ABSTRACT

KULKARNI, GIRISH. A Tabu Search Algorithm for the Steiner Tree Problem. (Under the direction of Professor Yahya Fathi.)

The Steiner Tree problem in graphs is an NP-hard problem having applications in many areas including telecommunication, distribution and transportation systems. We survey, briefly, a few exact methods and a few heuristic approaches that have been proposed for solving this problem. Further, we propose a tabu search algorithm whose key feature includes a neighborhood definition consisting of exchange of key paths. The algorithm is empirically tested by running computational experiments on problem sets, with known optimal values, that are available over the internet. The results from the tabu search are compared with the optimal values and with the results of a well-known heuristic procedure. The experimental results show that the tabu search algorithm is reasonably successful. It produces near-optimal solutions in the experiments conducted and performs better than the heuristic procedure. We also explore other avenues for future work and possible extensions to the tabu search algorithm.

A TABU SEARCH ALGORITHM FOR THE STEINER TREE PROBLEM

By

GIRISH KULKARNI

A Thesis submitted to the graduate faculty of
North Carolina Sate University
in partial fulfillment of the
requirements for the degree of
Master of Science

OPERATIONS RESEARCH

Raleigh
2002

APPROVED BY

Stephen D. Roberts

Dr. Stephen Roberts

[illegible]

Dr. George Rouskas

Yahya Fathi

Dr. Yahya Fathi

(Chair of Advisory committee)

BIOGRAPHY

Girish Kulkarni was born on the 1st of November 1976 in the beautiful coastal city of Margao in the state of Goa, India. After completing his schooling in Mumbai, he obtained the Bachelor of Technology degree in Metallurgical Engineering and Materials Science from IIT Bombay. Before joining the Operations Research program at NCSU , he worked as a software engineer for a year at Infosys Technologies Ltd in Pune, India.

As graduate student at NCSU, Girish Kulkarni undertook course work in Operations Research and Computer Science and worked in the area of meta-heuristics for combinatorial optimization problems under the direction of Dr Yahya Fathi. He was also an active member of the international honor society for Operations Research, *Omega Rho*. His research interests include areas related to discrete optimization and algorithm development. Presently, he is enrolled as a doctoral candidate in Operations Research at NCSU.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents Dr. Suneela Kulkarni and Dr. Mangesh Kulkarni whose support and encouragement has led me to success. I would also like to thank my advisor Dr. Yahya Fathi for his advice and painstaking guidance. I am grateful to my committee members Dr Stephen Roberts and Dr. George Rouskas for their encouragement and feedback.

I would also like to mention my friends Prashant, Rajesh, Bobby and Sai and finally, last but not the least, I would like to thank my room mates, Prabhas Sinha and Vijay Iyer, for the memorable time I had in the last two years.

TABLE OF CONTENTS

List of Figures

List of Tables

1 Introduction

1.1 The Steiner Tree Problem

The S*teiner Tree Problem* involves constructing the least cost network that spans a given set of points. The novelty of the *Steiner Tree Problem* is that new auxiliary points can be introduced between the original points so that the spanning tree of all the points will be of lower cost than otherwise possible. The S*teiner Tree Problem* can be categorized into three major areas: the *Euclidean Steiner Tree Problem* (SPE), the *Rectilinear Steiner Tree Problem* (SPR) and the S*teiner Tree Problem in Graphs* (SPG). Here, we focus our attention only on the SPG, which is a combinatorial version of the *Steiner Tree Problem*. Henceforth, in this text we will refer to the SPG as the steiner tree problem.

A large amount of literature concerning the various aspects of the steiner tree problem exists and it includes exact methods as well as various heuristic techniques. Here, we propose a new tabu search algorithm for solving the steiner tree problem. In this chapter, we give the exact problem definition along with an introduction to the terminology and notation used in the remainder of the text. The applications and the computational complexity of the problem are also discussed. Chapter 2 is a survey of the recent literature with emphasis on the ideas that are relevant to the tabu search algorithm that has been developed here. First, the Cheapest Insertion heuristic is explained along with a few of its multiple pass variants. Then we briefly discuss a meta-heuristic strategy called the pilot method. We also touch upon other meta-heuristic approaches like simulated annealing and genetic algorithms and lastly the motivation behind the tabu search algorithm that we propose is given. Chapter 3 describes the various aspects of the tabu search algorithm, including the pseudo-code and experiments that led to selection of the various parameter values. Chapter 4 presents the results of the experiments conducted to test the tabu search algorithm. The tabu search results are compared to known optimal solutions of two sets of problem instances. Furthermore the tabu search is compared with another heuristic method. Chapter 5 mainly talks about avenues and ideas for future work.

1.2 Problem Definition

The steiner tree problem can be defined as follows.

Given: An undirected graph G = (**V**, **E**, w), where, w: E $\rightarrow$R is a non-negative weight function and a non-empty set **N** of terminals, **N** $\subseteq$ **V**. Here **V** denotes the set of all nodes present in the graph and **E** represents the set of all edges present in the graph. The graph G consists of two sets of nodes, the set of terminals denoted by **N,** and a set of non-terminals denoted by **Q,** i.e., **Q** = {v: v $\in$V/N}.

Find: sub-graph T_g of G such that:

1) Each pair of terminals is linked through a path in T_g.
2) The total weight of all edges in T_g is minimized.

The sub-graph T_g consists of all terminal nodes **N** of G and some non-terminal nodes. These non-terminal nodes which are present in the sub-graph T_g are called steiner nodes or steiner vertices. The sub-graph T_g obtained is known as the steiner *minimal* network (it should be referred to as the steiner *minimum* network, however for historical reasons the less correct term is used). If all the edge weights are positive, the steiner minimum network has to be a tree and hence the problem is often referred to in the literature as the steiner tree problem and T_g is often called the steiner minimal tree. Throughout our discussion we will refer to the optimal solution as the steiner minimum tree and a sub-optimal solution as the steiner tree. $|\mathbf{N}|$ represents the number of terminals, $|\mathbf{V}|$ the number of vertices and $|\mathbf{E}|$ the number of edges. The graph will be represented as G and any tree will be represented as T. $\mathbf{V}_T$, $\mathbf{E}_T$ and $\mathbf{N}_T$ represent the set of all nodes, all edges and all terminals in the tree T, respectively. If T represents a legitimate solution then $\mathbf{N} = \mathbf{N}_T$. A path in any graph G having end nodes u_i and v_i and intermediate nodes $x_1, x_2, x_3, \ldots, x_k$ is denoted by $P(u_i,\ldots,v_i)$. The total length of this path is represented by $c(P)$ or by $d(v_i,u_i)$. Similarly the total length of any tree T is represented by $c(T)$.

1.3 Complexity

Some well known special cases of the steiner tree problem can be solved in polynomial time. When $|N| = 2$ the problem reduces to the shortest path problem while when $\mathbf{N} = \mathbf{V}$ the problem reduces to the minimum spanning tree problem. Both these problems can be solved in polynomial time. On the other hand, a large number of special cases have been proved to be NP-hard. The steiner tree problem is NP-hard when the graph G is a chordal graph, a bipartite graph or a complete graph with edge weights either 1 or 2. Thus in the general case the problem is an NP-hard problem. However the complexity of the problem in some special cases like that of planar graphs with all edges of weight 1 has not yet been settled. [6]

1.4 Applications

The steiner tree problem has numerous applications especially in the areas of telecommunication, distribution and transportation systems. The computation of phylogenetic trees in biology and the routing phase in VLSI design are real life problems that have been modeled as the steiner tree problem [6]. Minimization of the cost of trees generated as a solution for multicast routing in communication networks have been traditionally formulated as the steiner tree problem. Another interesting application is based on the billing strategies of large telecommunications network service providers. The bill isn't based on the actual number of circuits provided, which may change over a period of time, but on the basis of a simple formula calculated for an ideal network which will provide all the facilities at minimum cost. Moreover, several network design problems can be formulated or considered as generalizations of the steiner tree problem.

2 Literature Survey

2.1 Exact Methods

Many exact methods for optimal solutions for the steiner tree problem have been proposed in the literature. Hwang et al. [6] give a comprehensive overview of the many approaches including enumeration schemes, branch and bound methods, dynamic programming as well as mathematical programming formulations. Here, two methods, the spanning tree enumeration and a branch and bound algorithm are explained.

2.1.1 Spanning Tree enumeration

This algorithm was proposed by Hakimi [5] and a modification was proposed by Lawler [8] based on the observation that finding a steiner minimum tree for **N** in graph G is equivalent to finding a steiner minimum tree for N in the distance network induced by all vertices V in graph G. This enumeration is based on two results [6]

1) There exists a steiner minimum tree for |**N**| terminals in a given distance network where each steiner vertex has degree of at least 3.
2) There exists a steiner minimum tree for |**N**| terminals in a given distance network with at most |**N**|-2 steiner vertices.

This enumeration procedure involves enumerating minimum spanning trees of sub-networks induced by supersets of the $|\mathbf{N}|$ terminals of size at most $2|\mathbf{N}| - 2$.

The number of minimum spanning trees that must be determined is given by

$$\sum_{i=0}^{|\mathbf{N}|-2} \binom{|\mathbf{V}|-|\mathbf{N}|}{i} \leq 2^{(|\mathbf{V}|-|\mathbf{N}|)}$$

Here, $\binom{n}{k}$ denotes "*n choose k*" and so $\binom{n}{k}$ = n!/(n-k)! k!.

The execution time of this algorithm is asymptotically upper bounded by $(|\mathbf{N}|^2 \cdot 2^{(|\mathbf{V}|-|\mathbf{N}|)} + |\mathbf{V}|^3)$ i.e. time complexity is given by O(by $(|\mathbf{N}|^2 \cdot 2^{(|\mathbf{V}|-|\mathbf{N}|)} + |\mathbf{V}|^3)$ [6]. The term $|\mathbf{V}|^3$ comes form the Floyd-Warshall algorithm used to compute the shortest path from each vertex to all other vertices.

2.1.2 Branch and Bound method for the steiner tree problem

Many branch and bound algorithms are available in literature to optimally solve the steiner tree problem. Here we briefly describe the algorithm proposed by Shore et al. [11]

For a graph G, let $\mathbf{F_0}$ represent the set of all feasible solutions (all trees spanning the set of terminals **N**). The branch and bound algorithm proceeds by systematically partitioning the set $\mathbf{F_0}$ into subsets. These subsets are investigated using upper and lower bounds to check whether they can contain a steiner minimum tree. Let $\mathbf{F_i}$ be one such subset. $\mathbf{F_i}$ is characterized by two sets of edges $\mathbf{IN_i}$ and $\mathbf{OUT_i}$. The set $\mathbf{IN_i}$ contains all the edges that have to be present in all solutions belonging to $\mathbf{F_i}$. $\mathbf{OUT_i}$, on the other hand, contains all those edges that are not permitted in any solution present in set F_i. Initially, for $\mathbf{F_0}$, $\mathbf{IN_0} = \Phi$ (empty set) and $\mathbf{OUT_0} = \Phi$.

