# An Efficient Algorithm for Generalized Minimum Spanning Tree Problem

He Jiang
School of Software
Dalian University of Technology
Dalian, 116621 China
jianghe@dlut.edu.cn

Yudong Chen
School of Software
Dalian University of Technology
Dalian, 116621 China
chenyudong@mail.dlut.edu.cn

## ABSTRACT

The Generalized Minimum Spanning Tree problem (GMST) has attracted much attention during the last few years. Since it is intractable, many heuristic algorithms have been proposed to solve large GMST instances. Motivated by the effectiveness and efficiency of the muscle (the union of all optimal solutions) for solving other NP-hard problems, we investigate how to incorporate the muscle into heuristic design for GMST. Firstly, we demonstrate that it's NP-hard to obtain the muscle for GMST. Then we show that the muscle can be well approximated by the principle and subordinate candidate sets, which can be calculated on a reduced version of GMST. Therefore, a Dynamic cAndidate set based Search Algorithm (DASA) is presented in this paper for GMST. In contrast to existing heuristics, DASA employs those candidate sets to initialize and optimize solutions. During the search process, those candidate sets are dynamically adjusted to include in new features provided by good solutions. Since those candidate sets cover almost all optimal solutions, the search space of DASA can be dramatically reduced so that elite solutions can be easily found in a short time. Extensive experiments demonstrate that our new algorithm slightly outperforms existing heuristic algorithms in terms of solution quality.

## Categories and Subject Descriptors

I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search-*Heuristic methods*.

## General Terms

Algorithms

## Keywords

Local Search, Generalized Minimum Spanning Tree, Candidate Set

## 1. INTRODUCTION

The Generalized Minimum Spanning Tree problem (GMST) was firstly introduced by Myung et al. [1]. As an extension of the classical Minimum Spanning Tree problem (MST), GMST was shown to be a NP-hard problem [1] arising in telecommunication, design of backbones in large communication networks, energy distribution, and agricultural irrigation.

According to the computational complexity theory, there is no algorithm to solve NP-hard problems to optimality in polynomial time unless P=NP. As a result, exact algorithms are only applicable to small instances. Therefore, many heuristic algorithms have been proposed for GMST to achieve near optimal solutions in reasonable time. Feremans [2] proposed a Tabu Search algorithm for GMST in 2001. Pop [3] presented a Simulated Annealing heuristic (SA) based on the global edge-change neighborhood. Ghosh [4] developed a series of heuristic algorithms for GMST, including Tabu Search based on recency memory (TS1), Tabu Search based on recency and frequency memory (TS2), Variable Neighborhood Descent Search, Reduced VNS, and VNS with Steepest and a Variable Neighborhood Decomposition Search (VNDS). In those four variants of the VNS, 1-swap and 2-swap were used as neighborhoods. Golden et al. [5] presented a Local-Search Heuristic (LSH) based on the 1-swap neighborhood. Furthermore, they also proposed a Genetic Algorithm (GA) which took LSH as the mutation operator. Another Tabu Search was developed recently by Wang et al. [6]. Temel et al. [7] presented an attribute based Tabu Search employing new neighborhood called $N^c(s)$. Hu et al. [8] devised a VNS with the global edges exchange neighborhood designed by Pop [3] and the restricted nodes exchange neighborhood provided by Ghosh [4]. Moreover, by employing an additional neighborhood type based on the small enough parts of a candidate solution via Mixed Integer Programming (MIP), Hu et al. proposed a new VNS for the GMST [9].

As an efficient tool for heuristic algorithm design, the muscle was firstly introduced by Jiang et al. [10] for the Three-Index Assignment Problem (AP3). The muscle is defined as the union of all optimal solutions. In [10], an Approximate Muscle guided Global Optimization (AMGO) was proposed to solve the AP3. Experimental results demonstrated that the new conception could dramatically improve the effectiveness for heuristic algorithm.

Motivated by the success of the muscle in AP3, we investigate how to employ this tool for GMST as follows. Firstly, we demonstrate that it's NP-hard to obtain the muscle for GMST. The key idea behind the proof is to map any GMST instance to a biased GMST instance with a unique optimal solution. Therefore, finding the muscle of the biased instance is equivalent to finding an optimal solution to the original instance. Secondly, it's shown that the muscle can be approximated by those principle and subordinate candidate sets, which are defined by the lower bounds on a reduced version of GMST. Finally, a new algorithm named DASA is proposed to efficiently solve GMST. DASA consists of two phases. In the first phase, those candidate sets are generated. The second phase is a loop composed of several steps. In each loop, an initial solution is constructed for future improvement. After that, a local search based on those candidate sets is employed to improve the initial solution. Then, a path relinking procedure is conducted to further improve the current solution. During the whole loop, those candidate sets will be dynamically adjusted to retain new features provided by new solutions. Experiments on 46 widely used instances [9] demonstrate that our new algorithm outperforms existing heuristic algorithms in terms of solution quality.

## 2. PRELIMINARIES

In this section, we shall present some related definitions and notations about GMST.

Given an undirected weighted graph $G=(V,E,w)$, where $V=\{1,2,\cdots,n\}$ is the node set, $E$ is the edge set, and $w:E\to R^+$ is the edge cost function. The node set $V$ is partitioned into $k$ disjoint nonempty clusters $C_1,C_2,\cdots,C_k$ such that $C_1\bigcup C_2\bigcup\cdots\bigcup C_k=V$ and $C_i\bigcap C_j=\varnothing$ ($1\le i\ne j\le k$). A feasible solution to the GMST instance (denoted by $GMST(V,E,w)$) is defined as a MST on the sub-graph $G_s=(V_s,E_s,w_s)$, where $V_s=\{v_i|1\le i\le k\}$ contains exactly one node from every cluster, i.e., $v_i\in C_i$ ($1\le i\le k$), $E_s=\{(v_i,v_l)|(v_i,v_l)\in E,v_i,v_l\in V_s\}$, and $w_s(v_i,v_l)=w(v_i,v_l)$. GMST aims to find a feasible solution whose cost is minimized. Since a MST can be constructed from the sub-graph $G_s=(V_s,E_s,w_s)$ in $O(|E_s|\log|V_s|)$ time [11], most papers use the set $V_s$ to represent the feasible solution rather than the MST. The cost for the MST derived from $G_s=(V_s,E_s,w_s)$ is denoted by $sol(V_s)$.

