TSPM we find a routing for travelling salesman who starts at a given node of G, visits each node at least once and comes back to the starting node in such a way that the total distance travelled is minimized. The TSPM is transformed into a TSP by replacing the edge costs with the shortest path distances in G. The node set of G is partitioned into clusters. Then the objective of clustered TSP is to find a least-cost tour in G subject to the constraint that cities within the same cluster must be visited consecutively. This problem can be transformed into a TSP by adding a large cost (LC) to the cost of each inter-cluster edge.

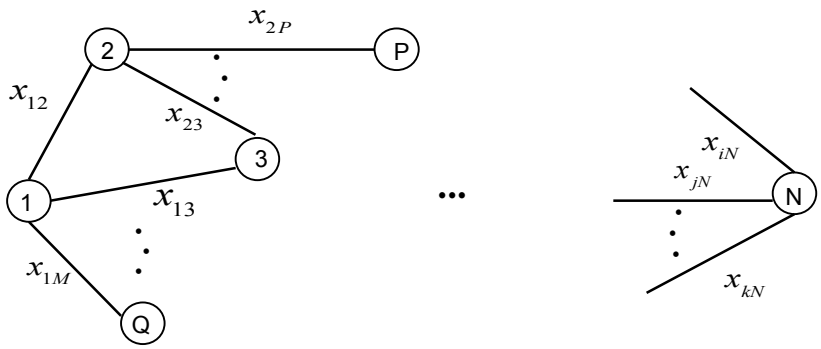
TSP and its variations have a very wide application and span several areas of knowledge which include operations research, computer science, genetics, electronics and logistics. TSP is used in machine sequencing and scheduling. The machine sequencing problem is to find an order in which the jobs are to be processed such that the total machine set up costs are minimized. In some manufacturing systems such as cellular processes, certain products require similar processing and the problem is to group them so as to achieve efficiency and cost reductions. Let $G=(V,A\cup E)$ be a mixed graph where elements of A are arcs (directed) and elements E are edges

(undirected), $A^1 \subset A$ and $E^1 \subset B$. The arc routing problem is to find a minimum cost closed walk on G containing all arcs in A^1 and all edges in E^1. The TSP model is also used in creating matrices with certain desired structures which is applied in statistical data analysis and theory of group discussion. Many other applications of TSP models are contained in literature.

2 The TSP model

The objective is to move from node 1 and back in such a way that every node is visited once and the total distance is minimized. It is also assumed that two arcs emanate from each of the nodes.

Figure 1
TSP model



The objective is to move from node 1 and back in such a way that every node is visited exactly once and the total distance minimized. It is also assumed that each node has at least two arcs.

$$TSP_o = \text{Min } Z = \sum_i \sum_j c_{ij} x_{ij}$$

Such that $\sum_{i=1}^{N} x_{ij} = 1$, (for $j = 1,2,...,N$)

$$\sum_{j=1}^{N} x_{ij} = 1, \text{ (for } i = 1,2,...,N)$$

$u_i - u_j + N x_{ij} \leq N - 1$ (for $i \neq j; i = 1,2,...,N; j = 1,2,...,N$)

All $u_j \geq 0$ and x_{ij} is a binary variable.

Where c_{ij} is the distance from city *i* to city *j* for $i \neq j$.

Letting $c_{ii} = M$, where M is very large relative to the actual distances between the cities ensures that the movement to city *i* immediately after leaving city *i* is not possible.

3 Solving the TSP

There are two main categories of TSP which are symmetric and asymmetric. If the distance between two nodes in the TSP network is the same in both directions, then the TSP is called symmetric, otherwise it is referred to as asymmetric. We are not aware of any efficient exact method for solving the traveling salesman problem. There are several heuristics (see Berman & Kapinski, 2006; Gutin & Punnen, 2006; Wolsey, 1980; Winston, 2004) and for exact approaches (see Gutin & Punnen, 2006; Nadef, 2002; Padberg & Rinald 1991; Winston, 2004) available in the literature.

3.1 Heuristics

These are approximation methods that quickly lead to good solutions which are not necessarily optimal. There are several classes of heuristics, some of which are:

- constructive heuristics;
- iterative improvement;
- randomized improvement.

More information on each class of these heuristics can be found in writing by Berman & Kapinski (2006), Gutin & Punnen (2006) or Wolsey (1980). Even though some of these approximating methods have improved we cannot be sure how far these approximated solutions are from the optimal ones. For example when all the towns in very large countries such as Russia, USA or China are visited, the difference between the exact and

approximate solutions may amount to millions of dollars.