For any subset $\mathbf{F_i}$, the graph G_i is obtained by removing from G all edges belonging to OUT_i and contracting the edges belonging to $\mathbf{IN_i}$. Contraction of an edge is a process in which the edge and its end nodes are replaced by a single node and this node is marked as being a terminal node. Contracting an edge e_i is one way of ensuring that every solution T_i contains edge e_i. Thus, by reducing graph G to G_i, each edge of $\mathbf{IN_i}$ is forcibly included in every solution T_i and every edge in $\mathbf{OUT_i}$ is excluded from T_i. Let $\mathbf{N_i}$ be the set of terminals in G_i. Finding the steiner minimum tree for the graph G is equivalent to finding the steiner minimum tree T_i, for all terminals of $\mathbf{N_i}$ in graph G_i. First, an upper bound B_i for set $\mathbf{F_i}$ can be determined by adding the lengths of edges in $\mathbf{IN_i}$ to the length of a tree spanning $\mathbf{N}_i$ in G_i obtained by any of the steiner tree heuristics described in section 2.2. Let c_{min} be the length of the shortest tree spanning all the terminals found till date. At the start of the branch and bound algorithm, $c_{min} = \infty$, and at any later stage if $c_{min} > B_i$ then c_{min} is set equal to the upper bound B_i.

In order to find a lower bound $\underline{b_i}$ for F_i, we define the following

- w_{ni} is the length of the shortest edge incident on n_i, where $n_i \in \mathbf{N}_i$
- $w^{\sim}_{ni}$ is the length of the second shortest edge incident on n_i, where $n_i \in \mathbf{N}_i$

- w'_{ni} is the length of the shortest edge between n_i and n_k, where n_i and $n_k \in \mathbf{N}_i$

and $c(T_i)$ is the total length of all edges in T_i. Thus given a terminal n_i we have a notation for the shortest and second shortest edge incident on it and the shortest edge between n_i and any other terminal.

We consider two cases while trying to find a lower bound. The first case is when T_i contains at least one steiner node then it has atleast $|\mathbf{N}i|$ edges and each terminal $n_i \in \mathbf{N}_i$ is incident on atleast one edge in T_i. We can now define a quantity t_i such that

$$t_i = \sum w_{nj} \leq c(T_i)$$

Thus t_i is the sum of the weights of the shortest edge incident on each terminal n_i. The second case is when T_i contains only terminals, then T_i has $|\mathbf{N}_i|$ -1 edges. For this case we define the quantity t_i' such that

$$t_i' = \sum w'_{nj} - \min_{n_j \in \mathbf{N}_i} (w'_{nj} \mid n_j \in \mathbf{N}_i) \leq c(T_i)$$

Now, the quantities t_i and t_i' are used to define a lower bound $\underline{b_i}$, which is given below

$$\underline{b_i} = \min(t_i, t_i') + \sum_{e_m \in IN} (e_m)$$

If $c_{min} \leq \underline{b_i}$, then no tree in F_i can be shorter than the shortest tree found so far and so there is no need to partition the set $\mathbf{F_i}$ any further. A solution is a feasible solution when all the terminals present in $\mathbf{N_i}$ are connected to each other in the solution T_i only with edges from $\mathbf{IN_i}$. Whenever it is uncertain whether F_i contains a steiner minimum tree, it is partitioned into two subsets for a suitably chosen edge $e_m \in \mathbf{E} \backslash (\mathbf{IN_i} \cup \mathbf{OUT_i})$:

$\mathbf{F_j}$: where $\mathbf{IN_j} = \mathbf{IN_i} \cup e_m$, $\mathbf{OUT_j} = \mathbf{OUT_i}$

$\mathbf{F_k}$: where $\mathbf{IN_k} = \mathbf{IN_i}$, $\mathbf{OUT_k} = \mathbf{OUT_i} \cup e_m$

The choice of e_m is crucial to the performance of the algorithm. The authors select as e_m, the minimum length edge incident with a terminal $n_j \in \mathbf{N_i}$ for which $(w^{\sim}_{nj} - w_{nj})$ is maximum. $\mathbf{F_j}$ is examined first and a steiner tree is found for $\mathbf{F_j}$ or it is established that an optimal feasible solution cannot be found in $\mathbf{F_j}$. Then $\mathbf{F_k}$ is examined and the algorithm backtracks to $\mathbf{F_i}$. The minimum length steiner tree found while the algorithm backtracks to $\mathbf{F_0}$ is the solution.

2.2 Heuristic Techniques

Various approaches have been used to develop heuristics for the steiner tree problem. A comprehensive review of these heuristics is presented by Hwang et al. [6] and more recently by Voss [14]. Most of these heuristic methods are constructive methods and can be classified on the basis of a common characteristic in their approach. Some heuristics follow the path-based approach, there are others that are vertex-based and still others that can be considered to be tree based heuristics. Furthermore, there exist a number of other heuristics that do not fall in any of the above categories such as the contraction heuristic or the dual ascent heuristic. These constructive heuristics give rise to important ideas and many of the ideas for neighborhood definitions and solution representations have their origins in these constructive heuristics. Theoretically, nearly all heuristics have a worst case upper bound of 2. One variation of the shortest distance graph heuristic has been proven by Zelikovsky [16] to have worst-case error bound of 11/6.

2.2.1 The Cheapest Insertion Heuristic

Below, we briefly describe a constructive heuristic originally proposed by Takahashi and Matsuyama [13]. This heuristic, which will be referred to as CHINS henceforth, is based on Prims's algorithm for finding the minimum spanning tree in a graph. The authors have also shown that the length of the steiner tree obtained using the CHINS heuristic $c(T_{CHINS})$ is bounded. The bound is also proved to be tight and is given by $c(T_{CHINS})/c(T_{opt}) \leq 2 - 2/|\mathbf{N}|$ for any graph G and any set of terminals **N**.

The shortest Path heuristic involves the following steps

1. Begin by choosing an arbitrary terminal n_k and include it in the tree T. T is the current partial solution and contains only one vertex: the terminal n_k. Let k= 1. (`Fig 1` a)
2. If $k = |\mathbf{N}|$ then Stop. Else determine a terminal n_{k+1} that is closest to the current solution T. Add to T all the vertices on the shortest path from n_{k+1} to T. (`Fig 1` b)
   k = k+ 1. Repeat.
3. Find a minimum spanning tree for the sub-graph induced by the nodes in T. (`Fig 1` d)
4. Remove from the minimum spanning tree non-terminals of degree 1, one at a time. The resulting tree is a sub-optimal solution.

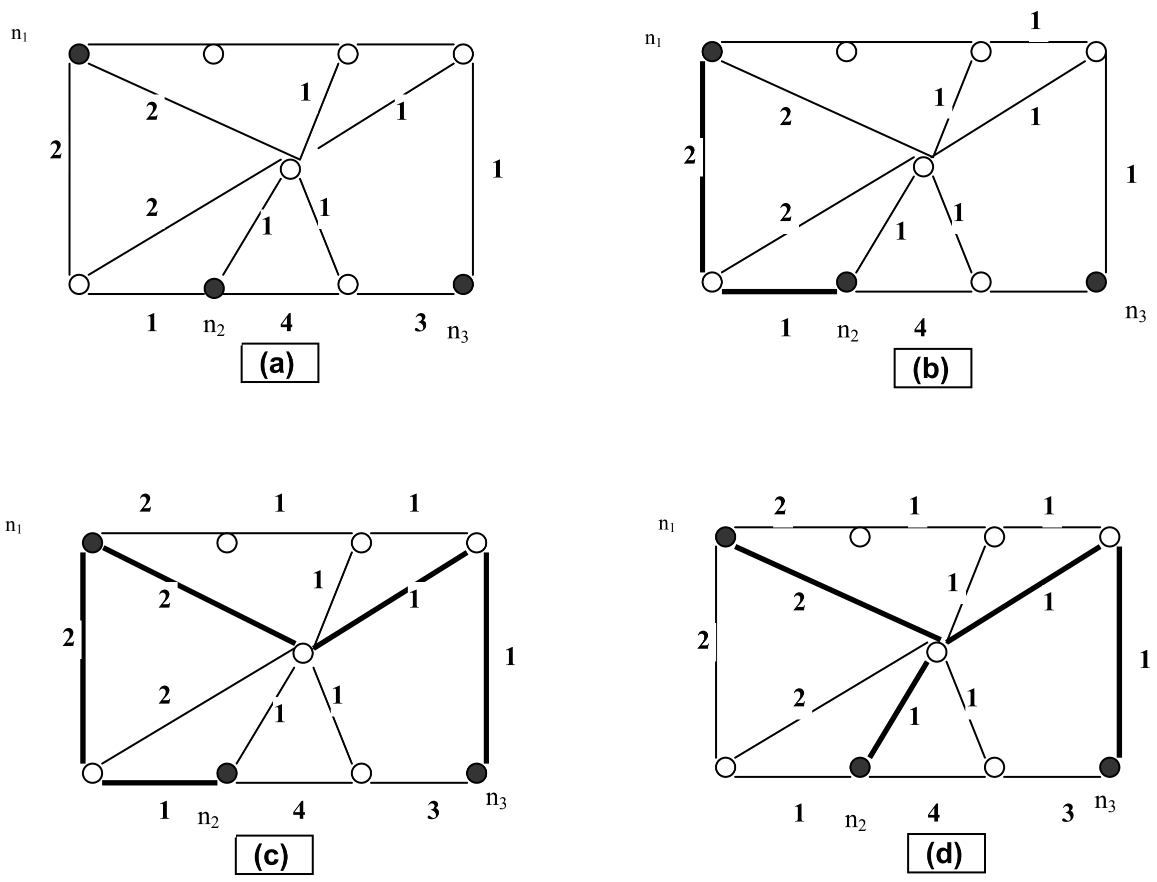

Fig 1 The cheapest insertion heuristic (CHINS)

This heuristic is sensitive to the choice of initial solution and so it needs to be repetitively applied. Some strategies that have been suggested for repetitive runs of this heuristic are

CHINS – N	: Begin each run with a different terminal. Total number of runs = |**N**|
CHINS – V	: Begin each run with a different vertex. Total number of runs = |**V**|
CHINS – nV	: Fix a terminal n_i . Begin each run with the shortest path from a terminal n_k ($k \neq i$) to n_i. Total number of runs = N –1.
CHINS – NN	:Begin each run with the shortest path from a different pair of terminals. Total number of runs = N (N – 1).

2.2.2 The Pilot Method

The pilot strategy proposed by Cees Duin and Stefan Voss [2] is a meta heuristic approach which can be applied for solving the steiner tree problem. This meta-heuristic has been proposed by the authors to enable, without much effort, a tradeoff between computation time and accuracy. For NP-hard problems heuristics exist that solve the problem in polynomial time using quick greedy rules to incrementally construct a solution. These greedy rules are often myopic and do not take into account the long term effect of a decision and hence such constructive heuristics often return reasonable but not near-optimal solutions. It would be extremely beneficial if it were possible to take into account how present decisions affect later choices in the algorithm. The pilot heuristic strategy is one such look-ahead method. A similar strategy has been used in the area of artificial intelligence where there is an evaluation of a decision with respect to the likelihood of that decision yielding a good solution in the future and to prune away those decisions which are not likely to be promising. The authors call this approach a tempered greedy approach and it is meta-heuristic in the sense that it can be formulated to suit all kinds of combinatorial problems.

