HEURISTIC APPROACHES FOR THE NO-DEPOT K-TRAVELING SALESMEN PROBLEM WITH A MINMAX OBJECTIVE

A Thesis

by

BYUNGSOO NA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2006

Major Subject: Industrial Engineering

HEURISTIC APPROACH FOR THE NO-DEPOT K-TRAVELING SALESMEN PROBLEM WITH A MINMAX OBJECTIVE

A Thesis

by

BYUNGSOO NA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Sergiy Butenko
Committee Members,	Illya Hicks
	Yunlong Zhang
Head of Department,	Brett A. Peters

May 2006

Major Subject: Industrial Engineering

ABSTRACT

Heuristic Approaches for the No-Depot k-Traveling Salesmen Problem with a Minmax Objective. (May 2006)

Byungsoo Na, B.S., Pohang University of Science & Technology

Chair of Advisory Committee: Dr. Sergiy Butenko

This thesis deals with the *no-depot* minmax Multiple Traveling Salesmen Problem (MTSP), which can be formulated as follows. Given a set of n cities and k salesmen, find k disjoint tours (one for each salesmen) such that each city belongs to exactly one tour and the length of the longest of k tours is minimized. The *no-depot* assumption means that the salesmen do not start from and return to one fixed depot. The no-depot model can be applied in designing patrolling routes, as well as in business situations, especially where salesmen work from home or the company has no central office. This model can be also applied to the job scheduling problem with n jobs and k identical machines.

Despite its potential applicability to a number of important situations, the research literature on the no-depot minmax k-TSP has been limited, with no reports on computational experiments. The previously published results included the proof of NP-hardness of the problem of interest, which motivates using heuristics for its solution. This thesis proposes several construction heuristic algorithms, including greedy algorithms, *cluster first and route second* algorithms, and *route first and cluster second* algorithms. As a local search method for a single tour, 2-opt search and Lin-Kernighan were used, and for a local search method between multiple tours, relocation and exchange (edge heuristics) were used. Furthermore, to prevent the drawback of trapping in the local minima, the *simulated annealing* method is used.

Extensive computational experiments were carried out using TSPLIB instances. Among construction algorithms, *route first and cluster second* algorithms including *removing two edges* method performed best. In terms of running time, *clustering first and routing second* algorithms took shorter time on large-scale instances. The simulated annealing could produce better solutions than the descent method, but did not always perform well in terms of average solution. To evaluate the performance of the proposed heuristic methods, their solutions were compared with the optimal solutions obtained using a mixed-integer programming formulation of the problem. For small-scale problems, heuristic solutions were equal to the optimal solution output by CPLEX.

ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my advisor, Professor Sergiy Butenko. He has inspired me to choose the research topic in the first place and to have creative thinking during the research. Besides the research, he always encouraged me when I was having through hard time. Without his guidance and persistent help this thesis would not have been possible. I would like to thank my committee members, Professor Illya Hicks and Professor Yunlong Zhang, whose valuable advice enhanced my thesis and gave me the idea about my future work related to this thesis.

I thank Balabhaskar Balasundaram, who sincerely corrected my written thesis. I would also like to thank Yuanchang Xie. During the meta heuristic optimization class, he helped me to develop better construction algorithms working together on the course project, which became the basis of my thesis. Thanks are also due to Dr. SangHo Kwon, who conveyed the knowledge of the classical TSP and advised me whenever I encountered a difficult situation. I would like to acknowledge my friends and colleagues at Industrial & Systems Engineering of Texas A&M University, Seungho Lee, Wonju Lee, Kyungnam Ha, Daeheon Choi, and Soondo Hong.

I do not have enough room to express adequate thanks to my beautiful and lovely wife, Hyoim. She has been always supportive and patiently waited for me when I returned home late. She has anxiously awaited the completion of my research. I am the luckiest man in the world being with her. I love you, Hyoim.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The Traveling Salesman Problem (TSP) is a classical combinatorial optimization problem, which seeks to minimize the distance that a salesman travels while visiting each city in a given set exactly once and then returning to the original city. Its various extensions have also been considered in the literature. For example, the *Multiple Traveling Salesman Problem* (MTSP) is used to model a situation where more than one salesmen need to be routed. In MTSP with k salesmen, or k-TSP, a feasible solution is given by k disjoint tours, such that each city belongs to exactly one tour. The objective of the *minsum* MTSP is to minimize the sum of distances traveled by all salesmen.

The minsum MTSP model may be useful in some situations, however, it has some drawbacks that limit its applicability. In particular, the lengths of tours traveled by different salesmen may be significantly different, which results in unfair workload distribution. On the other hand, the *minmax* MTSP model, whose objective is to minimize the length of the longest tour traveled by a salesman, is expected to distribute workloads more uniformly. Moreover, it minimizes the "makespan" of visiting all the cities if travel time is used instead of distances. Thus, the *minmax* criterion is especially useful in situations when the distance traveled by a salesman should not exceed a given limit. In this regard, the *minmax* MTSP is similar to the *Capacitated Vehicle Routing Problem* (CVRP) studied in [1, 2]. Figure 1 and Table I illustrate the difference between variations of MTSP.