3.2 Exact approaches

The most obvious exact approach is to try all the possible routes to decide which one is the best. The worst cost for this approach is of factor $O(n!)$ and is not practical for a large number of towns. Other exact approaches include

- Linear programming (LP) based branch and cut methods (see Gutin & Punnen, 2006; Karlof, 2005; Mitchell, 2001; Nemhauser & Wolsey, 1988);
- Branch and bound methods with assignment sub-problems (see Winston, 2004);
- Dynamic programming techniques (see Gutin & Punnen, 2006).

Efforts to improve these exact approaches have been unsuccessful. Large amounts of computational times are required. The computational times for these exact approaches are not practical since decisions have to be made urgently in war or disaster situations.

4 Network branch and bound approach

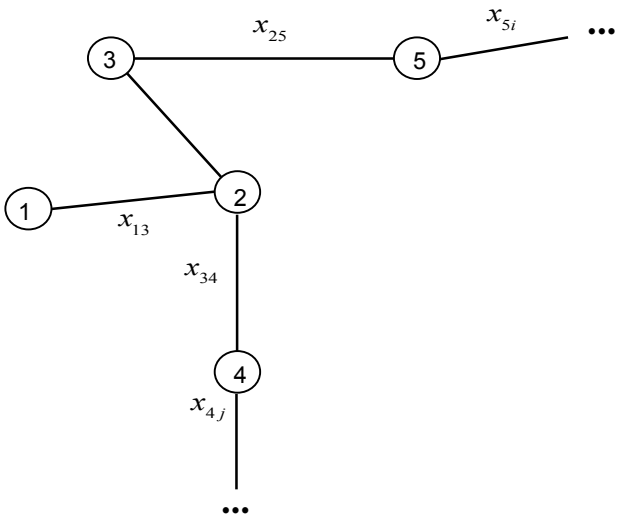
In this paper an exact branch and bound approach is proposed. The network branch and bound uses minimum spanning trees (MST) as sub-problems. The minimum spanning tree can be solved efficiently by the available approaches. The MST model is easier to solve than either the LP-based or assignment sub-problems that are currently used. In this paper the following definitions apply.

4.1 Definitions

A **tree** is a set of nodes that are connected by arcs to form a single structure.

A leaf is a node that has no children and is connected to a tree.

Figure 2
An example of a tree



Node 1 is an example of a leaf. Clearly it has no children.

For a network with N nodes, **a spanning tree** is a group of $n-1$ arcs connecting all nodes of the network and contains no loops.

Minimum spanning tree (MST)

A spanning tree of minimum length in a network is called a minimum spanning tree.

4.2 MST algorithm

The minimum spanning tree algorithm is used to find the minimum spanning tree for a given network. The algorithm comprises of the following steps.

Step One: Begin at any node i and join node i to node j, closest to node i. The two nodes i and j now form a connected set of nodes

$C = \{i, j\}$ and arc (i, j) will be in the minimum spanning tree. The remaining nodes in the network $(\bar{C})$ are the unconnected set of nodes.

Step Two: Choose a member of $\bar{C}(n)$ that is closest to some node in C. Let *m* represent the node in C that is closest to *n*. Then the arc (m,n) will be the minimum spanning tree. Update C and $\bar{C}$. Since *n* is now connected to $\{i, j\}$, C now equals $\{i, j, n\}$, and we must eliminate node *n* from $\bar{C}$.

Step Three: Repeat this process until a minimum spanning tree is found. Ties for the closest node and arc are broken arbitrarily. In this paper preference is given to the arc that is least likely to form a leaf. The strategy of the algorithm proposed in this paper lies in solving the TSP as an MST and then using branching to remove leaves.

Theorem 1

The MST algorithm finds a minimum spanning tree.

4.2.1 Proof by contradiction:

Let

S be the minimum spanning tree,

C_k be the nodes connected after iteration *k* of MST has been completed,

$\bar{C}_k$ be the nodes not connected after iteration *k* of MST has been completed,

A_k be the set of arcs in the minimum spanning tree after *k* iterations of MST algorithm have been completed.

Suppose that the MST algorithm does not yield a minimum spanning tree.

Then the arc chosen at iteration k (a_k) is not in *S*, i.e., $a_k \notin S$.