In the pilot strategy, a combinatorial optimization problem is tackled by using multiple passes of an existing heuristic that is called the pilot heuristic. The pilot heuristic is used to build step-by-step a partial solution that is called the "master solution". Consider that a partial solution **M** exists and let there be a number of choices represented by $\boldsymbol{C}$ that can be used to extend this partial solution. Now for each choice $c_i \in \boldsymbol{C}$, a complete solution is built using the pilot heuristic. The choice c_0 corresponding to the best solution is chosen to extend the partial solution **M** by changing it in minimal fashion. On the basis of the changed master solution new pilot calculations are started for each $c_i \notin$ **M** providing a new solution element c'_0 and so on. The pilot heuristic actually provides an upper bound on the objective value of the final solution that can be obtained if the choice c_0 is chosen. Suppose the pilot heuristic or the sub-heuristic takes execution time $O(n^k)$ then the pilot method takes time $O(n^{k+2})$ where n represents problem size and k is a positive integer. Specifically for the steiner tree problem, a slight modification of the CHINS heuristic described in section 3.2.1 can be used to formulate a pilot method. At any given time a partial solution **M** is a forest covering the vertices of set **N**. Each tree in this forest can be considered a component and can be replaced by a single terminal. Initially, **M** consists of all the terminals, i.e., it has a total of |**N**| components. The pilot method proceeds by running the CHINS heuristic from each vertex v as the starting point. Let v_0 be the starting vertex from which the best objective value is obtained. We now run one step of the CHINS heuristic from v_0. If v_0 is a terminal then we look for a component that is closest to it and create a new one by combining it with vertex v_0. If v_0 is not a terminal then we combine v_0 with the closest terminal and then repeat this procedure once more to obtain the new component. This process continues till the partial solution **M** consists of only one component that contains all vertices of **N**.

2.2.3 Local Search Methods

Apart from these constructive heuristics, other techniques such as neural networks, local improvement methods as well as meta-heuristics like simulated annealing, genetic algorithms and Tabu Search have also been investigated. Local improvement strategies are based on two types of transformations: edge oriented transformation and node oriented transformation. Edge oriented transformations may involve deletion of an edge from a steiner tree along with the resulting steiner nodes of degree one. The disconnected components are then connected to each other through the shortest path. In node oriented transformation all nodes in the graph are classified as *in* or *not in*. Critical vertices (steiner nodes with degree greater than or equal to 3) in the solution are eliminated and the components are reconnected using the shortest paths. Local improvement approaches based on the node-oriented strategies are time consuming. This could be because the search may involve large number of exchanges of steiner nodes of degree 2 without any change in the objective value. Another reason is the recalculation of minimum spanning trees at each iteration.

2.2.4 Simulated Annealing and Genetic Algorithms

Quite a few Simulated Annealing [1] and Genetic Algorithm [3] approaches have been investigated and according to Voss [14] the results indicate that the SA approach can be effective with respect to solution quality. The most effective SA approach for the steiner tree problem, implemented by Wade and Rayword-Smith [15], regularly changes its search space topology by changing the problem representation and related neighborhood transformation. The solution in a GA can be represented using a bit string of size equal to the number of non-terminal nodes in the graph. Numerical results show that using appropriate values of parameters, the optimal solution to small-sized problems can be obtained within reasonable time. Genetic algorithms may also be combined with the idea of chunking. The general idea of chunking approach is to speed up the process of evolution by combining chunks of attributes that seem to be superior and to define schemata being responsible for the solution quality.

2.2.5 Tabu Search

Tabu search heuristics based on the node-oriented neighborhood have been investigated in a number of papers. Sondergeld and Voss [12] investigate a specific intensification-diversification approach where the diversification is achieved by scattering multiple search trajectories over the solution space and intensification is based on strategic restarts of search strategies on “good” solutions that have been previously encountered. Gendreau et al. [4] present a Tabu search strategy where the initial solution is constructed using the cheapest insertion heuristic and the search strategy is based on insertion and deletion of steiner nodes. The algorithm has a diversification procedure which includes ideas such as, include steiner nodes previously not part of the solutions, drop the path between nodes where shortest in-tree distance between the nodes is much larger than the shortest distance between them in the graph, etc. Another important contribution is the consideration of efficient data structures for manipulating trees when adding and deleting nodes, as the computational effort for exploring a neighborhood can be enormous. Rebeiro et al. [10] have proposed a tabu search algorithm which has similar neighborhood structure and search strategy as the one mentioned above. In this paper the authors define procedures by which the neighborhood search is made faster and more efficient so that comparable solution quality is obtained in considerably smaller execution times.

2.2.6 A New Tabu Search Algorithm

Voss [14] summarizes the comparison of the various algorithms whose performance on the problem sets B, C, D and E created by Beasley is available in the literature. For the local improvement the crucial edge exchange strategy produced better results than the key path exchange idea when considering quality of solutions. The tabu search strategies showed markedly better results then the local improvement methods but the results for different diversification schemes for the tabu search were very close to each other. The genetic algorithms preformed slightly worse then the tabu search methods. When reduction techniques were used before the application of GA or tabu search, the results were much better for both tabu search and GA. Thus published results point out the fact that the tabu search algorithms show a lot of promise for the steiner tree problem. Moreover, the edge oriented neighborhood definition for Tabu search still lacks thorough consideration [14].

We propose a tabu search algorithm based on an edge oriented neighborhood definition for the undirected steiner tree problem in graphs. This algorithm begins with an initial solution obtained by running the CHINS heuristic from a starting vertex. The tabu search improves the solution over a series of iterations. Each iteration investigates all neighbors of the current solution obtained by a key path exchange mechanism. This mechanism is performed so that the computation required for the evaluation of each neighbor is minimal and hence identification of the best neighbor is speedier. The algorithm proceeds by moving from the current solution to its best neighbor. This procedure is repeated and terminates when a stopping criterion is met. This algorithm also employs a diversification procedure that forces the search into unexplored regions of the search space.

The various aspects of this tabu search algorithm are explained in detail in the next chapter.

3 Tabu Search

This chapter describes the various aspects of the tabu search algorithm in detail. We begin with the solution representation and go on to describe the neighborhood structure, search strategy, tabu list, aspiration criterion, stopping criteria and objective evaluation. The chapter also includes a description of the diversification procedure and the pseudo-code for the tabu search.

3.1 Solution Representation

In the context of the tabu search, any tree T that contains all terminal nodes is a solution. Apart from all the terminal nodes a solution will also contain non-terminal nodes called steiner nodes. The set of steiner nodes in a solution T, will be referred to as $\mathbf{Q}_{T,}$ i.e., $\mathbf{Q}_T = \{v: v \in \mathbf{Q}$ and v is in T$\}$.

The solution tree T is stored in two formats, the first one is a one dimensional array called the parent array and the second is a list of arrays called the adjacency list structure. The solution T can be considered to be a rooted tree and hence each node in T, except the root node, has another node in T as its parent (or predecessor). The parent array format stores the complete list of predecessor nodes. The adjacency list is motivated as follows; if a node *u* in T is connected to another node *v*, which also is in T, through a single edge, than *v* is a neighbor of *u*. Thus, every node *u* in T has an adjacency list that contains all its neighbors. The adjacency list structure representation of solution T is a collection of the adjacency lists of all nodes in T. The parent array structure is important because it helps in the traversing of all the nodes of the steiner tree. The adjacency list is required because it is used in the various graph algorithms used as part of the tabu search.

3.2 Data Structures

Firstly, we assume that the nodes are numbered from 1 to $|\mathbf{V}|$ in some arbitrary manner. Let the current solution T, have $\mathbf{V}_T$ as the set of all nodes, $\mathbf{E}_T$ as set of all edges and node s_T as the source. We denote the parent array for T by $\boldsymbol{\pi}$, where the kth element of $\boldsymbol{\pi}$, $\pi(k)$ contains the parent (or predecessor) of node k for all $k \in \mathbf{T}$ and $k \neq s_T$. For the source node s_T, $\pi(s_T)$ is defined to be equal to s_T, i.e., the source node is its own parent. Since the total number of edges in a tree is one less than the total number of nodes, a parent array of size $|\mathbf{V}_T|$ is adequate to store the tree. In this Tabu search implementation, the parent array has size equal to $|\mathbf{V}|$, the total number of nodes in the graph. For those nodes v of the graph G which are not in the present solution, i.e., $\forall\ v \in \mathbf{V}$ but $v \notin \mathbf{V}_T$, we define $\pi(v) = 0$.

The adjacency list Adj, for the steiner tree T, is implemented as an array of $|\mathbf{V}|$ arrays, one for every node in the graph G. For each node u, such that $u \in \mathbf{V}_T$, the adjacency list Adj(u) contains all the vertices v such that there is an edge e_{uv} joining u to v and $e_{uv} \in \mathbf{E}_T$. Thus Adj(u) contains all nodes adjacent to u in the tree T. The vertices in the adjacency list are stored in arbitrary order. Those nodes k that do not belong to the tree have empty adjacency lists, $\forall\ k \in \mathbf{V}$ but $k \notin \mathbf{V}_T$, Adj(v) = ϕ. The size of the adjacency list for a node *u* equals the number of edges incident on *u*. Thus, the total space used to store the adjacency list structure is at least $2(|\mathbf{N}| -1)$.

The graph G itself is stored in two formats one form is the adjacency list described above and the other data structure is known as the adjacency matrix. The adjacency matrix A of the graph G is a square matrix with number of rows and columns equal to $|\mathbf{V}|$. If the weight w of the edge connecting node *u* to node *v* is represented by $w(u,v)$, then $a_{uv} = w(u,v)$, when the edge $e_{uv} \in \mathbf{E}$, and $a_{ij} = 0$, otherwise.

3.3 Neighborhood Structure

Any intermediate solution occurring during the course of the search is a tree that consists of all terminal nodes in the graph and some steiner nodes. Based on the degree of the nodes in the solution T, we classify the nodes in the graph into two types: critical nodes and non-critical nodes. All steiner nodes of degree greater than 2 and all terminal nodes are classified as critical nodes. Thus, all non-critical nodes are steiner nodes with degree exactly equal to 2. Further, we define a key path to be a path in the steiner tree that has critical nodes as its end points and each intermediate node is a non-critical node. A steiner tree can be considered to be a minimum spanning tree with critical nodes as vertices and key paths as edges. When all the steiner nodes are critical nodes, then the total number of key paths equal the total number of edges in the tree. Hence the maximum number of key paths in a steiner tree T equals $|\mathbf{E}_T| = |\mathbf{V}_T| - 1$.

A tree T' is defined as a neighbor of tree T, if T' is constructed from T by removing a key path from T and reconnecting the two fragments formed with the shortest path between the two fragments. Every key path P_i in tree T gives rise to a neighbor T' and hence the total number of neighbors for given steiner tree T equals the total number of key paths in T. Thus the neighborhood size is at the most $|\mathbf{V}_T|$.