This thesis deals with the *no-depot* minmax Multiple Traveling Salesmen Prob-

This thesis follows the style of *IEEE Transactions on Automatic Control.*

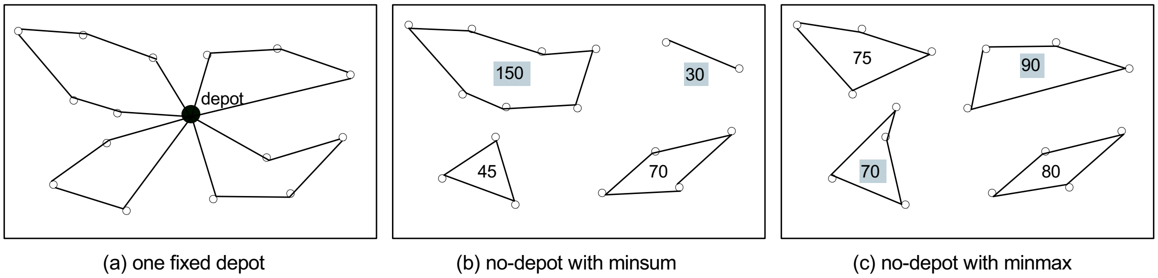


Fig. 1. Variations of k-TSP.

Table I. Variations of k-TSP

	Single TSP	One Fixed Depot TSP	No-depot Minsum TSP	No-depot Minmax TSP
Number of Salesmen	1	k	k	k
objective	min tour length	min sum of tour lengths	min sum of tour lengths	min length of longest tour
depot	-	one fixed	not fixed	not fixed

lem (MTSP). The *no-depot* assumption means that there is no requirement that all salesmen should start from and return to one fixed depot, which is often used in other MTSP models. The no-depot model can be applied in designing patrolling routes, as well as in business situations, especially where salesmen work from home or the company has no central office. This model can be also applied to the job scheduling problem with n jobs and k identical machines.

Despite its potential applicability to a number of important situations, the research literature on the no-depot minmax k-TSP has been very limited, with no reports on computational experiments. The previously published results included the proof of NP-hardness of the problem of interest, which motivates using heuristics for its solution. This thesis proposes several construction heuristic algorithms, including greedy algorithms, *cluster first and route second* algorithms, and *route first and* *cluster second* algorithms. As a local search method for a single tour, 2-opt search and Lin-Kernighan were used, and for a local search method between multiple tours, relocation and exchange (edge heuristics) were used. Furthermore, to prevent the drawback of trapping in the local minima, the *Simulated Annealing* method is used.

Extensive computational experiments were carried out using TSPLIB instances. Among construction algorithms, *route first and cluster second* algorithms including *removing two edges* method performed best. In terms of running time, *clustering first and routing second* algorithms took shorter time on large-scale instances. And the Simulated Annealing could produce better solutions than the descent method, but didn't always perform well in terms of average solution. To evaluate the performance of the proposed heuristic methods, their solutions were compared with the optimal solutions obtained using a mixed-integer programming formulation of the problem. For small-scale problems, heuristic solutions were equal to the optimal solution output by CPLEX.

The chapters of this thesis are organized as follows. The reminder of the current chapter is used to introduce the definitions and notations used throughout the thesis and to review the existing research literature concerning the minmax MTSP. Chapter II describes the proposed construction heuristics for the problem of interest. Tour improvement strategies are introduced in Chapter III. Chapter IV reports the results of numerical experiments performed, and Chapter V concludes the thesis with a discussion of the obtained results and directions for future work.

A. Definitions and Notations

Let $G = (V, E)$ be a complete undirected graph, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices and $E = \{(v_i, v_j) : v_i \neq v_j\}$ is the set of edges. Every edge (v_i, v_j) has an associated weight $c(i,j)$, and all the weights form a matrix $C = [c(i,j)]_{i,j=1}^{n}$. In the TSP context, a vertex can be interpreted as a city and the edge weight can be the distance between the cities or the time of travel between the two cities. With these notations, the classical Traveling Salesman Problem (TSP), which is to find a minimum-weight tour T that visits each city exactly once and returns to the starting point (depot), can be formulated as follows:

$$\min \; c(T(n), T(1)) + \sum_{i=1}^{n-1} c(T(i), T(i+1))$$

where

$T(i)$ is the i^{th} city in the tour;

$c(T(i), T(i+1))$ is the distance from the i^{th} city to the $(i+1)^{th}$ city.

As motivated above, this thesis considers the *minmax* optimization criterion for the MTSP. This problem can be formulated as follows:

$$\min \max_{1 \le j \le k} \{ \; c(T_j(n_j), T_j(1)) + \sum_{i=1}^{n_j - 1} c(T_j(i), T_j(i+1)), \; \}$$

where

$T_j(i)$ is the i^{th} city in the j^{th} tour;

$c(T_j(i), T_j(i+1))$ is the distance from the i^{th} city to the $(i+1)^{th}$ city in the j^{th} tour;

n_j is the number of cities in the j^{th} tour;

k is the number of salesmen;

$n = \sum_{j=1}^{k} n_j$ is the total number of cities.

B. Integer Programming Formulation

In general, most combinatorial problems can be represented as the integer programming formulation. We modified Miller-Tucker-Zemlin (MTZ) formulation of the TSP [3] and proposed IP formulation for a no-depot k-TSP with minmax objective. This formulation will be used to compute the exact solution of small test instances and to examine the performance of heuristic algorithms.