All arcs in A_{k-1} are in *S*, i.e., $A_{k-1} \subseteq S$.

This implies $\bar{a}_k \in S$ and $\bar{a}_k$ leads from node in C_{k-1} to a node $\bar{C}_{k-1}$.

Replacing $\bar{a}_k$ with a_k, we obtain a spanning tree shorter than *S*.

The contradiction proves that all arcs chosen by the MST must be in S. The MST algorithm does indeed find a minimum spanning tree. This proof can be found in books such as Gutin and Punnen (2006).

4.3 TSP tree, transformation and arc fixing

TSP tree

Is an MST without leaves other than the first and last nodes.

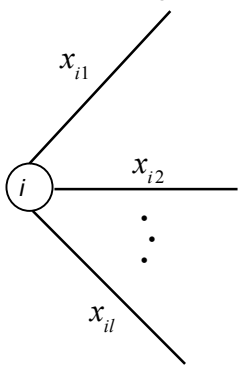
Transforming a spanning tree

Branching can be used to transform any MST into TSP.

Fixing an arc

The MST algorithm uses $n-1$ arcs to connect the nodes in the network, but the optimal solution of the TSP must have *n* arcs. This problem can be alleviated if one arc which is part of the optimal tour is known before the problem is solved. Unfortunately this arc is not known. In the optimal tour exactly two arcs are used in completing a tour at every node. With this knowledge arcs can be fixed and the necessary branching done so as to remove the unwanted leaves from a current minimal spanning tree. Let the number of arcs at a node *i* be *l*.

Figure 3
Arc fixing

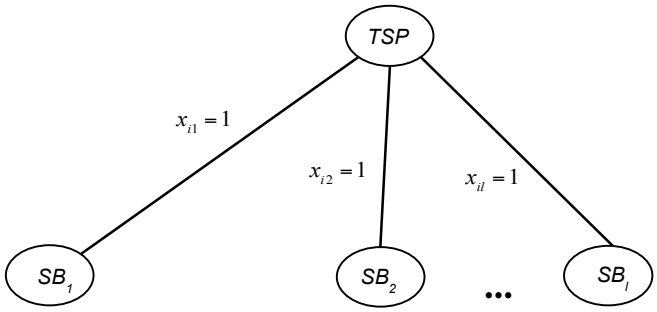


Arc fixing way 1: If a single arc is fixed then

$$x_{i1} + x_{i2} + ... + x_{il} = 1.$$

There will be a total of l sub-problems (SB) as shown in Figure 4 below.

Figure 4
Branching at node *i*



The sub-problems $SB_1, SB_2, ..., SB_l$ are mutually exclusive subsets of TSP.

$$SB_1 \cup SB_2 \cup ... \cup SB_l = TSP$$

For $\alpha \neq \beta$

$$SB_\alpha \cap SB_\beta = \varnothing$$

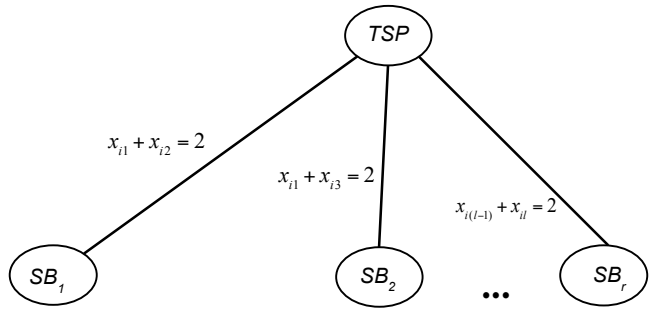
Fixed value in the h^{th} branch is given by F_h and is the value of a single arc. Fixing way 1 is done before applying the spanning tree procedure. The fixed arc is removed from the TSP before applying the spanning tree procedure.

Arc fixing way 2: At every node exactly two arcs are used in completing the tour. This is because at any node we must have one arc in and one arc out for us to complete the tour. This is also referred to as the *one arc in and one arc out rule*.

$$x_{i1} + x_{i2} + ... + x_{il} = 2$$

Fixing of two arcs at node *i* will result in $\frac{1}{2}l(l-1)$ sub-problems (*SP*) or branches.