3.4 Search Strategy

A single iteration of the tabu search consists of investigating the entire neighborhood of the current solution and selecting the best non-tabu neighbor. This neighbor is accepted and used as the current solution for the next iteration. If the best solution in a given iteration is not an improving solution it is still selected and the search proceeds by exploring all its neighbors. The algorithm always keeps track of the best ever solution found in the course of the search.

3.5 Evaluation of Objective

The objective value of a steiner tree is the total weight of all edges in it. To find the objective value of any given solution T, we need to add up the weights of all the edges in T. The search strategy involves choosing the neighbor with the best objective value and this objective value is speedily calculated as follows. At the start of the tabu search the CHINS heuristic calculates an initial solution T_i and also gives the total edge weight of T_i. We move from T_i to its neighbor T_i' by dropping an existing key path from the tree T_i and re-connecting the fragments with another path to obtain T_i'. Now, using the weights of all the edges of the dropped and the added paths and the objective value of T we can easily find the objective value of T'. Let us denote the difference between the weights of the dropped path and the added path as δ. Thus δ subtracted from the objective value of T_i will give us the objective value for T_i'. In the neighborhood search, we do not need to calculate the objective value of each neighbor obtained during the search. The neighbor T' which corresponds to the highest value of δ is the best neighbor of T.

3.6 Tabu list and Aspiration criteria

The search algorithm searches for the best possible neighbor T_{nbr} of the current accepted solution T. The Tabu search will go on to T_{nbr} even though T_{nbr} may have objective value greater than T. This can cause the tabu search to cycle between T_{nbr} and T unless we introduce appropriate measures to prevent cycling. To this end, we maintain a tabu list. The tabu list stores a list of attributes (or it might store complete solutions too) and these attributes reside in the tabu list for a specific number of iterations called the tabu tenure. A solution containing attributes which are tabu active (i.e., attributes that are present in the tabu list) is avoided. Here, the tabu list is implemented as a square matrix TL having number of rows and columns equal to |**V**|. Each element TL(u,v) of the matrix represents the edge e_{uv} connecting nodes *u* and *v*, and stores the number of iterations for which that edge is tabu active. Evaluating a neighbor involves two steps, the first step is to select a key path to drop and the second step is to determine the shortest path between the two fragments formed. It is during the first step that additions to the tabu list are made. Whenever the best neighbor is selected, all edges in the key path that is going to be dropped are designated as tabu (or tabu active) for number of iterations equal to the tabu tenure. The tabu list is used in the second step. When we are finding the shortest path that connects the two fragments we check the tabu status of each edge. If all the edges in the shortest path are in the tabu list then that neighbor is discarded and we go on to the next best neighbor. Thus any key path that has been dropped cannot enter the solution for at least *tt* number of moves, where *tt* represents the tabu tenure.

The aspiration criterion is used to override the tabu restriction. If a particular neighbor has total edge length less than the edge length of the best solution obtained till that point in the search then we go to that neighbor irrespective of whether the move is tabu.

3.7 Diversification strategy

Even with the implementation of tabu lists and tabu tenure, there might be some regions in the graph which are not explored at all by the tabu search. To force the search into unexplored regions of the solution space we implement a diversification procedure. As the search proceeds, along with the tabu list we maintain another list L. This list consists of all nodes in the graph that were part of some accepted solution. We use this information in the following diversification procedure. First, from the existing solution, we select a key path to drop. Next we look at the list L and choose a node *u* that was never before part of any solution. The two fragments formed after dropping the selected key path are now joined through the shortest path containing node *u*. This ensures that a node that has never been part of a solution enters the solution and hence takes the search into previously unexplored regions. This diversification procedure is triggered when the search does not find an improving solution for a certain number of iterations. We use the notation *maxIterTillDiversify* to represent this number. After the diversification procedure is run the solution obtained is accepted as the current solution and the regular search strategy is resumed. The best value for *maxIterTillDiversify* is determined empirically through computational experiments.

3.8 Starting solution and stopping criterion.

The Tabu search is started with a steiner tree constructed using a heuristic developed by Takahashi and Matsuyama [13] which we previously referred to as the cheapest insertion or the CHINS heuristic. The CHINS heuristic constructs a steiner tree T over the Graph G rooted at a source node s_T. Every run of the Tabu search begins from a solution constructed using the CHINS heuristic.

There are two stopping criteria associated with this tabu search algorithm; a primary criterion related to the diversification procedure and a secondary criterion based on the number of iterations. As described in the previous section, diversification begins by identifying a node that has never been part of an accepted solution. The primary criterion stops the algorithm when the diversification is triggered but no node is available for the diversification procedure. This situation will arise when every node $v \in \mathbf{V}$ has been part of at least one accepted solution and no unexplored regions are available in the search space. Since the diversification procedure is invoked only when the search does not find an improving solution for a certain number of iterations, the stopping criterion is related to the total number of iterations without improvement. This stopping criterion is practical for problems of small size but as the number of nodes in the graph increases, this stopping criterion causes the running time to be very high. The secondary stopping criterion is implemented precisely to prevent this from happening. According to the secondary criterion, the search is terminated once it performs a total number of iterations equal to a prespecified parameter *maxIter*.

Preliminary runs of a few large problem instances were carried out to obtain an estimate of the amount of time taken by the tabu search to complete a given number of iterations. Based on these runs, the value of *maxIter* was fixed at 5000. The secondary stopping criterion can be adjusted or even removed altogether to ensure better solution quality at the cost of execution time.

3.9 Parameter selection

Tabu tenure and the number of iterations before diversification are two important parameters of a tabu search. Here we present the results of experiments conducted to understand how these parameters affect the quality of the solution and to fix the values of these parameters based on empirical evidence obtained through these experiments.

3.9.1 Tabu Tenure

Tabu tenure is an important parameter in the Tabu search algorithm and is an example of how short term memory can be used to guide the search towards better solutions. According to Glover and Laguna [4], the Tabu tenure should be small if the activation rule is too restrictive and vice versa. Also, small Tabu tenure is recognized when cycling occurs during the search. Large Tabu tenure is indicated when the solution quality deteriorates. Effective Tabu tenures have been empirically shown to depend on the nature and size of the problem. For the proposed algorithm, some computational experiments were conducted to try and determine the value of the Tabu tenure for a given problem size. The size of the problem is based on the number of nodes |**V**|, the number of edges |**E**| and the number of terminals |**N**| present in the graph. The steiner tree includes all the terminals and so the size of the solutions will increase with |**N**| for fixed |**V**|. The neighborhood size also increases with the size of the solution tree. Similarly, for a fixed number of nodes, an increase in the number of edges makes the graph dense and increases the number of neighbors that a solution might have. The aim was to try and find a co-relation between the tabu tenure and the quality of the solutions obtained, noting that the best tabu tenure may depend on the two values that would represent the size of the problem: |**E**| and |**N**|.

A set of problem instances was chosen so that it was representative of the various problem sizes in our experiment. For each instance the tabu tenure was kept constant and the tabu search was run with as many restarts as there were nodes in the instance. It is possible that the same initial solution might be obtained from different nodes and so only those restarts were considered which correspond to distinct starting solutions, i.e., gave unique objective values of the initial solution. In other words, for a graph having 80 nodes, the Tabu search was re-started 80 times and during each run it was allowed to run for 200 iterations. The best value obtained during each run was noted. Hence, for a graph having **V** nodes, we obtained at most **V** solutions for a particular value of tabu tenure. The objective values of these solutions were summarized using box plots. One representative box plot is shown in Fig 2. The box plot is plotted for a problem instance having |**V**| = 160, |**E**|= 2544 and |**N**| = 40. The box plot has the objective values on the vertical axis and the tabu tenure values on the horizontal axis. The figure shows that the range which spans the objective values of the best 50% of the solutions shifts downwards with an increase in tabu tenure until T=25 and then moves upwards with further increase in tabu tenure. This indicates an improvement in overall solution quality in the sense that, on the whole, the objective values of the best 50% of the solutions decrease as the tabu tenure increases until T=25, and the objective values increase with further increase in tabu tenure, beyond T=25. If we use a similar argument considering the best 75% of the solutions, then the best results are obtained at T=30.

Experimental runs conducted on various problem instances showed that the range of the objective values of the top 50% solutions as well as the range of objective values for the top 75% of the solutions were comparatively low for tabu tenure values lying between |**N**|/2 and |**N**|, and the best ranges were also observed for some tabu tenure value lying between |**N**|/2 and |**N**|. Thus, a dynamic tabu tenure was implemented. The tabu tenure for each edge that becomes tabu is decided by generating a uniformly distributed random number between |**N**|/2 and |**N**|.

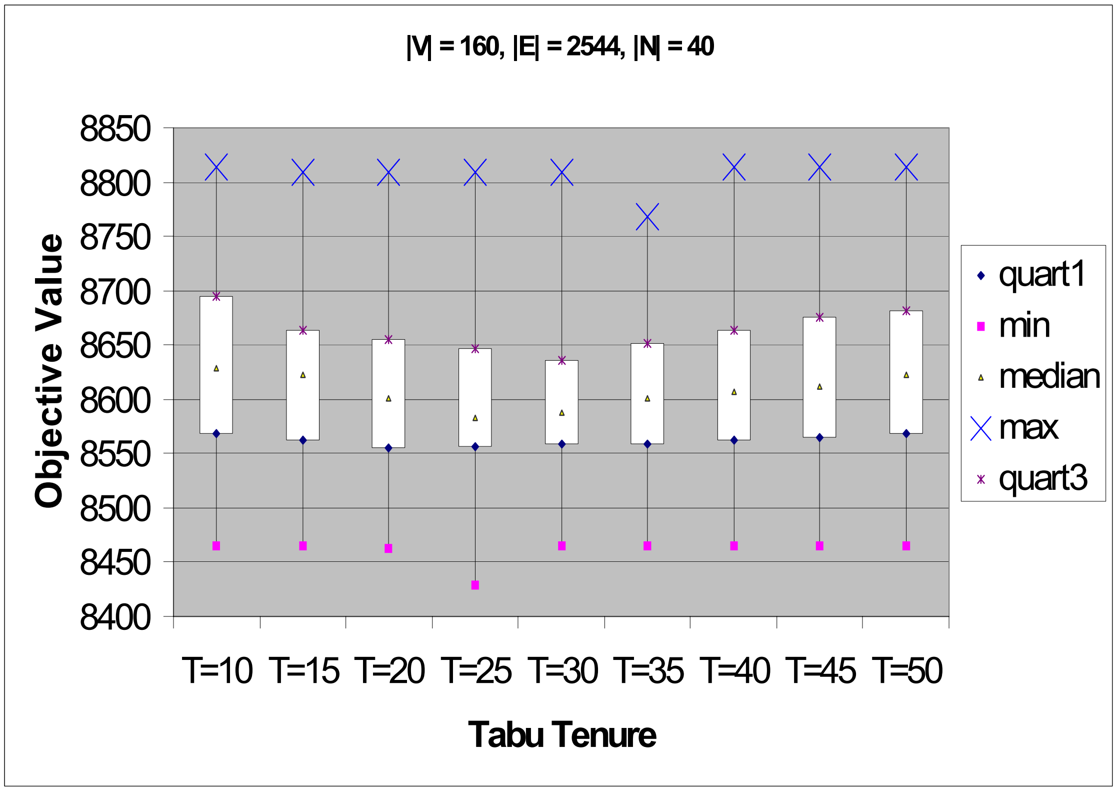

Fig 2 Relating Solution quality with the tabu tenure

3.9.2 Number of iterations before diversification

The diversification procedure is intended to take the tabu search to an unexplored region in the search space. The diversification takes place only if there is no improvement in the best solution found for *maxIterTillDiversify* iterations. It is clear, that if the number of iterations before diversification is too small then the neighborhood of a particular solution will not be explored thoroughly. On the other hand, if the number of iterations is too large, then a large amount of the solution space may remain unexplored by the time the stopping criterion is satisfied.