Figure 1 illustrates an example for a GMST instance. There're 5 clusters in this instance, including $\{1,2,3,4\}$, $\{5,6,7,8,9\}$, $\{10,11\}$, $\{12,13,14\}$, and $\{15,16\}$. For brevity, the edges among these clusters are not given in Figure 1. A MST generated from the solution {4, 6, 10, 12, 16} is linked by the black lines.

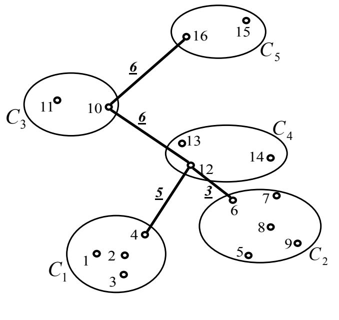


**Figure 1. An Example of GMST**

Given a GMST instance $GMST(V,E,w)$, let $\Pi^*=\{V_s^1,V_s^2,\cdots,V_s^q\}$ be the set of all optimal solutions, where $|\Pi^*|=q$ represents the number of optimal solutions. The muscle of $GMST(V,E,w)$ is defined as $muscle(V,E,w)=V_s^1\bigcup V_s^2\bigcup\cdots\bigcup V_s^q$. It's easy to see that we can dramatically reduce the search space if the muscle is available.

Given a GMST instance $GMST(V,E,w)$, its biased instance is defined as $GMST(V,E,\hat{w})$, where $\hat{w}(i,j)=w(i,j)+1/2^{in+j}$ for every $(i,j)\in E$ ($i<j$) and $\hat{w}(i,j)=\hat{w}(j,i)$ ($i>j$). Obviously, the biased instance is also a GMST instance and a feasible solution to the biased instance is also feasible to its original instance. Given a solution $V_s$ to $GMST(V,E,\hat{w})$, its cost is denoted by $s\hat{o}l(V_s)$.

We shall shows that it's NP-hard to obtain the muscle for GMST as follows.

**Lemma 1.** Given a GMST instance $GMST(V,E,w)$, if $w(i,j)$ is nonnegative integer for every $(i,j)\in E$, then the biased instance $GMST(V,E,\hat{w})$ has a unique optimal solution.

**Proof.** To prove this lemma, we only need to verify that $s\hat{o}l(V_s')\ne s\hat{o}l(V_s'')$, for any two distinct feasible solutions $V_s'\ne V_s''$ to the biased instance $GMST(V,E,\hat{w})$.

By definition, the cost function of any MST is the total edge cost to the spanning tree. Since $V_s'\ne V_s''$, there must exist one edge $(i^*,j^*)$ ($i^*<j^*$) which is only contained in $MST(V_s',\hat{w})$ rather than $MST(V_s'',\hat{w})$, where $MST(V_s',\hat{w})$ and $MST(V_s'',\hat{w})$ are the MSTs derived from $G_s'=(V_s',E_s',\hat{w})$ and $G_s''=(V_s'',E_s'',\hat{w})$, respectively. When viewed as a binary string, the $i^**n+j^*$ th bit of the fractional part of $s\hat{o}l(V_s')$ will be 1. However, the same bit of $s\hat{o}l(V_s'')$ will be 0. Therefore, we have that $s\hat{o}l(V_s')\ne s\hat{o}l(V_s'')$ holds.

Thus, this lemma is proved. □

**Lemma 2.** Given a GMST instance $GMST(V,E,w)$, if $w(i,j)$ is nonnegative integer for every $(i,j)\in E$, then the unique optimal solution to the biased instance $GMST(V,E,\hat{w})$ is also optimal to $GMST(V,E,w)$.

**Proof.** By Lemma 1, given a GMST instance $GMST(V,E,w)$**,** there exists a unique optimal solution (denoted by $V_s^*$) to the biased instance $GMST(V,E,\hat{w})$. Obviously, $V_s^*$ is also a feasible solution to $GMST(V,E,w)$.

Assuming Lemma 2 is false, there exists at least one solution $V_s'$ such that $sol(V_s')<sol(V_s^*)$. We verify that a contradiction will be found in the following proof.

According to the assumption that $w(i,j)\in Z^+\bigcup\{0\}$ for $(i,j)\in E$, we have that $sol(V_s')\in Z$ and $sol(V_s^*)\in Z$. Since $sol(V_s')<sol(V_s^*)$, we have that $sol(V_s^*)-sol(V_s')\ge 1$. On the other hand, we have $0<s\hat{o}l(V_s')-sol(V_s')<\sum_{(i,j)\in E}1/2^{i*n+j}<1$. Similarly, we have $0<s\hat{o}l(V_s^*)-sol(V_s^*)<1$. Thus, it implies that

$$\begin{aligned}&s\hat{o}l(V_s^*)\ -s\hat{o}l(V_s')\\&=sol(V_s^*)-sol(V_s')+(s\hat{o}l(V_s^*)-sol(V_s^*))\ -(s\hat{o}l(V_s')-sol(V_s'))\\&\ge 1+(s\hat{o}l(V_s^*)-sol(V_s^*))\ -(s\hat{o}l(V_s')-sol(V_s'))\ >1-1=0\end{aligned}$$

However, it contradicts with the fact that $V_s^*$ is optimal to the biased instance $GMST(V,E,\hat{w})$. Thus, this lemma is proved. □

**Theorem 1.** There exists no polynomial time algorithm to obtain the muscle of GMST unless $P = NP$.

**Proof.** Otherwise, there must be a polynomial time algorithm (denoted by H) which can obtain the muscle of GMST. A contradiction will be found in the following proof by constructing a polynomial time algorithm to solve GMST.

Given any GMST instance $GMST(V,E,w)$, without loss of generality, we shall assume that $w(i,j)$ is nonnegative integer for every $(i,j) \in E$, otherwise we can rescale every edge weight by multiplying a large number. Therefore, an optimal solution to the instance $GMST(V,E,w)$ can always be found as follows.

Firstly, the biased instance $GMST(V,E,\hat{w})$ can be constructed in $O(n^2)$ time. Secondly, since the biased instance is also a GMST instance, its muscle can be found by H in polynomial time (denoted by $O(\bullet)$). By Lemma 1, the muscle is the unique optimal solution to $GMST(V,E,\hat{w})$. Meanwhile, by Lemma 2, the muscle is also optimal to $GMST(V,E,w)$. Therefore, we can always solve GMST in $O(\bullet)+O(n^2)$ running time. Obviously, it contradicts with the fact that GMST is NP-hard. Thus, this theorem is proved. □

## 3. CANDIDATE SETS

In this section, we shall discuss how to approximate the muscle.