Figure 5
Branching at node *i*



Where $r = \frac{1}{2}l(l-1)$. The number of sub-problems (*r*) formed by branching is given by

$$r = {}^{l}C_2 = \frac{l!}{2!(l-2)!}$$

$$= \frac{l \times (l-1) \times (l-2)!}{2!(l-2)!}$$

$$= \frac{l(l-1)}{2}$$

Also the sub-problems $SB_1, SB_2, ..., SB_r$ are mutually exclusive subsets of *TSP*.

$$SB_1 \cup SB_2 \cup ... \cup SB_r = TSP$$

For $\alpha \neq \beta$

$$SB_\alpha \cap SB_\beta = \varnothing$$

Fixed value in the h^{th} branch is given by F_h and is the sum of two arcs.

Fixing way two is also done before applying the spanning tree procedure. The two fixed

arcs and an included node *i* are removed from the TSP before applying the spanning tree procedure.

Theorem 2

Any sub-problem that results in at least one leaf in the network structure is infeasible.

From the definition of a TSP at node *i*,

$$x_{i1} + x_{i2} + ... + x_{il} = 2.$$

This is referred to as the one arc in and one arc out rule.

Proof: A leaf occurs when

$$x_{i1} + x_{i2} + ... + x_{il} = 1.$$

This is infeasible to the one arc in and one arc out rule. Two arcs are required to pass through a node.

4.4 Removing leaves in an MST

Besides fixing arcs, branching is also used to remove leaves in MST. A branch fathoms

- if the sub-problem becomes infeasible;
- if the MST for the sub-problem is also a TSP; and
- or if the objective value given by the sub-problem is larger than some given lower bound (LB).

4.4.1 Theorem 3

Any node *j* is selected for branching

a) if the number of arcs is greater than two, i.e., $x_{j1} + x_{j2} + ... + x_{jt} \geq 2.$
b) if it has the smallest number of arcs emanating from it.

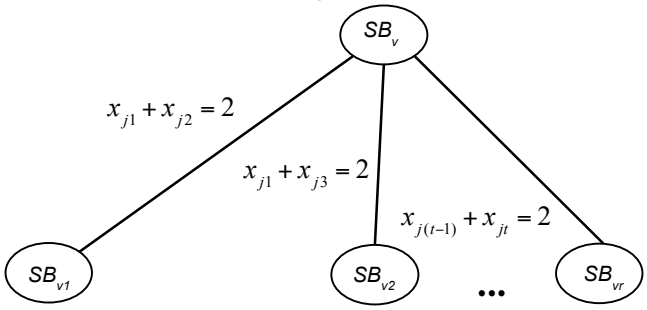
Where *t* is the number of arcs emanating from node *j*.

Proof (a)

Branching is done to enforce the restriction, $x_{j1} + x_{j2} + ... + x_{jt} = 2.$

This can be done if the number of arcs is two or more, i.e. $x_{j1} + x_{j2} + ... + x_{jt} \geq 2.$

Figure 6
Branching at node *j*



Where

- SB_v is the sub-problem after the v^{th} iteration;
- SB_{vk} is the k^{th} branch at k^{th} node.

Similarly the number of sub-problems (*vr*) formed by branching is given by

$$vr = \frac{t(t-1)}{2}$$

The sub-problems $SB_{v1}, SB_{v2}, ..., SB_{vr}$ are also mutually exclusive subsets of SB_v.

$$SB_{v1} \cup SB_{v2} \cup ... \cup SB_{vr} = SB_v$$

For $\delta \neq \lambda$

$$SB_\delta \cap SB_\lambda = \varnothing$$

Proof (b)

Let the number of arcs emanating from node i be given by t_i. The number of arcs at this node is directly proportional to the number of sub-problems r_i generated by branching at node i.

i.e. $t_i \sim r_i$

Suppose we are selecting nodes in an in increasing order of the number of arcs i.e.

$$r_1 \leq r_2 \leq ... \leq r_N$$

Where *N* is the number of nodes in the TSP and node 1 has the least number of arcs followed by node 2, then node 3 in that order up to variable node *N*. When the algorithm is applied the following sub-problems are visited in the search process. The worst case is assumed in the search process.

Starting with the most restricted variable

Stage 1: number of sub-problems visited = r_1;
Stage 2: number of sub-problems visited = $r_1 r_2$;

Stage N: number of sub-problems visited $= r_1 r_2 ... r_N$;
The total number of sub-problems (τ_A) is:

$$\tau_A = r_1 + r_1 r_2 + ... + r_1 r_2 ... r_N$$

Starting with the least restricted variable
Stage 1: number of sub-problems visited $= r_N$;
Stage 2: number of sub-problems visited $= r_N r_{N-1}$;
Stage N: number of sub-problems visited $= r_N r_{N-1} ... r_1$;
Total number of nodes (τ_B) is:

$$\tau_B = \gamma_N + \gamma_N \gamma_{N-1} + ... + \gamma_N \gamma_{N-1} ... \gamma_1$$

$$\therefore \quad \tau_A \leq \tau_B$$

It is computationally cheaper to start with node with the least number of arcs.