Some sample runs of the tabu search were performed with the intention of obtaining empirical evidence relating the quality of the solution to the parameter *maxIterTillDiversify*, as was done with the tabu tenure parameter. The tabu search was run with *maxIterTillDiversify* taking values between |**N**| and 5*|**N**| for several problem sets, each

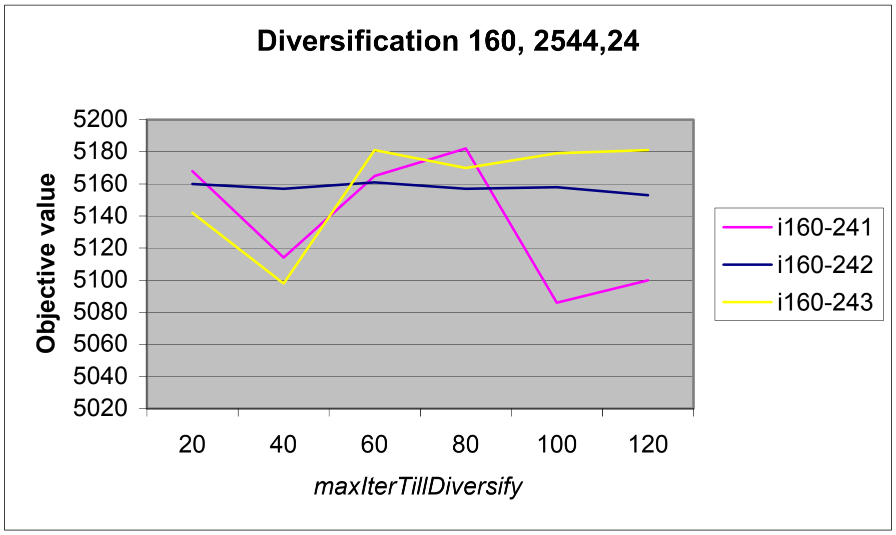

Fig 3 Relating maxIterTillDiversify with solution quality

set consisting of several instances, representing a particular problem size. The plot in Fig 3 is representative of the results obtained for the different problem sets. This plot illustrates the results for one problem set containing three instances of same size (|**V**| = 160, |**E**| = 2544, |**N**| = 24). In the figure, the objective value obtained is plotted on the vertical axis against the parameter *maxIterTillDiversify* on the horizontal axis. The three lines represent three different problem instances, one line for each instance. As seen in the Fig 3, the experimental runs do not indicate any obvious co-relation in the solution quality with change in *maxIterTillDiversify*. Clearly, the observation makes the task of selecting a value for this parameter somewhat difficult. A similar behavior was observed in plots related to other problem sets. Hence, throughout the experiment we fix the value of *maxIterTillDiversify* at 4|**N**|, since this value seems to produce more good quality solutions. Alternatively, instead of keeping the value of this parameter fixed, it can be kept dynamic by assigning it uniformly distributed random values as the tabu search proceeds.

3.10 Pseudo-code for the Tabu search

A pseudo-code for the tabu search is given in Fig 4. Here we explain the code in more detail. The tabu search algorithm begins by calling the function readfile() which reads the test problem from the appropriate file containing the test problem. After reading the data from the file, the function stores the graph as both an adjacency list as well as a weight matrix. Next (in line 3) we run Dijkstra's single source shortest path algorithm from every node in the graph, so as to obtain the shortest distance from any node to every other node in the graph. From line 5, we begin the Tabu search by designating one of the terminal nodes as a *source* node. The Tabu search runs for a fixed number of iterations which is denoted by the parameter *maxIter*. Next we run the cheapest insertion method (a constructive heuristic) to obtain the initial solution. This initial solution will be a tree rooted at the *source* node. The initial solution available through CHINS() is in the form of a parent array data structure. We use this parent array to create an adjacency list structure for the initial steiner tree, in line 8. We now start the iterations of the Tabu search that runs until no improved solutions are obtained during the search for *maxIterWithoutRestart* iterations. This allows us to control the number of restarts. If we want the Tabu Search to run only once then we assign *maxIterWithoutRestart* equal to *maxIter*. In line 12, we run the breadth first search algorithm on the current accepted solution T. Next, in line 16, we create an array that contains all the critical nodes in T.

Beginning from line 17, we evaluate all the neighbors of T and select the best neighbor. A neighbor of T is selected by identifying an unmarked key path P_d in T. We find the length of P_d, mark it and then temporarily remove it from T to obtain two fragments F_1 and F_2. We find the shortest path that connects the two fragments F1 and F2 in line 21. We ensure that the selected path is not Tabu Active and calculate the objective value of the neighbor T' formed by connecting fragments F1 and F2 with the shortest path P_a. In lines 24-27, we check whether P_a satisfies the aspiration criterion. If T' is better than the best neighbor in the current neighborhood search then T' is stored as the best neighbor. This ends the neighborhood search for T. If the total number of iterations without improvement since the last diversification is greater than *maxIterTillDiversify* then we apply the diversification procedure and obtain a T". Finally T' (or T") is made the current solution and all relevant data structures are updated and the objective is recalculated.

Pseudo –code for Tabu Search

```
1.  readFile( )
2.  CreateAdjList( )
3.  Dijkstra(u) $\forall$ u $\in$ $\mathbf{V}$
4.  iTabuIterationNo = 0
5.  While(iTabuIterationNo < maxIter)  //Stopping criteria
6.     source = n, where n $\in$ $\mathbf{N}$
7.     CHINS(source)
8.     createTreeAdjList( )
9.     iterWithoutImp = iterSinceLastDiversify = 0
10.    while(iterWithoutImp < maxIterWithoutRestart)  //restart criteria
11.      Begin 1
12.        T $\leftarrow$ BFS(source)
13.        iTabuIterationNo++
14.        iterWithoutImp++
15.        iterSinceLastDiversify++
16.        create($\mathbf{C}$) such that $\forall$ c $\in$ T and degree(c) > 2, c $\in$ $\mathbf{C}$
17.        Begin nhbd_search
18.            $\forall$ P(u, $a_1$, $a_2$, $a_3$, …, $a_k$, v) such that u, v $\in$ $\mathbf{C}$ and $a_1$, $a_2$, $a_3$, …, $a_k$ $\notin$ $\mathbf{C}$
19.            select and mark $P_d$(u, …, v)
20.            len = length($P_d$) and T = F1 $\cup$ F2 $\cup$ $P_d$
21.            calculate min($P_a$(x, …, y)) such that x $\in$ $F_1$ and y $\in$ $F_2$
22.            check TabuActiveStatus($P_a$)
23.            calculate obj(T') where T' = F1 $\cup$ F2 $\cup$ $P_a$
24.            if obj(T') < obj($T_{bestEver}$)
25.            then
26.                obj($T_{bestEver}$) = obj(T')  //Aspiration criteria
27.                iterWithoutImp = 0;
28.            if obj(T') < obj($T_{bestNbr}$)
29.            then
30.                obj($T_{bestNbr}$) $\leftarrow$ obj(T')
31.                $T_{bestNbr}$ = T'
32.        End nhbd_search
33.        T = $T_{bestNbr}$
34.        If iterSinceLastDiversify >= maxIterTillDiversify
35.        then
36.                iterSinceLastDiversify =0;
37.                diversify( )
38.        update_soln(T')
39.        End 1
```

Fig 4 Pseudo-code for the tabu search

4 Computational Experiments

In this chapter we present the results of computational experiments performed to evaluate the effectiveness of the proposed algorithm on an empirical basis. Henceforth, we refer to the tabu search algorithm that we have developed as TS. The experiments consist of running TS on test problems of various sizes and comparing the objective values of the solutions obtained to the known optimal values. The tabu search algorithm was also compared to the best value obtained from all possible runs of the CHINS heuristic, which is described in Section 2.2.1. The CHINS heuristic is applied repeatedly with each node in the graph serving as the initial node. Thus, CHINS is run on each problem instance |**V**| times and henceforth we will refer to the multiple runs of CHINS as the CHINS-V heuristic. All programs for this algorithm were written in the C programming language and were executed on a Sun Ultra 10 workstation.

4.1 Problem sets

The algorithm TS was tested on problem sets created by C. Duin [17]. These problem instances for the steiner tree problem are of various sizes, types and difficulty and are now available over the Internet. There are two important reasons for using these problem sets. The first one is that these problems have been solved using some exact method such as the branch and bound method described earlier and either the optimal value or an upper bound on the optimal value for each problem is reported. The second important reason is that the problem sets have been created to defy preprocessing. Thus simplistic edge exchange or node exchange procedures cannot be used to optimally solve these instances.

The problem instances can be divided into two main sets based on the total number of nodes in a particular instance. The first set contains instances of smaller size, each containing 80 nodes. The second set has larger instances each having 160 nodes. Each of these sets can be further broken down into subsets based on the edge density. Thus, each set has problem instances having five different edge densities $\mathbf{e} = 3\mathbf{v}/2$, $2\mathbf{v}$, $\mathbf{v}\ln\mathbf{v}$, $2\mathbf{v}\ln\mathbf{v}$, $\mathbf{v}(\mathbf{v}-1)/2$ and 4 different terminal densities $\mathbf{n} = [\log_2\mathbf{v}]$, $[\sqrt{\mathbf{v}}]$, $2[\log_2\mathbf{v}]$, $[\mathbf{v}/4]$, where $\mathbf{e} = |\mathbf{E}|$, $\mathbf{n} = |\mathbf{N}|$ and $\mathbf{v} = |\mathbf{V}|$. The source of the data also indicates the difficulty of the instance in terms of the type of algorithm required to solve the problem and the order of magnitude of time taken.