Given a node $v \in C_i$ ($1 \le i \le k$), a $v$-graph $G_v = (V_v, E_v, w_v)$ is defined as follows. Every cluster $C_j$ ($1 \le j \le k$) is reduced to a node $c_j$. Let $V_v = \{c_j \mid 1 \le j \le k\}$, let $E_v = \{(c_j, c_l) \mid 1 \le j, l \le k, j \ne l\}$. For every edge $(c_j, c_l)$ ($j \ne l, j \ne i, l \ne i$), the edge cost $w_v(c_j, c_l)$ is defined as the minimum cost of edges between cluster $C_j$ and $C_l$, i.e., $w_v(c_j, c_l) = \min w(v_p, v_q)$, where $v_p \in C_j$ and $v_q \in C_l$. For every edge $(c_i, c_j)$ ($j \ne i$), its cost $w_v(c_i, c_j)$ is defined as the minimum cost of edges between node $v$ and the nodes in $C_j$ on $G = (V,E,w)$. After the $v$-graph is constructed, the lower bound (denoted by $LB(v)$) for $v$ is defined as the cost of the MST on $G_v = (V_v, E_v, w_v)$. The MST can be obtained in $O(|E_v| \log |V_v|)$ time [11].

Given a cluster $C_i$, we sort the nodes in cluster $C_i$ by their lower bounds in ascending order. The principle candidate set (denoted by $PCS(C_i)$) for $C_i$ is defined as the set of those nodes ranked from 1st to $\lceil |C_i| * r \rceil$th, where $r$ ($0 \le r \le 1$) is the principle candidate set ratio. The union of all the principle candidate sets is denoted by $PCS(V)$, i.e., $PCS(V) = \bigcup_{1 \le i \le k} PCS(C_i)$.

To evaluate the effectiveness of the principle candidate sets, we conducted several experiments on some typical GMST instances whose optimal solutions are presented in [2]. As shown in Figure 2, along with the growth of the value of $r$, more and more nodes in optimal solutions will be contained by $PCS(V)$. When the principle candidate ratio $r$ exceeds 0.4, there're over 80% nodes in the optimal solutions will be contained by $PCS(V)$ for all the instances (gr137, kroa150, gr202, and krob200).

In addition to the principle candidate set, we also define the subordinate candidate set for every node $v \in C_i$ ($1 \le i \le k$). Given a node $v \in C_i$ ($1 \le i \le k$) and its $v$-graph $G_v = (V_v, E_v, w_v)$, let $A(v)$ be the set of nodes adjacent to $c_i$ in the MST on $G_v = (V_v, E_v, w_v)$. The subordinate candidate set (denoted by $SCS(v)$) for node $v$ is defined as the set of nodes $v' \in V$ such that $v' \in C_l$, $c_l \in A(v)$, and $w(v, v') = w_v(c_i, c_l)$. The union of all the subordinate candidate sets is denoted by $SCS(V)$, i.e., $SCS(V) = \bigcup_{v \in V} SCS(v)$. An example is given in Figure 3. Figure 3 (a) shows the 10-graph defined on GMST and its MST plotted by dashed lines. Figure 3 (b) shows that $SCS(10) = \{16, 13\}$, since $w(10,13) = w_{10}(c_3, c_4)$, $w(10,16) = w_{10}(c_3, c_5)$.

Similar to the principle candidate sets, we also evaluated the effectiveness of the subordinate candidate sets on some typical GMST instances. As shown in Figure 4 (a), along with the growth of the value of $r$, most nodes in optimal solutions will be contained by $SCS(V)$. Figure 4 (b) shows that the normalized size of $SCS(V)$ increases slowly along with the growth of the value of $r$.

In summary, it's a good way to approximate the muscle with both principle candidate sets and subordinate candidate sets.

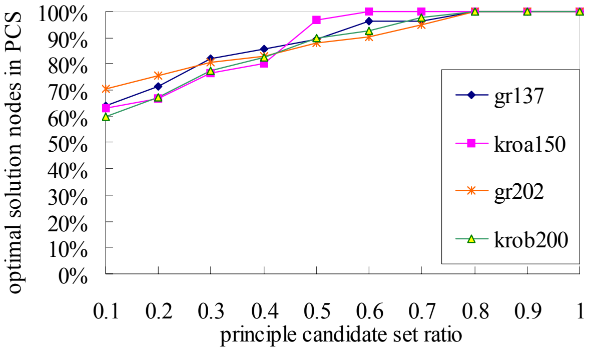
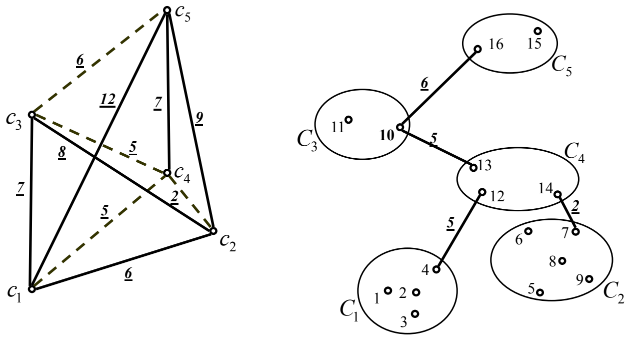


**Figure 2. Optimal Solution Nodes in PCS vs. Principle Candidate Set Ratio**



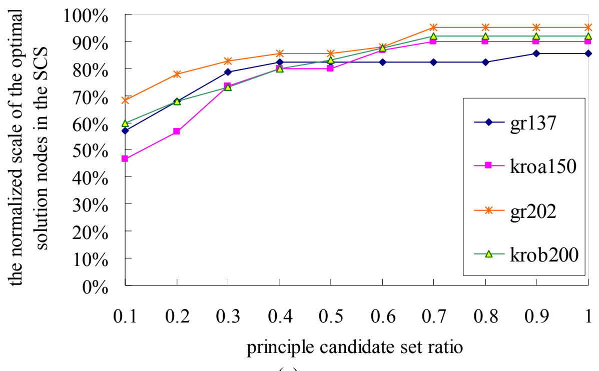
(a) 10-graph $G_{10}$ (b) Subordinate Candidate of Node 10

**Figure 3. Illustration of the Subordinate Candidate Set**

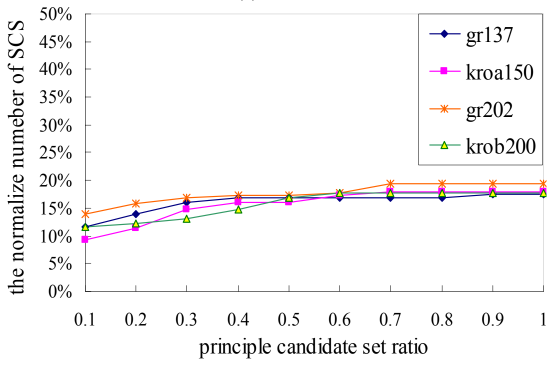
## 4. DASA

Inspired by the observation that the muscle can be approximated by those candidate sets (both principle and subordinate candidate sets), we propose the Dynamic cAndidate set based Search Algorithm (DASA) for GMST in this section. It can dramatically reduce the search space by restricting the search process in the candidate sets.