4.5 The TSP - MST inequality

The objective value (SUB(MST_o)) of any sub-problem obtained by solving as MST plus a fixed value (F_v) from one of the arcs is less than or equal to the optimal tour (SUB(TSP_o) of the sub-problem.

$$SUB(MST_o) + F_v \leq SUB(TSP_o)$$

Proof

$$SUB(MST_o) + F_v - F_v \leq SUB(TSP_o) - F_v$$
$$SUB(MST_o) \leq SUB(TSP_o) - F_v$$

Let $SUB(TSP_o) - F_v = T_s$, where a T_s is a spanning tree structure.
Then $SUB(MST_o) \leq T_s$
If $SUB(MST_o) = T_s$
Then T_s is a tree structure without any leaves but connects all the nodes.

This is valid, as SUB(MST) is the smallest sum of $(n-1)$ arcs used to connect all the nodes in any network structure where loops are not acceptable. Loops are not possible in T_s as one arc has been fixed.

4.6 Network branch and bound approach

Steps of the network branch and bound algorithm are summarized as follows.

Step 1: Select the node with the least number of arcs (l). Fix these arcs by branching into l sub-problems (way 1) or $\frac{l(l-1)}{2}$ sub-problems (way 2).

Step 2: Solve each sub-problem generated in Step 1 as MST. A sub-problem fathoms

- if it becomes infeasible;
- if the MST for the sub-problem is also a TSP; and
- or if the objective value given by the sub-problem is larger than some given lower bound (LB) obtained in an earlier sub-problem. The optimal tour is given as the sub-problem with the overall shortest tour.

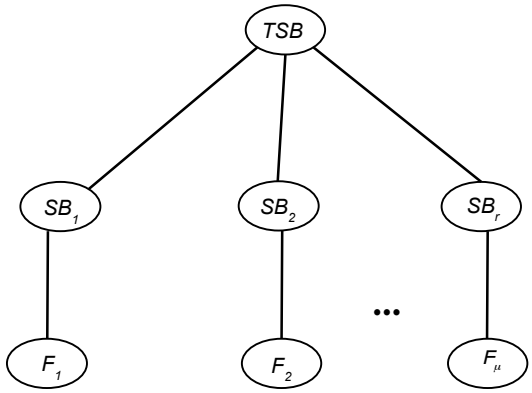
Else go to Step 3.

Step 3: From those MST with leaves select the node associated with the least number of arcs. (t). Branch into $\frac{t(t-1)}{2}$ sub-problems and return to Step 2.

4.7 Optimality

The solution obtained when using the network branch and bound is exact.

Figure 7
Optimality



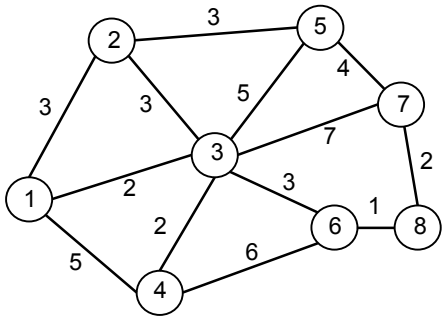
Where F_j is the fathomed sub-problem and μ is the number of fathomed sub-problems, $TSP_o = \min[F_1, F_2, ..., F_\mu] = SUB(MST_o)$.

4.8 Numerical illustration

Use the network branch and bound to solve the following TSP.

Figure 8

Main problem



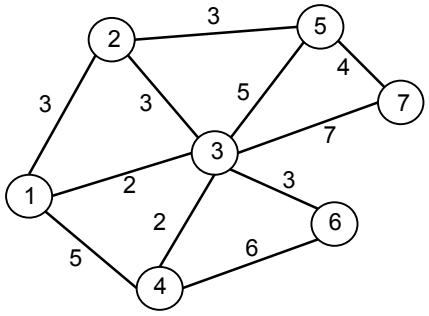
4.8.1 Solution using the network branch and bound method

The node with the least number of arcs is node 8. Fixing can be done in two ways. Way 1 is to fix either arc 6-8 or arc 7-8. This is done by using $x_{68} + x_{78} = 1$ to branch into two sub-problems. Way 2 is done by using $x_{68} + x_{78} = 2$ to branch into a single sub-problem $\left[\frac{2(2-1)}{2} = 1\right]$.