4.2 Performance measures

The tabu search algorithm is evaluated based on two performance measures. The first measure compares the objective value of the solution obtained by TS for an instance to its optimal value. This comparison is done with the help of the quantity η_{opt} which is defined as

$$\eta_{opt} = \left(\frac{C_{TS} - C_{opt}}{C_{opt}} \right) * 100$$

where C_{opt} = optimum value
and C_{TS} = objective value obtained by TS

Thus, η_{opt} is the percentage difference between the tabu search value and the optimum value for an instance. The second measure, $\eta_{CHINS\text{-}V}$, compares the objective value of the solution obtained via TS with that of the best solution obtained by CHINS-V.

$$\eta_{CHINS} = \left(\frac{C_{CHINS} - C_{TS}}{C_{opt}} \right) * 100$$

where C_{opt} = optimum value
C_{TS} = objective value obtained using TS
and C_{CHINS} = objective value using CHINS-V

Thus, η_{CHINS} is the difference between the objective value obtained using the CHINS-V heuristic and the tabu search value as a percentage of the optimum value for an instance.

4.3 Observations

All numerical results were obtained with the following parameter settings: tabu tenure was a uniformly distributed random number between |**N**|/2 and |**N**|, the search was stopped when no more nodes were available for the diversification procedure or when the search had performed 5000 iterations. The diversification procedure was performed whenever the search executed 4*|**N**| iterations without improvement in the best value obtained. Also, every 1000 iterations the search was restarted with a new initial solution generated using the CHINS heuristic.

4.3.1 Comparison to Optimal Values

The problem instances are categorized into two sets based on the number of nodes |**V**|. Let us refer to the problem set containing instances with 80 nodes as i080. Similarly i160 refers to the problem set containing instances having 160 nodes. The results for each set are summarized in separate tables. The comparison of the TS solutions to the known optimal values is displayed in Table 1 and Table 2. Table 1 contains results of the problem instances from the set i080 and Table 2 contains results from the set i160. The size of the problem is characterized by the number of nodes |**V**|, number of edges |**E**| and the number of terminals |**N**|. There are at most 5 instances of the same size, i.e., instances with the same values of |**V**|, |**E**| and |**N**|. Each row in Table 1 and Table 2 gives the average value of η_{opt} and the average time per instance for all instances of the same size. The tables also contain the total number of optimum solutions found by TS for each problem size. At the end of each table we present a summary of the results for all problem instances for that particular set.

For set i080, TS found the optimum value in 63 out of the 88 problem instances tested. On the other hand for set i160, TS found the optimum in 46 out of the 94 instances. On average, TS produced solutions, for set i080, which were within 0.38% of the optimal. For set i160, TS came up with solutions that were within 0.54% of the optimal. The maximum deviation from the optimal is 5.51% for i080 and 5.89% for i160. The time reported in the tables is the total cpu time taken in seconds and includes the time required to run *Dijkstra's all-source shortest path* algorithm. The time required for the tabu search algorithm increased with the problem size. Keeping the number of nodes constant, the increase in time was more sensitive to an increase in the number of terminals than an increase in the number of edges. This is to be expected, as the neighborhood search takes up a large fraction of the time required for TS. Hence, the time required for TS increases with neighborhood size. As we explained earlier in Section 3.3, the size of the neighborhood is determined by the number of vertices in the solution T, more accurately, the number of critical vertices in the solution. An increase in the number of terminals will definitely lead to an increase in the number of critical vertices in the solution. On the other hand, an increase in the number of edges (for the same |**V**| and |**N**|) may not lead to an increase of the number of critical vertices in the solution.

The results for both sets i080 and i160 indicate that with an increase in the number of terminals |**N**| there is a decrease in the number of optimal solutions found by TS. Also, the deviation from the optimal value of solutions found by TS increases with an increase in |**N**|. In many cases, TS finds the optimal in all but one instance. The deviation from the optimal value for this instance may be as large as 5%. Moreover, in problem sets having the same value of |**V**| and |**N**|, an increase in |**E**| does not increase the deviation from the optimal. This indicates that the performance of the algorithm depends not only on the size but also the structure of the instance. Thus, there seems to be a possibility that instances with specific structure could be created for which TS would perform poorly as compared to other instances of the same size.

Table 1 Summary of Results for problem set i080

Size			Number of Instances	Number of Optimums	Average η_{opt}	Average Time(secs)
|**V**|	|**E**|	|**N**|				
80	120	6	5	4	1.05%	2.14
80	350	6	5	4	1.12%	1.34
80	3160	6	5	5	0.00%	1.68
80	160	6	5	5	0.00%	1.56
80	632	6	5	4	0.05%	1.56
80	120	8	3	3	0.00%	3.00
80	350	8	3	3	0.00%	3.27
80	3160	8	4	4	0.00%	3.00
80	160	8	4	4	0.00%	2.20
80	632	8	5	4	0.01%	2.62
80	120	16	4	4	0.00%	11.05
80	350	16	5	3	0.66%	12.02
80	3160	16	5	0	0.74%	16.02
80	160	16	5	5	0.00%	14.06
80	632	16	5	1	0.86%	11.86
80	120	20	2	2	0.00%	-
80	350	20	5	4	0.36%	18.64
80	3160	20	5	0	0.65%	28.54
80	160	20	4	3	0.28%	18.30
80	632	20	4	1	1.18%	18.00

TS on i080

Total Instances	88		η_{opt}	Time(sec)
Total optimums	63	Maximum	5.51%	29.7
		Average	0.38%	9.09
		Minimum	0.00%	1.2

Table 2 Summary of Results for problem set i160

Size			Total Instances	No of Optimums	Average $\eta_{CHINS-V}$	Average Time(sec)
|V|	**|E|**	**|N|**				
160	240	7	5	5	0.00%	11.00
160	812	7	5	5	0.00%	8.90
160	12720	7	5	4	0.04%	9.78
160	320	7	5	5	0.00%	8.83
160	2544	7	4	4	0.00%	8.53
160	240	12	3	2	1.96%	35.40
160	812	12	5	2	0.49%	30.00
160	12720	12	5	2	0.26%	28.48
160	320	12	5	5	0.00%	38.48
160	2544	12	5	2	0.58%	25.78
160	240	24	5	4	0.16%	155.06
160	812	24	5	1	0.66%	119.32
160	12720	24	5	0	0.73%	87.30
160	320	24	4	2	0.51%	125.95
160	2544	24	5	0	1.53%	100.72
160	240	40	5	3	0.20%	313.72
160	812	40	5	0	1.73%	293.40
160	12720	40	5	0	0.74%	179.32
160	320	40	5	0	0.34%	334.68
160	2544	40	3	0	1.77%	192.70

				$\eta_{CHINS-V}$	Time
TS on i160	Total Instances	94			
	Total optimums	46	Max	5.89%	375.2
			Average	0.54%	106.85
			Min	0.00%	6.9

4.3.2 Comparison of CHINS-V with TS

The heuristic that we refer to as CHINS-V in this section is the multiple pass version of the CHINS heuristic that is described in Section 2.2.1. The heuristic is run |**V**| times starting from a different node in every run. The CHINS-V heuristic is run on every problem instance in the experiment. Tables 3a and 3b contain the comparison between CHINS-V and TS for problem set i080, while the results for set i160 are in Tables 4a and Table 4b.

As done in Table 1 and 2, the problem instances are grouped according to size and each row contains the summary for instances of the same size. Firstly, TS and CHINS-V are compared on the basis of the number of instances for which the optimal solution was found by each method. The difference between the CHINS-V values and the TS values is noted. Along with the number of optimums, Table 3a, for set i080, and Table 4a, for set i160, show the maximum and the minimum values of the difference between CHINS-V and TS for problem instances of the same size. The tables also indicate the number of instances in which TS performed better, as good as, or worse than CHINS-V. Table 3b displays a summary for all of the instances in i080. We divide the problem instances into two groups. One group contains all the instances where TS outperformed CHINS-V and the other contains all those instances where CHINS-V outperformed TS. The maximum difference and the average difference between TS and CHINS-V for these two groups are presented in Table 3b for the set i080 and Table 4b for set i160.

For the problem set i080, out of the 88 problem instances TS finds the optimal in 63 instances whereas CHINS-V finds the optimal in only 37 cases. With the exception of 3 problem sizes (all instances having same values of |**E**|, |**V**| and |**N**|), in all other problem sizes TS outperforms CHINS-V. Altogether in set i080, TS is better in a total of 47 instances and on an average the difference is 2.23% of the optimum. On the other hand, CHINS-V is better in only 12 instances and the average difference of 0.72% in these 11 instances is comparatively much less. The numbers for the set i160 indicate a similar trend. In set i160, apart from three problem sizes, TS outperforms CHINS-V in all other problem sizes. Out of the 94 instances tested, TS performs better in 61 with an average improvement over CHINS-V of 2.25% while CHINS-V does better in only 15 instances and with a comparatively lower average improvement over TS of 0.53%.

Thus for the 182 instances tested, the tabu search algorithm TS, obtains the optimal solutions in 108 instances. On average, TS produces solutions within 0.46% of the optimal value. Also, even though CHINS-V is much faster, TS produces better solutions in most cases and in those cases where TS performs worse than CHINS-V, the average difference in the values is 0.6% of the optimal values.

Table 3a Comparison of CHINS-V with TS for set i080

Size			Total Instances	No of Optimals		η_{CHINS}		Number of instances where		
|**V**|	|**E**|	|**N**|		TS	Chins	Max	Min	$C_{TS} < C_{CHINS-V}$	$C_{TS} = C_{CHINS-V}$	$C_{TS} > C_{CHINS-V}$
80	120	6	5	4	3	0.17%	-0.93%	1	3	1
80	350	6	5	4	1	0.34%	0.00%	3	2	0
80	3160	6	5	5	5	0.00%	0.00%	0	5	0
80	160	6	5	5	4	0.57%	0.00%	1	4	0
80	632	6	5	4	2	6.25%	0.00%	2	2	1
80	120	8	3	3	2	3.22%	0.00%	1	2	0
80	350	8	3	3	0	4.30%	0.11%	3	0	0
80	3160	8	4	4	4	0.00%	0.00%	0	4	0
80	160	8	4	4	3	4.42%	0.00%	1	3	0
80	632	8	5	4	3	3.30%	0.00%	1	4	0
80	120	16	4	4	0	2.34%	2.03%	4	0	0
80	350	16	5	3	0	4.27%	0.52%	5	0	0
80	3160	16	5	0	5	-0.06%	-1.62%	0	0	5
80	160	16	5	5	0	4.57%	1.58%	5	0	0
80	632	16	5	1	0	4.17%	0.06%	5	0	0
80	120	20	2	2	0	3.45%	1.25%	2	0	0
80	350	20	5	4	0	3.88%	1.58%	5	0	0
80	3160	20	5	0	5	-0.43%	-0.81%	0	0	5
80	160	20	4	3	0	3.37%	0.23%	4	0	0
80	632	20	4	1	0	2.81%	0.44%	4	0	0
Total			**88**	**61**	**37**			**47**	**29**	**12**

Table 3b Summary for set i080

	Instances from set i080 where	
	TS performs better	CHINS-V performed better
Max	6.25%	1.62%
Avg.	2.23%	0.72%