1. IP Formulation for the TSP

The Miller-Tucker-Zemlin (MTZ) formulation of the TSP [3] is given by:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1.1}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \tag{1.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j \tag{1.3}$$

$$u_1 = 1, \tag{1.4}$$

$$2 \le u_i \le n, \quad \forall i \neq 1 \tag{1.5}$$

$$u_i - u_j + 1 \le (n-1)(1-x_{ij}), \quad \forall i \neq 1, \forall j \neq 1 \tag{1.6}$$

$$x_{ij} \in \{0,1\} \quad \forall i,j \tag{1.7}$$

- Indices and Parameters

  i: departing city, $i \in \{1,\ldots,n\}$;

  j: destination city, $j \in \{1,\ldots,n\}$;

  n: total number of cities;

- Decision variables

  $$x_{ij} : x_{ij} = \begin{cases} 1, & \text{if the arc from city } i \text{ to city } j \text{ is on the tour}, \; i,j \in \{1,\ldots,n\} \\ 0, & \text{otherwise} \end{cases}$$

u_i: extra variables to exclude sub tours, $i \in \{1, \ldots, n\}$.

The constraints of (1.2) and (1.3) are called the *degree constraints*, which enforce that every vertex is entered and left exactly once. The constraints of (1.4), (1.5) and (1.6) are *subtour elimination constraints*, which prohibit the formation of subtours having less than n vertices.

2. IP Formulation for No-depot k-TSP with Minmax Objective

$$
\begin{aligned}
& \min y && && (1.8)\\
\text{s.t.} \quad & y && \geq \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ijk}, \quad \forall k && (1.9)\\
& \sum_{k=1}^{K}\sum_{j=1}^{n} x_{ijk} && = 1, \quad \forall i && (1.10)\\
& \sum_{k=1}^{K}\sum_{i=1}^{n} x_{ijk} && = 1, \quad \forall j && (1.11)\\
& \sum_{i=1}^{n} x_{ijk} && = \sum_{i=1}^{n} x_{jik}, \quad \forall j, k && (1.12)\\
& \sum_{i=1}^{n} v_i && = K && (1.13)\\
& 2 - v_i && \leq u_i \leq n, \quad \forall i && (1.14)\\
& u_i - u_j - n(v_i + v_j) + 1 && \leq (n-1)(1 - \sum_{k=1}^{K} x_{ijk}), \quad \forall i, j && (1.15)\\
& v_i, x_{ijk} && \in \{0, 1\} \quad \forall i, j, k && (1.16)
\end{aligned}
$$

- Additional Indices and Parameters

  k: the number of assigned salesmen, $k \in \{1, \ldots, K\}$;

  K: total number of salesmen.

- Decision variables

  y : objective value, the distance of longest traveled salesman's tour;

$$x_{ijk}\colon\ x_{ijk} = \begin{cases} 1, & \text{if the trip from city } i \text{ to city } j \text{ is assigned to the salesman } k \\ 0, & \text{otherwise} \end{cases}$$

v_i : extra variables to generate sub tours, $i \in \{1, \ldots, n\}$;

u_i : extra variables to generate sub tours, $i \in \{1, \ldots, n\}$.

C. Applications

The no-depot multiple traveling salesmen problem can be applied in a number of scenarios. If some cities are located far away from the headquarters of the company, it may be expensive to return to there. In this case, the company may have the policy that salesmen do not need to return to the headquarter office or even can choose the location of his office among his touring cities. On the other hand, if the company has the policy that all salesmen should start at and return to the fixed depot, different problem formulation should be used.

This problem can also be applied to patrol routing. Bugera [4] mentioned *Submarine Routing Problem* as one possible application. If there are limited submarines that should monitor several specific locations, no-depot k-TSP can assign the patrol route to each submarine to have every location covered by one submarine. Likewise, it can be applied to rescue operations and border patrolling.

Another application is the job scheduling problem. If there are n jobs and k identical parallel machines, the objective is to minimize the makespan. A salesman in the *minmax* k-TSP can represent a machine, then the tour of a salesman can be the job sequence of a machine. Additionally, the distance matrix of TSP corresponds to the setup time from one job to another job processed by the machine. A weighted vertex in TSP can represent a stay time or working time of a salesman in that city. In the job scheduling, the weighted vertex is the processing time of a job by a machine.

Table II. Application *minmax* k-TSP to job scheduling

Graph Expression	*Minmax* k-TSP	Job Scheduling
vertex	city	job
weighted vertex	staying time of salesman	processing time by machine
edge	distance between two cities	setup time between two jobs
–	salesman	machine
–	tour of salesman	job sequence of machine
–	*min* longest tour	*min* makespan