The two ways will produce the same solution and way 2 is arbitrarily selected for this illustration. Fixing is done by removing the two arcs 6-8 and 7-8 and the included node 8 from the TSP network diagram. The network diagram reduces to 7 nodes and 12 arcs, as shown below in Figure 9.

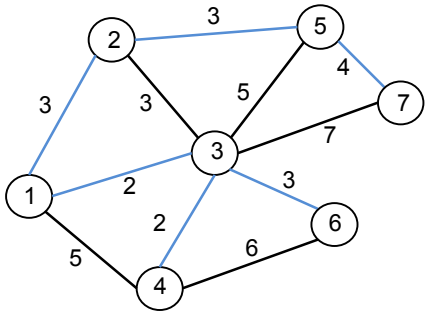
Figure 9

Fixing arcs 6-8 and 7-8 and removing included node 8, $F_1 = 1 + 2 = 3$.



Applying the minimum spanning tree algorithm, we have Figure 10.

Figure 10
First minimum spanning tree



$$MST = 3+2+2+3+3+4+F_1$$
$$= 3+2+2+3+3+4+(3)$$
$$= 20.$$

Selecting node 4 as the only leaf we have,

$$x_{14} + x_{34} + x_{46} = 2.$$

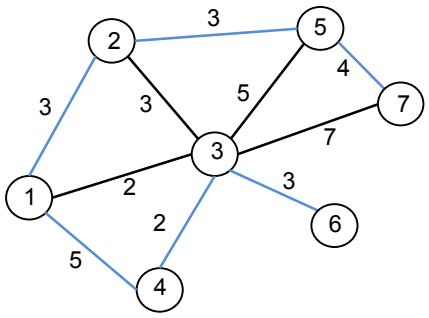
The equation implies the number of sub-problems is given by

$${}^3C_2 = \frac{3!}{2!(3-1)!}$$

$= 3$ sub-problems.

Sub-problem 2
(i.e. $x_{14} + x_{34} = 2$)

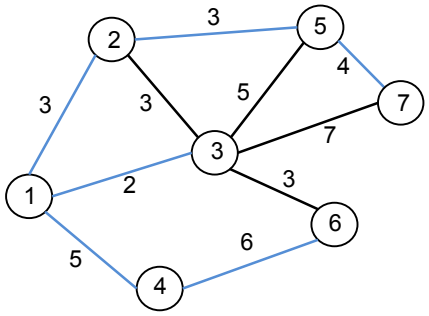
Figure 11
Fathomed Sub-problem 2



$$MST = 3+2+5+3+3+4+F_1$$
$$= 3+2+5+3+3+4+(3)$$
$$= 23 = LB$$
$$F_1 = 23.$$

Sub-problem 3
(i.e. $x_{14} + x_{46} = 2$)

Figure 12
Fathomed Sub-problem 3



$$MST = 6+5+2+3+3+4+F_1$$
$$= 6+5+2+3+3+4+(3)$$
$$= 26$$
$$F_2 = 26$$

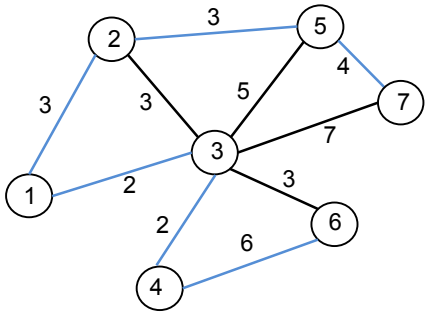
This sub-problem has fathomed, since F_2 is greater than the lower bound (LB) given earlier in sub-problem 2.