Table 4a Comparison between CHINS-V and TS for set i160

Size			Total Instances	No of Optimals		η_{CHINS}		Number of instances where		
|V|	|E|	|T|		TS	Chins	Max	Min	$C_{TS} < C_{CHINS-V}$	$C_{TS} = C_{CHINS-V}$	$C_{TS} > C_{CHINS-V}$
160	240	7	5	5	2	3.84%	0.00%	3	2	0
160	812	7	5	5	0	5.00%	0.51%	5	0	0
160	12720	7	5	4	5	0.00%	-0.22%	0	4	1
160	320	7	5	5	4	0.14%	0.00%	1	4	0
160	2544	7	4	4	2	3.28%	0.00%	2	2	0
160	240	12	3	2	1	2.05%	0.00%	2	1	0
160	812	12	4	2	0	5.26%	0.14%	4	0	0
160	12720	12	5	2	5	0.00%	-0.72%	0	2	3
160	320	12	5	5	1	2.53%	0.00%	4	1	0
160	2544	12	5	2	0	3.68%	0.00%	4	1	0
160	240	24	5	4	0	3.18%	1.35%	5	0	0
160	812	24	5	1	0	4.60%	2.34%	5	0	0
160	12720	24	5	0	5	-0.21%	-1.06%	0	0	5
160	320	24	4	2	0	4.18%	1.20%	4	0	0
160	2544	24	5	0	0	1.36%	-0.14%	4	0	1
160	240	40	5	3	0	1.91%	1.60%	5	0	0
160	812	40	5	0	0	2.67%	1.79%	5	0	0
160	12720	40	5	0	0	0.11%	-0.94%	1	0	4
160	320	40	5	0	0	4.08%	2.66%	5	0	0
160	2544	40	3	0	0	1.32%	-0.20%	2	0	1

Total	**94**	**46**	**25**			**61**	**17**	**15**

Table 4b for set i160

	Instances from set i160 where	
	TS performs better	CHINS-V performed better
Max	5.26%	1.06%
Avg.	2.25%	0.53%

4.4 Concluding Remarks

To assess the efficiency of the proposed tabu search algorithm, it was applied to problem sets specifically built for testing algorithms for the steiner tree problem [17]. The algorithm was also compared with a multiple pass version of the CHINS heuristic. The proposed tabu search algorithm produces optimal solutions in 59% of the instances tested, and on average produces solutions within 0.46% of the optimal value. The tabu search also performs better than or as good as CHINS-V in 85.2% of the instances and in the 14.8% cases where it does worse than CHINS-V the difference in objective values, on average, is just 0.6%.

5 Summary and avenues for future work

We have presented a new tabu search heuristic for the steiner tree problem in undirected graphs. The algorithm is based on a neighborhood structure involving exchange of key paths which is the key difference between this algorithm and all previously developed search procedures. The experimental results presented in Chapter 4 show that the tabu search algorithm is reasonably successful. It produces near-optimal solutions in the experiments conducted and it out-performs the CHINS-V heuristic.

There is a lot of scope for carrying out more experiments as well as further investigations related to the tabu search algorithm developed here. The ideas for extending this research can be divided into three broad categories: further experiments, alternative neighborhood structures and variations/modifications of the tabu search algorithm TS.

Further experimentation: The experiments for testing this algorithm have been performed on two sets of problem instances. The experimentation can be extended to problem sets containing instances that are larger in size. Moreover, other problem sets exist for the testing of algorithms for the steiner tree problem. These instances are different from the problem sets used to test algorithm TS in the sense that different ideas have been used for the creation of these instances. The present algorithm has been compared to known optimal values and the CHINS-V heuristic method. Comparisons with other algorithms can also be made by running TS on problem sets that have been used for testing other heuristic methods.

Modifications to the tabu search: The tabu search algorithm itself maybe modified with respect to the values of the selected parameters like *tabu tenure* and *iterations before diversification*. Other ideas for the stopping criterion as well as various modifications of memory structures like the tabu list can also be investigated. There are many ideas for implementing a diversification procedure. In the current algorithm, diversification is achieved by dropping a key path and the fragments formed are reconnected through a shortest path that runs through a node that has never been part of an accepted solution. Alternatively, two random nodes can be selected, one in each of the fragments, and a key path connecting these two nodes could be added to the solution. Here the fragments may get re-connected using a key path that has higher cost than the shortest key-path connecting the two fragments. This might introduce edges and nodes that previously did not enter the solution because reconnection of the fragments is always done using the shortest path, leading to diversification.

Alternative Neighborhood Structure and Search Strategy: There are many variations of the neighborhood structure and the search strategy implemented in the current algorithm and these could also be investigated further. For instance, in the current algorithm TS, a neighbor is obtained by the removal of one key path from the current solution. This causes two fragments to be formed that are then optimally re-connected by adding a key path. Instead, a neighbor of the current solution could be obtained by the removal of more than one key path that will lead to the creation of multiple fragments. The reconnection of these fragments would require more than one key path to enter the solution. This is an alternative neighborhood structure that can be explored. Furthermore, in the key path exchange idea used here dropping of the key path is performed first and then a key path is selected for addition. This procedure could be reversed and a key path could be added first and the path to be dropped can be selected from the cycle formed by the addition of the key path.

References

1. Downsland, K. A., “Hill-Climbing, Simulated annealing and the steiner problem in Graphs”, *Engineering Optimization* **17** (1991) 91-107.
2. Duin Cees and Stefan Voss, “The Pilot Method: A Strategy for heuristic repetition with application to steiner problems in graphs”, *Networks* **34** (1999), 181 - 191
3. Esbensen, H., “Computing near –optimal Solutions to the steiner problem in a Graph using a genetic algorithm”, *Networks*, **26** (1995), 173-185.
4. Gendreau, M, J. –F. Larochelle, and B. Sanso, “A Tabu Search Heuristic for the Steiner Tree Problem”, *Networks*, **34** (1999), 162-172.
5. Hakimi, S. L., “Steiner’s problem in graphs and its implications”, *Networks*, **1** (1971) 113-133.
6. Hwang, F. K., D. S. Richards and Pawel Winter, The Steiner Tree Problem, North Holland, Amsterdam, 1992.
7. Kapsalis, A., V.J. Rayward-Smith, and G. D. Smith, Solving the Graphical Steiner Tree Problem using genetic algorithms, *J Oper Res Soc* **44** (1993), 397-406.
8. Lawler, E. L., Combinatorial optimisation: networks and matroids, Holt, Rinehart, and Winston, N.Y. (1976).
9. Ramanathan, S., “Multicast Tree generation in Networks With Asymmetric Links”, *IEEE/ACM Transactions On Networking*, **4** (1996), 558-568.
10. Ribeiro, C. C., M. C. De Souza, “Tabu Search for the steiner problem in Graphs”, *Networks*, **36** (2000), 138 – 146.

11. Shore, M. L., Foulds, L. R., Gibbons, P. B., “An Algorithm for the steiner problem in Graphs”, *Networks*, **12** (1982), 323 – 333.

12. Sondergeld, L., and Stefan Voss, “A multi-level star shaped intensification diversification approach in tabu search for the steiner tree problem in graphs“, Working Paper, TU Braunschweig(1996)

13. Takahashi, H., and A. Matsuyama, An approximate solution for the steiner problem in graphs, Math. Jap. **24** (1983) 573 – 577.

14. Voss Stefan, “Modern Heuristic Search methods for the Steiner Tree Problem in Graphs”, Kluiver Academic Publishers, Amsterdam, 2000.

15. Wade A. S. C. and V. J. Rayward-Smith, “Effective Local Search Techniques for the Steiner Tree Problem”, *Studies in Locational Analysis* **11** (1997), 219 – 241.

16. Zelikovsky, A. Z., “An 11/6-approximation algorithm for the steiner problem in Graphs”, J. Nesetril and M. Fiedler (eds.), Fourth Czechoslavokian symposium on Combinatorics, Graphs and Complexity, (Elsevier, Amsterdam, 1992) pp. 351-354.

17. http://elib.zib.de/steinlib/testset.php

Appendix 1A

Complete results for set i080

Instance	|V|	|E|	|N|	Optimal	Tabu	CHINS	% diff from Opt	% diff from CHINS-V	Time (TS)
080-001	80	120	6	1787	1787	1787	0.00	0.00	2.7
080-002	80	120	6	1607	1691	1676	5.23	-0.93	2.1
080-003	80	120	6	1713	1713	1713	0.00	0.00	2.2
080-004	80	120	6	1866	1866	1866	0.00	0.00	1.7
080-005	80	120	6	1790	1790	1793	0.00	0.17	2.0
080-011	80	350	6	1479	1479	1482	0.00	0.20	1.7
080-012	80	350	6	1484	1484	1489	0.00	0.34	1.2
080-013	80	350	6	1381	1381	1381	0.00	0.00	1.3
080-014	80	350	6	1397	1474	1478	5.51	0.29	1.3
080-015	80	350	6	1495	1496	1496	0.07	0.00	1.2
080-021	80	3160	6	1175	1175	1175	0.00	0.00	1.2
080-022	80	3160	6	1178	1178	1178	0.00	0.00	1.9
080-023	80	3160	6	1174	1174	1174	0.00	0.00	1.8
080-024	80	3160	6	1161	1161	1161	0.00	0.00	1.9
080-025	80	3160	6	1162	1162	1162	0.00	0.00	1.6
080-031	80	160	6	1570	1570	1570	0.00	0.00	1.5
080-033	80	160	6	1794	1794	1794	0.00	0.00	1.7
080-034	80	160	6	1688	1688	1688	0.00	0.00	1.3
080-035	80	160	6	1862	1862	1862	0.00	0.00	1.6
080-041	80	632	6	1276	1279	1279	0.24	0.00	1.2
080-042	80	632	6	1287	1287	1287	0.00	0.00	2.1
080-043	80	632	6	1295	1295	1376	0.00	6.25	1.3
080-044	80	632	6	1366	1366	1366	0.00	0.00	1.3
080-045	80	632	6	1310	1310	1362	0.00	3.97	1.9
080-101	80	120	8	2608	2608	2608	0.00	0.00	2.1
080-102	80	120	8	2403	2403	2403	0.00	0.00	4.1
080-104	80	120	8	2486	2486	2566	0.00	3.22	2.8
080-111	80	350	8	2051	2051	2059	0.00	0.39	4.1
080-113	80	350	8	1884	1884	1965	0.00	4.30	3.4
080-115	80	350	8	1868	1868	1870	0.00	0.11	2.3
080-121	80	3160	8	1561	1561	1561	0.00	0.00	2.8
080-123	80	3160	8	1569	1569	1569	0.00	0.00	3
080-124	80	3160	8	1555	1555	1555	0.00	0.00	3.3
080-125	80	3160	8	1572	1572	1572	0.00	0.00	2.9
080-131	80	160	8	2284	2284	2284	0.00	0.00	2.2
080-133	80	160	8	2261	2261	2261	0.00	0.00	2.3
080-135	80	160	8	2102	2102	2102	0.00	0.00	2.1
080-141	80	632	8	1788	1788	1847	0.00	3.30	2
080-142	80	632	8	1708	1708	1708	0.00	0.00	3.7
080-143	80	632	8	1767	1768	1768	0.06	0.00	3.5
080-144	80	632	8	1772	1772	1772	0.00	0.00	1.9