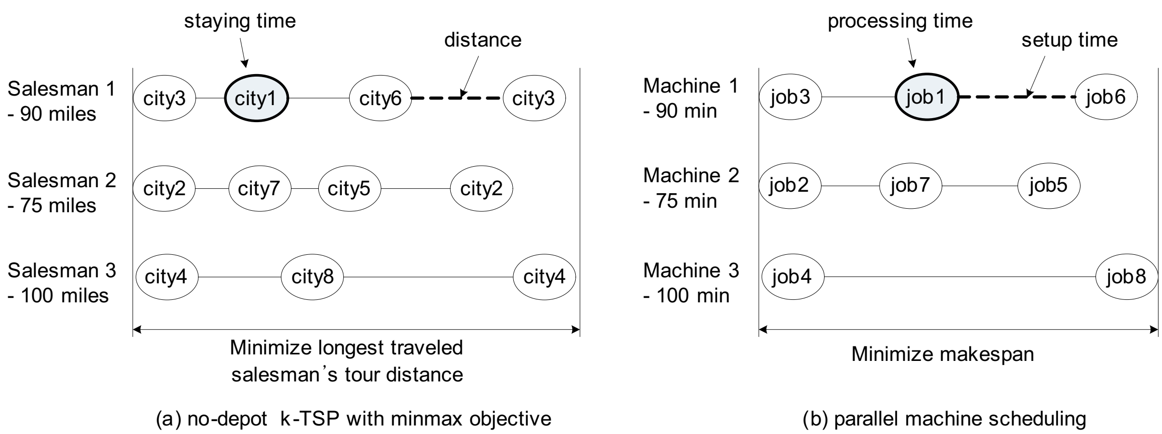


Fig. 2. Graphic comparison with job scheduling.

Table II shows the corresponding relation when k-TSP is applied to the job scheduling problem. And Figure 2 represents a graphical comparison between k-TSP and job scheduling problem.

D. Literature Review

The research of MTSP can be traced back to the idea of Bellmore and Hong [5]. They proved that a multiple TSP with n cities and k salesmen can be transformed to a

TSP with $n+k-1$ cities and single salesman. They applied the minsum criterion. However, this idea holds only when all salesmen have same fixed depot.

Frederickson et al. [6] were the first to study the minmax k-TSP with a single starting vertex (depot). They suggested two approximation algorithms. The first method produces k subtours simultaneously using the nearest neighbor and nearest insertion algorithms. The approximation ratio of the nearest neighbor is $k+(k/2)\log n$, and that of the nearest insertion is two. The second method builds single salesman tour first, then splits it into k subtours. Its approximation ratio is $e+1-1/k$, where e is the bound of approximation factor for the single traveling salesman problem.

Franka et al. [1] proposed a tabu search heuristic and two exact algorithms for the minmax k-TSP with a single fixed depot. They used randomly generated points as test instances and showed that their result derived by tabu heuristic is better than that of nearest neighbor algorithm.

Golden et al. [2] presented the tabu search heuristic based on *Adaptive Memory Procedure* for the minmax k-TSP with a single depot. The authors reported computational results on several test instances from VRPLIB, including running time and solution value, but it was restricted to the case of a single fixed depot.

Spriggs [7] proved that the k-TSP is NP-hard and presented several approximation algorithms based on *Minimum Spanning Tree* for the minmax k-TSP with a single unfixed depot. Unlike other versions of TSP, the author allowed to visit each vertex multiple times. Its approximation ratio is at best close to either $4/(1+1/k)$ or 2δ, where δ is the bound of approximation factor for the single traveling salesman problem.

Sofge et al. [8] proposed two level optimization, *Cluster First then Route*, and compared several evolutionary computation algorithms for minmax k-TSP with no-depot. They used a neighborhood attractor schema, a variation of k-means clustering in clustering phase, and a shrink-wrap algorithm in local search phase. Computational results were generated by the following evolutionary algorithms: Genetic algorithm, evolutionary strategy, particle swarm optimization and generational Monte-Carlo optimization.

CHAPTER II

CONSTRUCTION PHASE

In general, there are two main types of heuristics for the classical TSP. First one is the *Tour Construction Heuristics*, which construct an initial solution using Nearest neighbor, Nearest insertion, and so on. Another one is the *Tour Improvement Heuristics*, which improve a previously obtained solution using two-opt, Lin-Kernighan [9], and so on. The advanced, meta heuristic, search strategies, such as TABU search [10], Simulated Annealing [11], and so on, are used to escape local minima of poor quality. The multiple traveling salesmen problem is more complicated than the classical single TSP in terms of constructing an initial solution and improving the tour. These two phases sometimes can be divided distinctly; however, such a distinction sometimes is not necessary in order to get a better solution. Although local search within a single tour belongs to tour improvement phase, it can be used in the construction phase for the MTSP.

In the construction phase, we tried three kinds of construction heuristics: greedy, *cluster first and route second*, and *route first and cluster second*. Each approach considers two different algorithms as seen in Table III.

Table III. Construction phase algorithms

Greedy algorithm	First select k pairs of closest cities
	First select k cities located farthest
Cluster first and Route second	k-center clustering
	k-means clustering
Route first and Cluster second	k dividing algorithm
	Removing two edges algorithm

Table IV. Operations for the set of cities

Operation	Effect
$init(V)$	Initialize the set V as an empty set
$insert(V, c_1)$	Insert a city c_1 to the set V
$remove(V, c_1)$	Remove a city c_1 from the set V
$pop(V)$	Return one member of the set and remove it from the set V
$is\_empty(V)$	If the set V is empty, then return TRUE else FALSE

Table V. Usage of set operations

Usage	Result
$V = init(V)$	$V = \{\}$
$V = insert(V, 4)$	$V = \{4\}$
$V = insert(V, 1)$	$V = \{1, 4\}$
$V = insert(V, 7)$	$V = \{1, 4, 7\}$
$V = insert(V, 5)$	$V = \{1, 4, 5, 7\}$
$V = remove(V, 7)$	$V = \{1, 4, 5\}$
$c = pop(V)$	$c = 1, \quad V = \{4, 5\}$
$bool = is\_empty(V)$	$bool = FALSE$

Before we explain the detail of algorithms, we introduce some notations and terms used throughout this thesis. Recall that V represents the set of cities, $V = \{1, 2, \ldots, n\}$. We define several operations using the set of cities in Table IV and give examples of their usage in Table V. S_j represents the list of cities visited by salesman $j, j \in \{1, \ldots, k\}$. Table VI and Table VII show the operations and usage of the list of cities.