Sub-problem 4

$$x_{34} + x_{46} = 2$$

Figure 13

Fathomed sub-problem

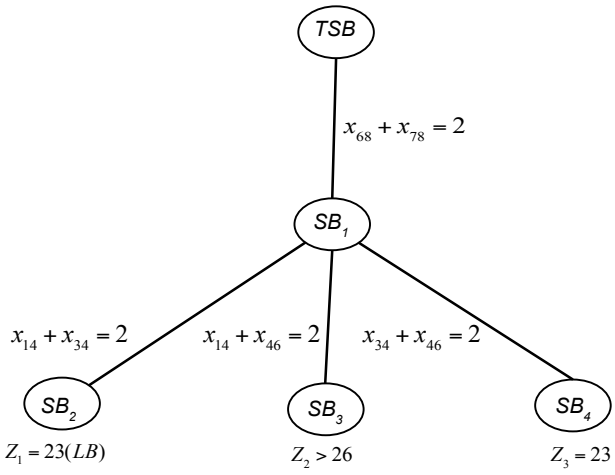


$$MST = 6+2+2+3+3+4+F_1$$
$$= 6+2+2+3+3+4+(3)$$
$$= 23$$
$$F_3 = 23$$

The sub-problem has fathomed, since the minimum spanning tree does not have leaves. The network branch and bound algorithm full search tree is presented in Fig 13.

Figure 14

Full Search tree



$$Z = \min[23+25+23] = 23$$

Figure 15
Optimal tour

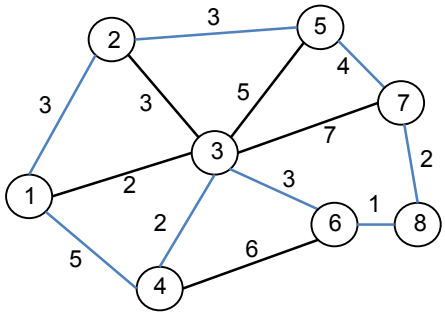
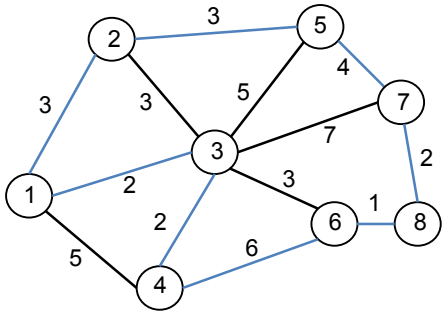


Figure 16
Optimal tour - alternative solution



7 Conclusion

The algorithm proposed in this paper uses spanning tree approaches as sub-problems in solving the difficult traveling salesman problem. A spanning tree approach is more efficient than either the LP based or the assignment sub-problems. For small TSP models it makes sense to use way 2 for fixing arcs since the number of branches is given by $\frac{l(l-1)}{2}$. This number of branches increases rapidly with an increase in the number of arcs. Thus for large TSP models it is wise to use way 1 since the number of branches is just *l*. The strength of the approach lies in the fact that the number of arcs on the various nodes of practical problems is not the same. Our strategy is to target those nodes that have the smallest number of arcs to form branches. At the moment large amounts of money are being wasted the world over by sales persons, rubbish trucks, delivery or postal companies and other organizations because exact solutions for routing problems cannot be determined and used in acceptable times. The network branch and bound approach proposed in this paper is still in its early stages of development and more effort will be put into refining it so that it reaches its full computational efficiency level. In future efficiency tests for this algorithm will be conducted on standard benchmark TSP instances.

References

BERMAN, P. & KARPINSKI, M. 2006. *8/7-Approximation algorithm for* (1,2)-TSP. Proc. 17th ACM-SIAM SODA conference:641-648.

GUTIN, G. & PUNNEN, A.P. 2006. *The traveling salesman problem and its variants.* Heidelberg: Springer.

KARLOF, J.K. 2005. *Integer programming: theory and practice*. Boca Raton FL: CRC Press Inc.

MITCHELL, J.E. 2001. Branch and cut algorithms for integer programming: In Floudas, C.A. & Pardalos, P.M. (eds.) *Encyclopedia of optimization*. Boston: Kluwer Academic Publishers.

NADEF, D. 2002. Polyhedral theory and branch and cut algorithms for the symmetric TSP. In: Gutin, G. & Punnen, A. (eds.) *The traveling salesman problem and its variations*. Dordrecht: Kluwer:29-116.

NEMHAUSER, G.L. & WOLSEY, L.A. 1988. *Integer and combinatorial optimization*. New York: John Wiley.

PADBERG, M. & RINALDI, G. 1991. A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60-100.

WINSTON, W.L. 2004. *Operations research applications and algorithms* (4th ed.) Boston: Duxbury Press.

WOLSEY, L. A. 1980. Heuristics analysis, linear programming and branch and Bound. *Mathematical Programming Study*, 13:121-134.