The framework of DASA is presented in Algorithm 1. After the initiation of the best solution, the principle and subordinate candidate sets will be generated. Then, a loop is repeated in DASA until the stopping criterion is met. In this paper, a pre-defined running time is given as the stopping criterion. The loop mainly



(a)



(b)

**Figure 4. Optimal Solution Nodes in SCS vs. Principle Candidate Set Ratio**

```
Algorithm 1: DASA
Input: GMST instance $GMST(V,E,w)$, $r$, $\Delta j$, $\Delta r$
Output: solution $V_s^*$
Begin
(1) let the best solution iteration number $j^*$=0, the best solution $V_s^*$= $\varnothing$ with cost $sol(V_s^*)=+\infty$, $j=0$
(2) generate principle candidate sets with the ratio $r$ and subordinate candidate sets
(3) while the stopping criterion is not met do
        //to generate an initial solution
    (3.1) if $(j-j^*)\bmod \Delta j=0$ //no improvement in $\Delta j$ iterations then
        (3.1.1) $V_s^0$=InitSolution($GMST(V,E,w)$)
        (3.1.2) $r=r+\Delta r$ //to enlarge the principle candidate sets
          else
        (3.1.3) shake $V_s^*$ by random perturbation to generate $V_s^0$
    (3.2) $V_s$=LS($V_s^0$) //local search
    (3.3) $V_s$=PathRelinking($V_s$,$V_s^*$)
    (3.4) if $sol(V_s)<sol(V_s^*)$
        (3.4.1) adjust the principle candidate set
        (3.4.2) $V_s^*=V_s$, $j^*=j$
    (3.5) $j++$
(4) return $V_s^*$
End
```

consists of 4 steps. Firstly, an initial solution is constructed for further improvement (see Step (3.1)). At most times, the initial solution is generated by randomly perturbing the best solution, so that the initial solution can retain some components in the best solution while some new features can also be introduced for future improvement. To avoid getting trapped in the neighborhood of the best solution, a new solution will be constructed from those candidate sets directly, when no improvement is achieved during predefined iterations ($\Delta j$ iterations in DASA). Secondly, a local search process is employed to improve the initial solution (see Step (3.2)). Thirdly, a path relinking process is called to further improve the current solution (see Step (3.3)). Finally, DASA checks whether a better solution is obtained. If so, the principle candidate sets will be adjusted to include new features provided by this new solution, and the best solution is replaced with the new solution (see Step (3.4)).

## 4.1 Initializing a Solution with Candidate Sets

Obviously, a good initial solution is essential for local search. In this subsection, we present the algorithm of InitSolution which is used in Step (3.1) of DASA. The key idea of InitSolution is to generate a new solution by employing the principle candidate sets and the subordinate candidate sets. As shown in Section 2, these two kinds of candidate sets contain most nodes appearing in optimal solutions. Therefore, the initial solutions generated by InitSolution are more likely to converge to optimal solutions than random ones.

```
Algorithm 2: InitSolution
Input: GMST instance $GMST(V,E,w)$
Output: initial solution $V_s^0$
Begin
(1) set $C=\{C_1,C_2,\cdots C_k\}$, $V_s^0=\varnothing$, $Q=\varnothing$
(2) while $|V_s^0|<k$ do
        if $Q=\varnothing$ then
            (2.1) randomly choose a cluster $C_\xi\in C$
            (2.2) select the 1st ranked node $v$ from $PCS(C_\xi)$
            (2.3) $Q=\{v\}$, $C=C\setminus C_\xi$, $V_s^0=V_s^0\cup\{v\}$
        else // $Q\neq\varnothing$
            (2.4) select a node $v\in Q$
            (2.5) $Q=Q\setminus\{v\}$
            (2.6) for every $v'\in SCS(v)$ do
                (2.6.1) if $\exists C_\eta\in C$ s.t. $v'\in PCS(C_\eta)$ then
                        $Q=Q\cup\{v'\}$, $C=C\setminus C_\eta$, $V_s^0=V_s^0\cup\{v'\}$
(3) return $V_s^0$
End
```

A detailed description of InitSolution is given in Algorithm 2. Two sets are used in InitSolution to record different kinds of information. The set $Q$ is employed to record those nodes under consideration and the set $C$ is to record those unchecked clusters. The algorithm InitSolution works as follows. Firstly, all the clusters are marked as unchecked by setting $C=\{C_1,C_2,\cdots C_k\}$ and the initial solution is also set to empty (see Step (1)). At this step, there's no node under consideration, i.e., $Q = \varnothing$ . Secondly, a loop is repeated until the initial solution is successfully constructed (see Step (2)). There're 2 cases in the loop. For the case that no node exists to be considered in $Q$ (see Step (2.1)-(2.3)), an unchecked cluster will be randomly chosen and removed from $C$ . The first ranked node in the principle candidate set of this chosen cluster will be added to $Q$ for further consideration. At the mean time, this first ranked node will be also added to the initial solution. For the other case that $Q \neq \varnothing$ (see Step (2.4)-(2.6)), a node $v$ in $Q$ will be picked out. After that, every node $v'$ in its subordinate candidate set will be checked in the following way. If the cluster containing $v'$ hasn't been checked and $v'$ belongs to the principle candidate set of this cluster, the node $v'$ will be added to $Q$ for further consideration. In addition, the node $v'$ will also be inserted to the initial solution.

## 4.2 Shaking by Random Perturbation

In DASA, a new initial solution can also be generated by shaking the best solution. As shown by Boose [12], there exist "big valley" structures for many combinatorial optimization problems (e.g. the traveling salesman problem), i.e., a lot of local optimal solutions cluster together around optimal solutions. Therefore, it's a good way to obtain the initial solution by shaking the best solution.