Table VI. Operations for the list of cities

Operation	Effect
$init(S_j)$	Initialize the list S_j as an empty list
$insert(S_j, c_i)$	Insert a city c_i to the back of the list S_j
$remove(S_j, c_i)$	Remove a city c_i from the list S_j
$num\_cities(S_j)$	Returns the number of cities in the tour S_j
$tour\_length(S_j)$	Return the length of the tour
$get\_city(S_j, i)$	Return the i^{th} city in the list S_j
$local\_search(S_j, A)$	Perform the local search within the tour if $A = LK$, apply Lin-Kernighan if $A = TW$, apply Two-Opt search
$get\_from\_to(S_j, p1, p2)$	Get the partial list of S_j from the city in the position $p1$ to the city in the position $p2$ in the list
$merge(S_i, S_j)$	Return the merged list between list S_i and list S_j if $S_i =< S_{i1} - S_{i2} - \ldots - S_{ik} >$ and $S_j =< S_{j1} - S_{j2} - \ldots - S_{jl} >$, then $merge(S_i, S_j) =< S_{i1} - \ldots - S_{ik} - S_{j1} - \ldots - S_{jl} >$

Table VII. Usage of list operations

Usage	Result
$S_1 = init(S_1)$	$S_1 = \phi$
$S_1 = insert(S_1, 4)$	$S_1 = < 4 >$
$S_1 = insert(S_1, 1)$	$S_1 = < 4 - 1 >$
$S_1 = insert(S_1, 7)$	$S_1 = < 4 - 1 - 7 >$
$S_1 = insert(S_1, 5)$	$S_1 = < 4 - 1 - 7 - 5 >$
$S_1 = insert(S_1, 3)$	$S_1 = < 4 - 1 - 7 - 5 - 3 >$
$S_1 = insert(S_1, 8)$	$S_1 = < 4 - 1 - 7 - 5 - 3 - 8 >$
$S_1 = insert(S_1, 9)$	$S_1 = < 4 - 1 - 7 - 5 - 3 - 8 - 9 >$
$c_1 = get\_city(S_1, 2)$	$c_1 = 1$
$c_2 = get\_city(S_1, 3)$	$c_2 = 7$
$S_2 = get\_from\_to(S_1, 1, 3)$	$S_2 = < 4 - 1 - 7 >$
$S_3 = get\_from\_to(S_1, 6, 7)$	$S_3 = < 8 - 9 >$
$S_4 = merge(S_2, S_3)$	$S_4 = < 4 - 1 - 7 - 8 - 9 >$
$S_5 = remove(S_4, 8)$	$S_5 = < 4 - 1 - 7 - 9 >$
$n = num\_cities(S_5)$	$n = 4$
$dist = tour\_length(S_5)$	$dist =$ (the tour length of S_5)

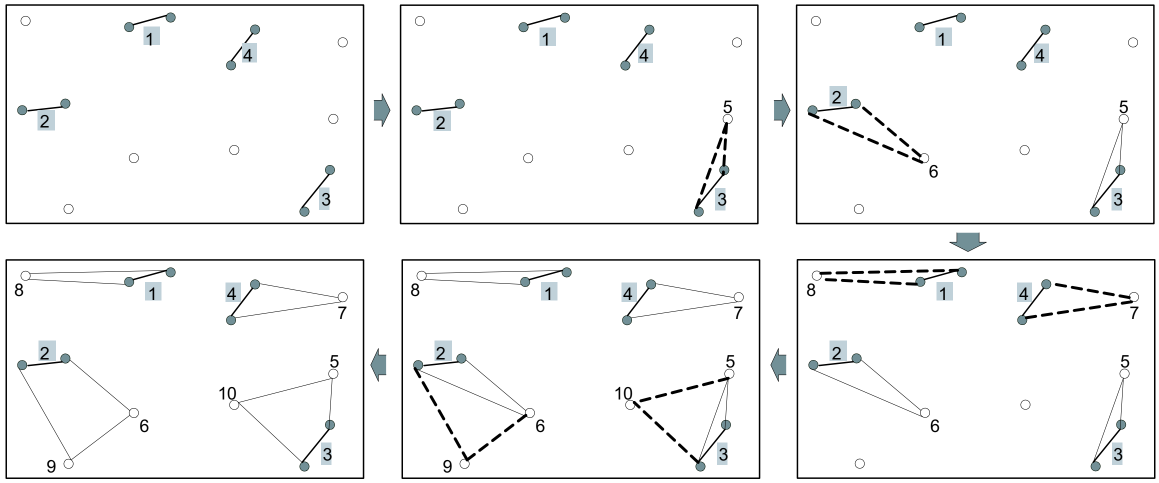


Fig. 3. Construction phase: k pairs of closest cities greedy.

A. Greedy Algorithms

We tried two kinds of greedy algorithms according to initial city-selection for k salesmen. After that, remaining cities are assigned to a proper salesman according to the greedy rule.