080-145	80	632	8	1762	1762	1762	0.00	0.00	2
080-201	80	120	16	4760	4760	4862	0.00	0.00	2.14
080-203	80	120	16	4599	4599	4697	0.00	0.00	2.13
080-204	80	120	16	4492	4492	4583	0.00	0.00	2.03
080-205	80	120	16	4564	4564	4671	0.00	0.00	2.34
080-211	80	350	16	3631	3631	3786	0.00	4.27	11
080-212	80	350	16	3677	3746	3887	1.88	3.83	11
080-213	80	350	16	3678	3731	3750	1.44	0.52	9
080-214	80	350	16	3734	3734	3834	0.00	2.68	12.7
080-215	80	350	16	3681	3681	3760	0.00	2.15	16.4
080-221	80	3160	16	3158	3160	3158	0.06	-0.06	15.5
080-222	80	3160	16	3141	3187	3141	1.46	-1.46	16.1
080-223	80	3160	16	3156	3164	3156	0.25	-0.25	16.2
080-224	80	3160	16	3159	3169	3159	0.32	-0.32	16.1
080-225	80	3160	16	3150	3201	3150	1.62	-1.62	16.2
080-231	80	160	16	4354	4354	4456	0.00	2.34	14.9
080-232	80	160	16	4199	4199	4391	0.00	4.57	14
080-233	80	160	16	4118	4118	4243	0.00	3.04	14.5
080-234	80	160	16	4274	4274	4365	0.00	2.13	12.8
080-235	80	160	16	4487	4487	4558	0.00	1.58	14.1
080-241	80	632	16	3538	3547	3551	0.25	0.11	13.4
080-242	80	632	16	3458	3519	3608	1.76	2.57	11.1
080-243	80	632	16	3474	3474	3619	0.00	4.17	10.7
080-244	80	632	16	3466	3473	3592	0.20	3.43	13.7
080-245	80	632	16	3467	3539	3541	2.08	0.06	10.4
080-302	80	120	20	5944	5944	6149	0.00	3.45	9.2
080-305	80	120	20	5932	5932	6006	0.00	1.25	9.8
080-311	80	350	20	4554	4636	4708	1.80	1.58	16.6
080-312	80	350	20	4534	4534	4710	0.00	3.88	22.4
080-313	80	350	20	4509	4509	4683	0.00	3.86	18
080-314	80	350	20	4515	4515	4636	0.00	2.68	16.4
080-315	80	350	20	4459	4459	4540	0.00	1.82	19.8
080-321	80	3160	20	3932	3956	3932	0.61	-0.61	27.6
080-322	80	3160	20	3937	3961	3937	0.61	-0.61	28.5
080-323	80	3160	20	3946	3963	3946	0.43	-0.43	28.3
080-324	80	3160	20	3932	3964	3932	0.81	-0.81	28.6
080-325	80	3160	20	3924	3954	3924	0.76	-0.76	29.7
080-331	80	160	20	5226	5226	5402	0.00	3.37	12.9
080-333	80	160	20	5381	5441	5540	1.12	1.84	20.9
080-334	80	160	20	5264	5264	5276	0.00	0.23	19.9
080-335	80	160	20	4953	4953	5047	0.00	1.90	19.5
080-341	80	632	20	4236	4302	4332	1.56	0.71	17
080-342	80	632	20	4337	4414	4433	1.78	0.44	11.3
080-343	80	632	20	4246	4305	4424	1.39	2.80	24.4
080-344	80	632	20	4310	4310	4431	0.00	2.81	19.3

Appendix 1B

Complete results for set i160

Instance	|V|	|E|	|N|	Optimal	Tabu	CHINS	% diff from Opt	% diff from CHINS-V	Time (TS)
i160-001	160	240	7	2490	2490	2496	0.00	0.24	10.9
i160-002	160	240	7	2158	2158	2158	0.00	0.00	11.8
i160-003	160	240	7	2297	2297	2297	0.00	0.00	12.6
i160-004	160	240	7	2370	2370	2461	0.00	3.84	10.9
i160-005	160	240	7	2495	2495	2499	0.00	0.16	8.8
i160-011	160	812	7	1677	1677	1757	0.00	4.77	7.1
i160-012	160	812	7	1750	1750	1773	0.00	1.31	9.3
i160-013	160	812	7	1661	1661	1744	0.00	5.00	7.7
i160-014	160	812	7	1778	1778	1787	0.00	0.51	10.6
i160-015	160	812	7	1768	1768	1845	0.00	4.36	9.8
i160-021	160	12720	7	1352	1352	1352	0.00	0.00	10.7
i160-022	160	12720	7	1365	1368	1365	0.22	-0.22	10.3
i160-023	160	12720	7	1351	1351	1351	0.00	0.00	10.3
i160-024	160	12720	7	1371	1371	1371	0.00	0.00	9.5
i160-025	160	12720	7	1366	1366	1366	0.00	0.00	8.1
i160-031	160	320	7	2170	2170	2170	0.00	0.00	-
i160-032	160	320	7	2330	2330	2330	0.00	0.00	8.9
i160-033	160	320	7	2101	2101	2101	0.00	0.00	8.3
i160-034	160	320	7	2083	2083	2086	0.00	0.14	11.2
i160-035	160	320	7	2103	2103	2103	0.00	0.00	6.9
i160-041	160	2544	7	1494	1494	1543	0.00	3.28	8.7
i160-042	160	2544	7	1486	1486	1486	0.00	0.00	8.6
i160-043	160	2544	7	1549	1549	1565	0.00	1.03	7.2
i160-045	160	2544	7	1554	1554	1554	0.00	0.00	9.6
i160-101	160	240	12	3859	3859	3859	0.00	0.00	43.4
i160-102	160	240	12	3747	3747	3824	0.00	2.05	35.6
i160-103	160	240	12	3837	4063	4069	5.89	0.16	27.2
i160-111	160	812	12	2869	2924	3075	1.92	5.26	29.7
i160-113	160	812	12	2866	2866	2935	0.00	2.41	31.7
i160-114	160	812	12	2989	2992	3031	0.10	1.30	35.4
i160-115	160	812	12	2937	2937	2941	0.00	0.14	29.5
i160-121	160	12720	12	2363	2364	2363	0.04	-0.04	31.1
i160-122	160	12720	12	2348	2348	2348	0.00	0.00	31.6
i160-123	160	12720	12	2355	2355	2355	0.00	0.00	27.2
i160-124	160	12720	12	2352	2369	2352	0.72	-0.72	26.7
i160-125	160	12720	12	2351	2364	2351	0.55	-0.55	25.8
i160-131	160	320	12	3356	3356	3356	0.00	0.00	36.1
i160-132	160	320	12	3450	3450	3471	0.00	0.61	47.2
i160-133	160	320	12	3585	3585	3675	0.00	2.51	42.1
i160-134	160	320	12	3470	3470	3481	0.00	0.32	27.4
i160-135	160	320	12	3716	3716	3810	0.00	2.53	39.6
i160-141	160	2544	12	2549	2549	2637	0.00	3.45	27.6
i160-142	160	2544	12	2562	2597	2597	1.37	0.00	26.2
i160-143	160	2544	12	2557	2557	2651	0.00	3.68	23.3
i160-144	160	2544	12	2607	2612	2694	0.19	3.15	26.4

i160-145	160	2544	12	2578	2612	2642	1.32	1.16	25.4
i160-201	160	240	24	6923	6923	7143	0.00	3.18	180
i160-202	160	240	24	6930	6930	7043	0.00	1.63	148.7
i160-203	160	240	24	7243	7302	7417	0.81	1.59	139
i160-204	160	240	24	7068	7068	7276	0.00	2.94	153.5
i160-205	160	240	24	7122	7122	7218	0.00	1.35	154.1
i160-211	160	812	24	5583	5583	5752	0.00	3.03	124.4
i160-212	160	812	24	5643	5651	5783	0.14	2.34	123.7
i160-213	160	812	24	5647	5720	5861	1.29	2.50	118.2
i160-214	160	812	24	5720	5744	5975	0.42	4.04	114.8
i160-215	160	812	24	5518	5597	5851	1.43	4.60	115.5
i160-221	160	12720	24	4729	4761	4729	0.68	-0.68	94.8
i160-222	160	12720	24	4697	4741	4697	0.94	-0.94	86.1
i160-223	160	12720	24	4730	4780	4730	1.06	-1.06	83.1
i160-224	160	12720	24	4721	4758	4721	0.78	-0.78	83.7
i160-225	160	12720	24	4728	4738	4728	0.21	-0.21	88.8
i160-231	160	320	24	6662	6723	6803	0.92	1.20	112.1
i160-232	160	320	24	6558	6558	6832	0.00	4.18	114.5
i160-233	160	320	24	6339	6411	6586	1.14	2.76	152.1
i160-235	160	320	24	6764	6764	6934	0.00	2.51	125.1
i160-241	160	2544	24	5086	5182	5175	1.89	-0.14	99.1
i160-242	160	2544	24	5106	5153	5158	0.92	0.10	109.1
i160-243	160	2544	24	5050	5181	5188	2.59	0.14	97.6
i160-244	160	2544	24	5076	5163	5186	1.71	0.45	99
i160-245	160	2544	24	5084	5111	5180	0.53	1.36	98.8
i160-301	160	240	40	11816	11816	12005	0.00	1.60	312.5
i160-302	160	240	40	11497	11497	11715	0.00	1.90	232.2
i160-303	160	240	40	11445	11445	11662	0.00	1.90	306.1
i160-304	160	240	40	11448	11527	11746	0.69	1.91	375.2
i160-305	160	240	40	11423	11456	11657	0.29	1.76	342.6
i160-311	160	812	40	9135	9264	9508	1.41	2.67	318.5
i160-312	160	812	40	9052	9219	9411	1.84	2.12	279.5
i160-313	160	812	40	9159	9308	9482	1.63	1.90	288.7
i160-314	160	812	40	8941	9148	9321	2.32	1.93	287.7
i160-315	160	812	40	9086	9216	9379	1.43	1.79	292.6
i160-321	160	12720	40	7876	7956	7903	1.02	-0.67	196.9
i160-322	160	12720	40	7859	7895	7892	0.46	-0.04	172.8
i160-323	160	12720	40	7876	7957	7883	1.03	-0.94	178.2
i160-324	160	12720	40	7884	7903	7912	0.24	0.11	175.2
i160-325	160	12720	40	7862	7938	7877	0.97	-0.78	173.5
i160-331	160	320	40	10414	10487	10912	0.70	4.08	354.3
i160-332	160	320	40	10806	10819	11241	0.12	3.91	335.1
i160-333	160	320	40	10561	10569	10850	0.08	2.66	337.8
i160-334	160	320	40	10327	10405	10758	0.76	3.42	314.2
i160-335	160	320	40	10589	10594	10928	0.05	3.15	332
i160-341	160	2544	40	8331	8530	8513	2.39	-0.20	224.5
i160-342	160	2544	40	8348	8460	8527	1.34	0.80	165.2
i160-343	160	2544	40	8275	8406	8515	1.58	1.32	188.4