In this paper, we randomly perturb the best solution for further local search as follows. Firstly, $t = \lceil \phi * \Delta iteration \rceil$ clusters are randomly selected, where $\Delta iteration$ is the iterations elapsed since the best solution is found in DASA, $\phi$ is the perturbation intensity factor between 0 and 1. Secondly, a node in every selected cluster is arbitrarily chosen to replace the existing one in the best solution. By this strategy, some useful features of the best solution can be retained in the new initial solution, while many new nodes are introduced for further improvement. Obviously, the more iteration elapsed since the best solution is achieved, the more randomness will be introduced by our shaking strategy.

## 4.3 Candidate Sets Adjustment

In DASA, the principle candidate sets are dynamically adjusted in two cases.

For the case that no improvement is achieved in $\Delta j$ iteration, we enlarge the principle candidate set size by setting $r = r + \Delta r$ (see Step (3.1.2) in DASA). This strategy is necessary, because an optimal solution may include a "bad" node when viewed from a local perspective. Although small principle candidate sets can contain most nodes in optimal solutions, a fraction of those nodes in optimal solution can still be excluded out of the principle candidate sets. Therefore, we dynamically enlarge the principle candidate sets to cover more potential nodes.

For the case that a better solution is achieved after local search and path relinking process, the principle candidate sets are adjusted to retain elite information from the improved solutions (see Step (3.4.1) in DASA). It works as follows. Firstly, every node appearing in both the new better solution and principle candidate sets is moved to the first ranked positions. If it isn't contained in its corresponding principle candidate set, a node in the new better solution will be inserted to the first ranked position in the principle candidate set.

## 4.4 Local Search

After an initial solution is generated, a local search process will be employed in DASA to further improve it. Algorithm 3 presents the framework of our local search process. Firstly, a random visit order $R_V$ for clusters will be initialized (see Step (1)). Then the current solution will be improved by a process named ClusterOptimizer, cluster by cluster in $R_V$ (see Step (3.2)). When a better solution is found, the current solution will be replaced with the new found solution. And then the new current solution will be further improved using clusters in $R_V$ until no improvement can be achieved.

```
**Algorithm 3:** LS
**Input:** solution $V_s^0$
**Output:** solution $V_s^*$
**Begin**
(1) initialize a random visit order $R_V$ for clusters, let $V_s^* = V_s^0$
(2) flag = true
(3) while flag = true do
    (3.1) flag = false
    (3.2) for i = 1 to $k$ do
        (3.2.1) let $C_\gamma$ be the i^th cluster in $R_V$
        (3.2.2) $V_s = ClusterOptimizer(V_s^*, C_\gamma)$
        (3.2.3) if $sol(V_s) < sol(V_s^*)$ then
                $V_s^* = V_s$ , flag = true, break
(4) return $V_s^*$
**End**
```

Given a cluster and an initial solution, let $v^*$ be the cluster's node appearing in the solution. The process ClusterOptimizer (see Algorithm 4) works as follows. For every node $v$ belonging to the principle candidate set of this cluster, a modified solution is generated by replacing $v^*$ with $v$ . If any improvement can be achieved, the modified solution is returned. Otherwise, a depth first search named DFS is further employed to improve the modified solution. If the resulting solution from DFS is better than the initial solution, this solution from DFS will be returned.

```
**Algorithm 4:** ClusterOptimizer
**Input:** solution $V_s^0$ , cluster $C_i$
**Output:** solution $V_s^*$
**Begin**
(1) let $v^*$ be the cluster $C_i$ 's node appearing in $V_s^0$ , $V_s^* = V_s^0$
(2) for every node $v \in PCS(C_i) \setminus \{v^*\}$ do
    (2.1) let $V_s = V_s^0 \setminus \{v^*\} \cup \{v\}$
    (2.2) if $sol(V_s) < sol(V_s^0)$ then $V_s^* = V_s$ , break
        else
            $V_s'$=DFS($V_s, V_s^0, v, \{C_i\}$) //depth first search from $v$
            if $sol(V_s') < sol(V_s^0)$ then $V_s^* = V_s'$ , break
(3) return $V_s^*$
**End**
```

DFS (see Algorithm 5) works in a recursive way. A set $P$ is used to record those visited clusters. Given a node $v$, the current solution $V_s$ and the initial solution $V_s^0$, DFS iteratively checks every node $v'$ belonging to the subordinate candidate set of node $v$. If the cluster (denoted by $C_{v'}$ in DFS) containing $v'$ hasn't been visited yet, a modified solution $V_s'$ will be generated by replacing $C_{v'}$'s node in $V_s$ with $v'$. If it is better than the initial solution $V_s^0$, the modified solution will be returned. Otherwise, the DFS process will be recursively called by taking in $V_s'$, $V_s^0$, $v'$,and $P \bigcup \{C_{v'}\}$ as input. More details can be found in Algorithm 5.

```
Algorithm 5: DFS
Input: solution $V_s$, solution $V_s^0$, node $v$, visited cluster set $P$
Output: solution $V_s^*$
Begin
(1) $V_s^* = V_s$
(2) for every node $v' \in SCS(v)$ do
    (2.1) let $C_{v'}$ be the cluster containing $v'$, let $v''$ be the $C_{v'}$'s
          node appearing in $V_s$
    (2.2) if $C_{v'} \notin P$ then //when $C_{v'}$ hasn't been visited before
        (2.2.1) let $V_s' = V_s \setminus \{v''\} \bigcup \{v'\}$
        (2.2.2) if $sol(V_s') < sol(V_s^0)$ then $V_s^* = V_s'$, break
        (2.2.3) $V_s'' = DFS(V_s', V_s^0, v', P \bigcup \{C_{v'}\})$
        (2.2.4) if $sol(V_s'') < sol(V_s^0)$ then $V_s^* = V_s''$, break
        (2.2.5) if $sol(V_s'') < sol(V_s^*)$ then $V_s^* = V_s''$
(3) return $V_s^*$
End
```

## 4.5 PathRelinking

In DASA, a PathRelinking strategy is called to improve the local optimal solution. PathRelinking is firstly proposed by Glover [13] as an intensification strategy to explore trajectories linking two elite solutions. The key idea behind PathRelinking is to combine basic components of two guiding solutions so that the search space between them can be explored to discover new better solutions.