1. First Select k Pairs of Closest Cities

This algorithm is based on the idea that as the distance between two cities is shorter, these two cities are more likely to be assigned to the same salesman. Hence, it first connects all cities, then finds the shortest k pair of cities. Based on these k pairs of cities, we can construct k salesmen's tour, each of which has two cities initially. Next step is to attempt allocating each of the remaining cities to a salesman so that the resulting longest tour length is minimized. Figure 3 illustrates *k pairs of closest cities greedy* algorithm and the overall procedure of this algorithm is shown in Algorithm 1.

Data: V is the set of cities

$V = \{1, 2, \ldots, n\}$;

for $j = 1$ *to* k **do**

  $S_j = init(S_j)$;

  $S_j = insert(S_j, k), S_j = insert(S_j, l), where(k, l) \in \{(k, l) \mid d_{kl} = \min d_{pq}, p, q \in V\}$;

  $V = remove(V, k), V = remove(V, l)$;

end

$S_{min} = init(S_{min})$;

while $is\_empty(V)$ *is FALSE* **do**

  $next\_city = pop(V)$;

  $S_{temp} = init(S_{temp})$;

  for $j = 1$ *to* k **do**

    $S_{temp} = insert(S_j, next\_city), S_{temp} = local\_search(S_{temp})$;

    if $tour\_length(S_{temp}) < tour\_length(S_{min})$ **then**

      $S_{min} = S_{temp}$;

      $min\_index = j$;

    end

  end

  $S_{min\_index} = S_{min}$;

end

return $S = \{S_1, S_2, S_3, \ldots, S_k\}$;

Algorithm 1: First select k pairs of closest cities

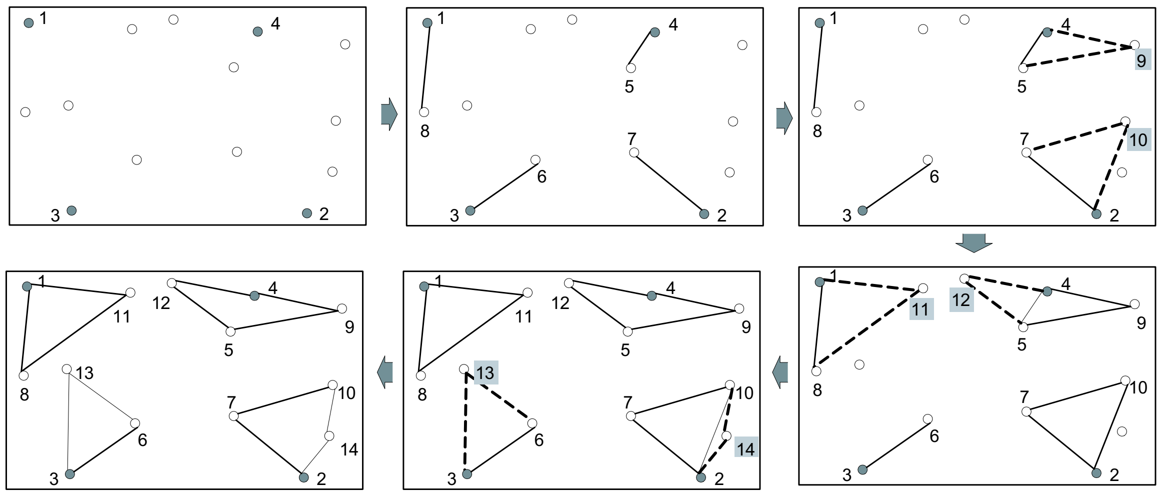


Fig. 4. Construction phase: k-farthest cities greedy.

2. First Select k Cities Located Farthest

In the previous greedy algorithm, if the initial k pairs of cities are very closely located to each other, it might not give a reasonable solution. To overcome this problem, we propose to use another greedy algorithm by modifying the way of generating the initial k salesmen tours. Instead of the shortest k pairs of cities, we try to find k cities located farthest from each other as the initial cities of the k salesmen tours. Figure 4 illustrates *k-farthest cities greedy* algorithm and the procedure of this algorithm is shown in Algorithm 2.

```
Data: C is the set of center cities in each cluster
V = {1, 2, ..., n};
C = init(C);
Randomly select one city c_1, c_1 ∈ V;
C = insert(C, c_1), V = remove(V, c_1);
S_1 = init(S_1), S_1 = insert(S_1, c_1);
for j = 2 : k do
    Select city k farthest from center set C, k ∈ V, where
    k ∈ {k | d_kl = max min d_pq, p ∈ V, q ∈ C};
    C = insert(C, k), V = remove(V, k);
    S_j = init(S_j), S_j = insert(S_j, k);
end
while is_empty(V) is FALSE do
    next_city = pop(V);
    S_temp = init(S_temp);
    S_min = insert(S_1, next_city), S_min = local_search(S_min);
    min_index = 1;
    for j = 2 to k do
        S_temp = insert(S_j, next_city), S_temp = local_search(S_temp);
        if tour_length(S_temp) < tour_length(S_min) then
            S_min = S_temp;
            min_index = j;
        end
    end
    S_min_index = S_min;
end
return S = {S_1, S_2, S_3, ..., S_k};
```

Algorithm 2: First select k cities located farthest

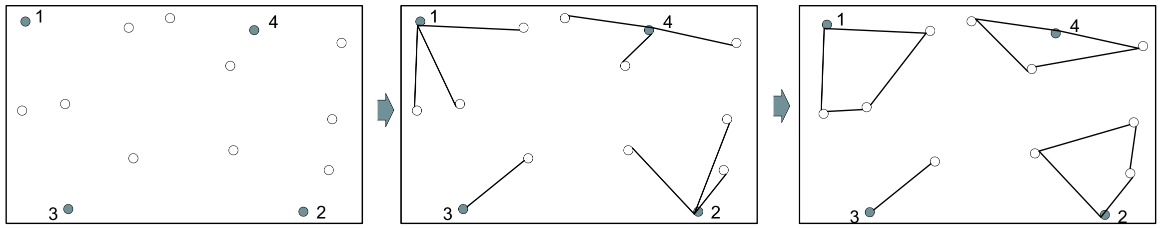


Fig. 5. Construction phase: k-center clustering.

B. Cluster First and Route Second

Since a lot of research have been conducted to find a reasonable solution of single TSP, it makes sense to reduce the multiple TSP to several single TSPs, and then apply algorithms available for the single TSP. The *Cluster first and Route Second* algorithm starts with this idea. First we partition all cities, and assign them to each salesman, then find a (sub)optimal route within a single tour. The number of clusters, k, is the same as the number of salesmen.

1. k-center Clustering

The *k-center clustering* for k-TSP is very straightforward. First we pick one city randomly and make it the center of the first cluster, and we keep picking new cities that are farthest from all the currently chosen cities until k cities are selected. Each city is the center of one cluster. For the remaining cities, we assign each of them to the cluster, whose center is closest to the city to be assigned. Figure 5 illustrates *k-center clustering* algorithm and the procedure of this algorithm is shown in Algorithm 3.

2. k-means Clustering

The *k-means clustering* is one of the widely used methods to partition some data when the number of clusters, k, is predefined. First, randomly generate the initial

Data: C is the set of center cities in each cluster

$V = \{1, 2, \dots, n\}$;

$C = init(C)$;

Randomly select one city $c_1, c_1 \in V$;

$C = insert(C, c_1), V = remove(V, c_1)$;

$S_1 = init(S_1), S_1 = insert(S_1, c_1)$;

for j = *2* : k **do**

  Select city m farthest from center set $C, m \in V$, where

  $m \in \{m \mid d_{ml} = \max\min d_{pq}, p \in V, q \in C\}$;

  $C = insert(C, m), V = remove(V, m)$;

  $S_j = init(S_j)$, $S_j = insert(S_j, m)$;

end

while *is_empty*(V) *is FALSE* **do**

  $v = pop(V)$;

  Find the center city p where $p \in \{p \mid d_{vp} = \min d_{vq},\ p, q \in C\}$;

  $S_j = insert(S_j, v)$, where $p \in S_j, j \in \{1, 2, \dots, k\}$;

end

return $S = \{S_1, S_2, S_3, \dots, S_k\}$;

Algorithm 3: k-center clustering

Data: $C_j(x, y)$ are the coordinates (x, y) of the cluster j's center
$C\_sum_j(x, y)$ is the sum of coordinates (x, y) of the cities in cluster j
$c_j(x)$ and $c_j(y)$ are x and y coordinates of city j, respectively

$V = \{1, 2, \ldots, n\}$, $C = init(C)$;
Randomly select one city c_1, where $c_1 \in V$;
$C = insert(C, c_1)$;
$C_1(x) = c_1(x),\ C_1(y) = c_1(y)$;
for $j = 2 : k$ **do**
  Select city m farthest from center set $C, m \in V$, where
  $m \in \{m \mid d_{ml} = \max\min d_{pq}, p \in V, q \in C\}$;
  $C_j(x) = c_m(x),\ C_j(y) = c_m(y)$;
end
$sum\_dist = \text{inf}$;
while $sum\_dist < \epsilon$ **do**
  $sum\_dist = 0$;
  for $v = 1 : n\ ,\ v \in V$ **do**
    for $j = 1 : k$ **do**
      $S_j = init(S_j), C\_sum_j(x, y) = (0, 0)$;
    end
    Find the center p where $p \in \{p \mid d_{vp} = \min d_{vq},\ p, q \in C\}$;
    $sum\_dist = sum\_dist + d_{vp}$;
    $S_p = insert(S_p, v)$, where $p \in \{1, 2, \ldots, m\}$;
    $C\_sum_p(x) = C\_sum_p(x) + c_v(x)$;
    $C\_sum_p(y) = C\_sum_p(y) + c_v(y)$;
  end
  for $j = 1 : k$ **do**
    $C_j(x) = C\_sum_j(x)/|S_j|$;
    $C_j(y) = C\_sum_j(y)/|S_j|$;
  end
end
return $S = \{S_1, S_2, S_3, \ldots, S_k\}$;

Algorithm 4: k-means clustering

centers of cluster, then assign each data point to the closest cluster center. That data point is now a member of that cluster. Next, calculate the new cluster center to be the average coordinate of all the members of a certain cluster (this step is applicable only for instances, in which each city is given by its coordinates on the plane). And calculate error function to be the sum of within-cluster sum-of-squares. If this value has not significantly changed over a certain number of iterations or cluster membership no longer changes, we consider such clustering final. Algorithm 4 shows the procedure of *k-means clustering*.