Our PathRelinking for DASA is presented in Algorithm 6, which takes in two solutions (denoted by $V_s$, $V_s'$, respectively) as input. The main part of PathRelinking consists of two phases. The first phase (Step (1)-(2)) constructs a solution from $V_s$ to $V_s'$, and the second one (Step (3)-(4)) does from $V_s'$ to $V_s$. Since these two phases work in a similar way, we just briefly investigate the first phase as follows. Firstly, the nodes appearing in $V_s'$ rather than $V_s$ are collected in the set $V_d$. Both the current solution $V_s^0$ and the resulting solution $V_s^1$ in the first phase are initialized by $V_s$. Secondly, a loop is repeated until $V_s^0$ equals to $V_s'$. In the loop (Step (2)), every node in $V_d$ will replace its corresponding node appearing in $V_s^0$ to construct a new solution (denoted by $V_s''$), respectively. Out of those new generated solutions, the best one will become the new current solution for further iteration (Step (2.3)). If the new current solution $V_s^0$ is better than $V_s^1$, it will become the new resulting solution (Step (2.4)). After such processes, the set $V_d$ is renewed and this loop continues until $V_d$ is empty (i.e., $V_s^0$ equals to $V_s'$). After these two phases, two solutions are generated and the better one will be returned.

```
Algorithm 6: PathRelinking
Input: solution $V_s$, $V_s'$
Output: solution $V_s^*$
Begin
// relink from $V_s$ to $V_s'$
(1) let $V_d = V_s' \setminus V_s$, $V_s^1 = V_s$, $V_s^0 = V_s$
(2) while $V_s^0 \neq V_s'$
    (2.1) $mc = +\infty$
    (2.2) for every $v \in V_d$ do
        (2.2.1) let $C_v$ be the cluster containing $v$
        (2.2.2) replace the $C_v$'s node in $V_s^0$ with $v$ to generate a
                new solution $V_s''$
        (2.2.3) if $sol(V_s'') < mc$ then
                    $\bar{V}_s = V_s''$, $mc = sol(V_s'')$, $\bar{v} = v$
    (2.3) $V_s^0 = \bar{V}_s$
    (2.4) if $sol(V_s^0) < sol(V_s^1)$ then $V_s^1 = V_s^0$
    (2.5) $V_d = V_d \setminus \{\bar{v}\}$
// relink from $V_s'$ to $V_s$
(3) let $V_d = V_s \setminus V_s'$, $V_s^2 = V_s'$, $V_s^0 = V_s'$
(4) while $V_s^0 \neq V_s$
    (4.1) $mc = +\infty$
    (4.2) for every $v \in V_d$ do
        (4.2.1) let $C_v$ be the cluster containing $v$
        (4.2.2) replace the $C_v$'s node in $V_s^0$ with $v$ to generate a
                new solution $V_s''$
        (4.2.3) if $sol(V_s'') < mc$ then
                    $\bar{V}_s = V_s''$, $mc = sol(V_s'')$, $\bar{v} = v$
    (4.3) $V_s^0 = \bar{V}_s$
    (4.4) if $sol(V_s^0) < sol(V_s^2)$ then $V_s^2 = V_s^0$
    (4.5) $V_d = V_d \setminus \{\bar{v}\}$
(5) if $sol(V_s^1) < sol(V_s^2)$ then $V_s^* = V_s^1$ else $V_s^* = V_s^2$
(6) return $V_s^*$
End
```

## 5. EXPERIMENTS AND ANALYSIS

In this section, we demonstrate the effectiveness of DASA by experimental results over the GMST instances used by Hu et al. [9]. There're 4 kinds of instances, including TSPLIB, grouped Euclidean, random Euclidean, and non-Euclidean. DASA is implemented in C++ on a Pentium D 2.66GHz PC with 1GB RAM running the Federal Linux 10 operating system. In the experiment, DASA sets $r$=0.5, $\Delta j$ =10, $\Delta r$ =0.2, $\phi$ =0.8. For comparison, we also list the results for TS2 [4], VNDS [4], SA [3], GA [5], and VNS [9]. All the experimental results for those algorithms are obtained from [9].

As presented in [9], the results for VNDS and VNS are averaged over 30 runs, and the results for SA are averaged over 10 runs due to its long running time. Since TS2 is deterministic, it's run only once in [9]. For every run of TS2, VNDS, and VNS,

**Table 1. Results on TSPLIB Instances with Geographical Clustering**

| TSP instances | | | | TS2 | VNDS | SA | | GA | VNS | | DASA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Names | \|V\| | $k$ | time | $C(T)$ | $\overline{C(T)}$ | $\overline{C(T)}$ | std dev | $\overline{C(T)}$ | $\overline{C(T)}$ | std dev | $\overline{C(T)}$ | std dev |
| gr137 | 137 | 28 | 150s | **329.0** | 330.0 | 352 | 0.00 | **329.0** | **329.0** | 0.00 | **329.0** | 0.00 |
| kroa150 | 150 | 30 | 150s | **9815.0** | **9815.0** | 10885.6 | 25.63 | **9815.0** | **9815.0** | 0.00 | **9815.0** | 0.00 |
| d198 | 198 | 40 | 300s | 7062.0 | 7169.0 | 7468.7 | 0.83 | **7044.0** | **7044.0** | 0.00 | **7044.0** | 0.00 |
| krob200 | 200 | 40 | 300s | 11245.0 | 11353.0 | 12532.0 | 0.00 | **11244.0** | **11244.0** | 0.00 | **11244.0** | 0.00 |
| gr202 | 202 | 41 | 300s | **242.0** | 249.0 | 258.0 | 0.00 | 243.0 | **242.0** | 0.00 | **242.0** | 0.00 |
| ts225 | 225 | 45 | 300s | 62366.0 | 63139.0 | 67195.1 | 34.49 | 62315.0 | 62268.5 | 0.51 | **62268.3** | 0.48 |
| pr226 | 226 | 46 | 300s | **55515.0** | **55515.0** | 56286.6 | 40.89 | **55515.0** | **55515.0** | 0.00 | **55515.0** | 0.00 |
| gil262 | 262 | 53 | 300s | **942.0** | 979.0 | 1022.0 | 0.00 | – | 942.3 | 1.02 | **942.0** | 0.00 |
| pr264 | 264 | 54 | 300s | **21886.0** | 22115.0 | 23445.8 | 68.27 | – | 21886.5 | 1.78 | **21886.0** | 0.00 |
| pr299 | 299 | 60 | 450s | 20339.0 | 20578.0 | 22989.4 | 11.58 | – | 20322.6 | 14.67 | **20317.4** | 1.52 |
| lin318 | 318 | 64 | 450s | 18521.0 | 18533.0 | 20268.0 | 0.00 | – | **18506.8** | 11.58 | 18513.6 | 7.82 |
| rd400 | 400 | 80 | 600s | 5943.0 | 6056.0 | 6440.8 | 3.40 | – | 5943.6 | 9.69 | **5941.5** | 9.91 |
| fl417 | 417 | 84 | 600s | 7990.0 | 7984.0 | 8076.0 | 0.00 | – | **7982.0** | 0.00 | 7982.7 | 0.47 |
| gr431 | 431 | 87 | 600s | 1034.0 | 1036.0 | 1080.5 | 0.51 | – | **1033.0** | 0.18 | **1033.0** | 0.00 |
| pr439 | 439 | 88 | 600s | 51852.0 | 52104.0 | 55694.1 | 45.88 | – | 51847.9 | 40.92 | **51833.8** | 36.07 |
| pcb442 | 442 | 89 | 600s | **19621.0** | 19961.0 | 21515.1 | 5.15 | – | 19702.8 | 52.11 | 19662.5 | 39.79 |

**Table 2. Results on Grouped Euclidean, Random Euclidean, and Non-Euclidean Instances**

| Instances | | | | | TS2 | VNDS | SA | | VNS | | DASA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Set | \|V\| | $k$ | \|V\|/$k$ | time | $C(T)$ | $\overline{C(T)}$ | $\overline{C(T)}$ | std dev | $\overline{C(T)}$ | std dev | $\overline{C(T)}$ | std dev |
| | 125 | 25 | 5 | 600s | **141.1** | **141.1** | 152.3 | 0.52 | **141.1** | 0.00 | **141.1** | 0.00 |
| Grouped Eucl 125 | 125 | 25 | 5 | 600s | **133.8** | **133.8** | 150.9 | 0.74 | **133.8** | 0.00 | **133.8** | 0.00 |
| | 125 | 25 | 5 | 600s | 143.9 | 145.4 | 156.8 | 0.00 | **141.4** | 0.00 | **141.4** | 0.00 |
| | 500 | 100 | 5 | 600s | **566.7** | 577.6 | 642.3 | 0.00 | 567.4 | 0.57 | 588.1 | 2.09 |
| Grouped Eucl 500 | 500 | 100 | 5 | 600s | 578.7 | 584.3 | 663.3 | 1.39 | 585.0 | 1.32 | **573.7** | 1.17 |
| | 500 | 100 | 5 | 600s | 581.6 | 588.3 | 666.7 | 1.81 | 583.7 | 1.82 | **581.1** | 1.94 |
| | 600 | 20 | 30 | 600s | 85.2 | 87.5 | 93.9 | 0.00 | **84.6** | 0.11 | **84.6** | 0.00 |
| Grouped Eucl 600 | 600 | 20 | 30 | 600s | **87.9** | 90.3 | 99.5 | 0.28 | **87.9** | 0.00 | **87.9** | 0.00 |
| | 600 | 20 | 30 | 600s | 88.6 | 89.4 | 99.2 | 0.17 | **88.5** | 0.00 | **88.5** | 0.00 |
| | 1280 | 64 | 20 | 600s | 327.2 | 329.2 | 365.1 | 0.46 | **315.9** | 1.91 | 320.1 | 3.52 |
| Grouped Eucl 1280 | 1280 | 64 | 20 | 600s | 322.2 | 322.5 | 364.4 | 0.00 | 318.3 | 1.78 | **317.2** | 2.37 |
| | 1280 | 64 | 20 | 600s | 332.1 | 335.5 | 372.0 | 0.00 | 329.4 | 1.29 | **329.1** | 1.99 |
| | 250 | 50 | 5 | 600s | **2285.1** | 2504.9 | 2584.3 | 23.82 | 2300.9 | 40.27 | 2292.5 | 23.53 |
| Random Eucl 250 | 250 | 50 | 5 | 600s | **2183.4** | 2343.3 | 2486.7 | 0.00 | 2201.8 | 23.30 | 2244.0 | 56.49 |
| | 250 | 50 | 5 | 600s | **2048.4** | 2263.7 | 2305.0 | 16.64 | 2057.6 | 31.58 | 2071.2 | 55.90 |
| | 400 | 20 | 20 | 600s | **557.4** | 725.9 | 665.1 | 3.94 | 615.3 | 10.80 | 612.4 | 7.23 |
| Random Eucl 400 | 400 | 20 | 20 | 600s | 724.3 | 839.0 | 662.1 | 7.85 | **595.3** | 0.00 | 611.5 | 23.35 |
| | 400 | 20 | 20 | 600s | 604.5 | 762.4 | 643.7 | 14.54 | **587.3** | 0.00 | 604.9 | 23.45 |
| | 600 | 20 | 30 | 600s | 541.6 | 656.1 | 491.8 | 7.83 | **443.5** | 0.00 | 508.3 | 59.08 |
| Random Eucl 600 | 600 | 20 | 30 | 600s | 540.3 | 634.0 | 542.8 | 25.75 | **537.0** | 10.20 | 550.7 | 32.03 |
| | 600 | 20 | 30 | 600s | 627.4 | 636.5 | 469.5 | 2.75 | **469.0** | 11.90 | 530.6 | 26.81 |
| | 200 | 20 | 10 | 600s | **71.6** | 94.7 | 76.9 | 0.21 | **71.6** | 0.00 | **71.6** | 0.07 |
| Non-Eucl 200 | 200 | 20 | 10 | 600s | **41.0** | 76.6 | 41.1 | 0.02 | **41.0** | 0.00 | **41.0** | 0.00 |
| | 200 | 20 | 10 | 600s | **52.8** | 75.3 | 86.9 | 5.38 | **52.8** | 0.00 | **52.8** | 0.13 |
| | 500 | 100 | 5 | 600s | 143.7 | 203.2 | 200.3 | 4.44 | 152.5 | 3.69 | **140.3** | 6.79 |
| Non-Eucl 500 | 500 | 100 | 5 | 600s | **132.7** | 187.3 | 194.3 | 1.20 | 148.6 | 4.27 | 144.3 | 6.76 |
| | 500 | 100 | 5 | 600s | 162.3 | 197.4 | 205.6 | 0.00 | 166.1 | 2.89 | **162.0** | 1.87 |
| | 600 | 20 | 30 | 600s | **14.5** | 59.4 | 22.7 | 1.49 | 15.6 | 1.62 | 16.4 | 2.41 |
| Non-Eucl 600 | 600 | 20 | 30 | 600s | 17.7 | 23.7 | 22.0 | 0.82 | **16.1** | 1.24 | **16.1** | 1.28 |
| | 600 | 20 | 30 | 600s | **15.1** | 29.5 | 22.1 | 0.44 | 16.0 | 1.66 | 15.8 | 1.48 |

some predefined CPU time limits are given in the experiments on a Pentium IV 2.8 GHZ PC [9]. Therefore, in order to normalize the running time, a scheme is used according to the well-known SPEC benchmark (Standard Performance Evaluation Corporation, www.specbench.org/osg/cpu2000/), which indicates that Pentium D 2.66 GHz is nearly 1.13 times faster than Pentium IV 2.8 GHZ (see Appendix). Hence, the CPU time used in DASA is limited to 88.3% of the one used in VNS.

Table 1 and 2 show the results of the experiments on the TSPLIB instances and other instances, respectively. Columns 1-3 in both Table 1 and Table 2 present instances names, number of nodes, and number of clusters, respectively. Column 4 in Table 2 shows the average number of nodes per cluster. Column 4 in Table1 (column 5 in Table 2) presents the time limits for the heuristics (except SA). In columns 5-11 of Table 1 (columns 6-11 of Table 2), we present the computational results reported by Hu et al. [9]. The column GA in Table 1 shows the results obtained by the GA of Golden et al. [5], where just some smaller TSPLIB instances are tested. Results of GA are unavailable for instances in Table 2.

It can be found from Table 1 and Table 2 that TS2 obtains 21 best results out of the 46 instances, VNDS obtains 4 best results, SA obtains no best results, VNS obtains 25 best results, and DASA obtains 29 best results. DASA outperforms other algorithms for most of the instances. Since VNS is the best heuristic algorithm reported in the literature [9], we will compare DASA with VNS in more details. Table 1 illustrates that DASA can obtain better results on 6 TSPLIB instances than VNS, while VNS can obtain better results on 3 TSPLIB instances. Both DASA and VNS can obtain best solutions on the same 7 TSPLIB instances. In addition, the standard deviations of solutions by DASA are better than that by VNS for most TSPLIB instances. Table 2 shows that DASA gets better results on 8 instances than VNS for both grouped Euclidean and non-Euclidean instances, while VNS gets better solutions on 3 instances than DASA for both grouped Euclidean and non-Euclidean instances. An exception is that no best result is found by DASA for random Euclidean instances, while best results can be found by VNS on 5 random Euclidean instances. The poor performance of DASA on random Euclidean instances may be caused by the hardness to find effective and informative candidate sets for such instances.

## 6. CONCLUSIONS

In this paper, we show that it's NP-hard to obtain the muscle for GMST. After that, a new heuristic algorithm named DASA is proposed to efficiently solve GMST. In future work, we will investigate the reasons why DASA works poorly on random Euclidean instances. Furthermore, more robust and efficient local search operators will be designed to accelerate DASA for solving larger instances. In addition, we will also explore the potential applications of the muscle on other NP-hard problems.

## 7. ACKNOWLEDGMENTS

Our work is partially supported by the NSFC under Grant No. 60805024, the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20070141020.

## APPENDIX

According to Table 3, Pentium IV 2.80 GHz: Intel Pentium D 820 2.8 GHz = 1166 / 1321=0.883.

**Table 3. CPU Benchmark from SPEC**

| | Pentium IV 2.80 GHz | Intel Pentium D 820 2.80 GHz |
|---|---|---|
| SPECint 2000 | 1166 | 1321 |

## REFERENCES

[1] Myung, Y.S., Lee, C.H., and Tcha, D.W. 1995. On the generalized minimum spanning tree problem. Networks 26, 4 (May 1995), 231–241. DOI= http://dx.doi.org/10.1002/net.3230260407

[2] Feremans, C. 2001. Generalized spanning trees and extensions. Doctoral Thesis. Université Libre de Bruxelles, Belgium.

[3] Pop, P.C. 2002. The generalized minimum spanning tree problem. Doctoral Thesis. University of Twente, Netherlands.

[4] Ghosh, D. 2003. Solving medium to large sized Euclidean generalized minimum spanning tree problems. Technical report NEP-CMP-2003-09-28. Indian Institute of Management, Research and Publication Department, India.

[5] Golden, B., Raghavan, S., and Stanojevic, D. 2005. Heuristic search for the generalized minimum spanning tree problem. INFORMS J. Comput. 17, 3 (Summer 2005), 290-304. DOI=http://dx.doi.org/10.1287/ijoc.1040.0077

[6] Wang, Z., Che, C.H., and Lim, A. 2006. Tabu search for generalized minimum spanning tree problem. In Proceedings of 9th Pacific Rim International Conference on Artificial Intelligence (Guilin, China, Aug. 07 - 11, 2006). PRICAI '06. Springer-Verlag, Berlin / Heidelberg, 918–922. DOI= http://dx.doi.org/10.1007/978-3-540-36668-3_106

[7] Öncan, T., Cordeau, J.-F., and Laporte, G. 2008. A tabu search heuristic for the generalized minimum spanning tree problem. Eur. J. Oper. Res. 191, 2 (Dec. 2008), 306-319. DOI=http://dx.doi.org/10.1016/j.ejor.2007.08.021

[8] Hu, B., Leitner, M., and Raidl, G.R. 2005. Computing generalized minimum spanning trees with variable neighborhood search. In Proceedings of the 18th Mini-Euro Conference on Variable Neighborhood Search (Tenerife, Spain, 2005).

[9] Hu, B., Leitner, M., and Raidl, G.R. 2008. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. J. Heuristics 14, 5 (Nov. 2008), 473-499. DOI= http://dx.doi.org/10.1007/s10732-007-9047-x

[10] Jiang, H., Xuan, J.F., and Zhang X.C. 2008. An approximate muscle guided global optimization algorithm for the three-index assignment problem. In Proceedings of the 2008 IEEE Congress on Evolutionary Computation (HongKong, China, Jun. 01 - 06, 2008). CEC '08. IEEE Computer Society Press, Piscataway, NJ, 2404-2410. DOI= http://dx.doi.org/10.1109/CEC.2008.4631119

[11] Kruskal, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. 7, 1 (Feb. 1956), 48–50.

[12] Boese. K. D. 1995. Cost versus distance in the traveling salesman problem. Technical Report CSD-950018. UCLA Computer Science Department.

[13] Glover, F. 1994. Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). Discrete Appl. Math. 49, 1-3 (Mar. 1994), 231–255. DOI= http://dx.doi.org/10.1016/0166-218X(94)90211-9