C. Route First and Cluster Second

The *Route First and Cluster Second* algorithm performs reversely to the *Cluster First and Route Second* algorithm. First, considering a single TSP, we find locally optimal tour using the local search such as two-opt and Lin-Kernighan. Then, we partition the whole tour into k sub tours. In this paper, two algorithms are proposed according to the partition method.

1. k Dividing Algorithm

This algorithm divides the whole tour into k segments with approximately equal lengths. By connecting the two ends of each segment we can obtain the initial solution with k tours. However, depending on the starting point, we may have different initial solutions. Hence, we compare all the solutions and select the one whose longest tour is the shortest. Figure 6 illustrates *k dividing* algorithm and the procedure of this algorithm is shown in Algorithm 5.

Data: S_{all} is the list of all cities visited by a single salesman;

$S_{all} = local\_search(S_{all})$;
$total\_dist = tour\_length(S_{all})$;
$dividing\_dist = total\_dist/k$;
$current\_opt = \inf$;
for $h = 1$ *to* n **do**
  $tour1 = get\_from\_to(S_{all}, h, n),\ tour2 = get\_from\_to(S_{all}, 1, h)$;
  $tour = merge(tour1, tour2)$;
  $start = 1$;
  for $i = 1$ *to* $k - 1$ **do**
    for $j = start$ *to* n **do**
      $sum\_dist = sum\_dist + dist(tour(j), tour(j + 1))$;
      if $sum\_dist > dividing\_dist$ **then**
        $S_i = get\_from\_to(tour, start, j)$;
        $sum\_dist = 0,\ start = j + 1$;
        break;
      end
    end
  end
  $S_m = get\_from\_to(tour, start, n)$;
  for $i = 1$ *to* k **do**
    $S_i = local\_search(S_i)$;
  end
  $max\_dist = \max\ tour\_length(S_i), i \in \{1, 2, \ldots, k\}$;
  if $max\_dist < current\_opt$ **then**
    $S_{opt} = \{S_1, S_2, S_3, \ldots, S_k\}$;
    $current\_opt = max\_dist$;
  end
end
return S_{opt};

Algorithm 5: k dividing algorithm

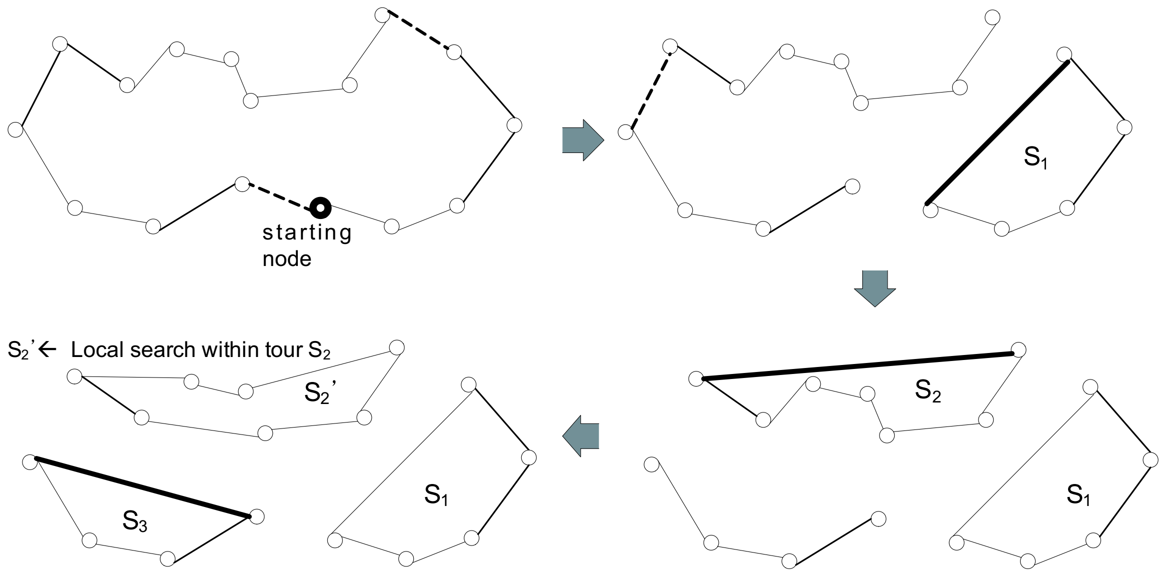


Fig. 6. Construction phase: k dividing algorithm.

2. Removing Two Edges Algorithm

The *removing two edges* algorithm generates locally optimal tour and splits the whole tour into two sub-tours as equally as possible by removing two edges and connecting two ends of each tour. We keep splitting the longest tour into two sub tours until we obtain k tours. Figure 7 illustrates *removing two edges* algorithm and Algorithm 6 shows the procedure of the *removing two edges* algorithm.

Data: S_{all} is the list of all cities visited by a single salesman;
$S_{all} = local\_search(S_{all})$;
$S_1 = S_{all}$;
for $h = 2$ *to* k **do**
  $max\_index = \{l \mid tour\_length(S_l) \geq tour\_length(S_k),\ \forall k \in \{1, 2, \ldots, h-1\}\ \}$;
  $num\_city = num\_cities(S_{max\_index})$;
  for $i = 1$ *to* num_city **do**
    $temp1 = get\_from\_to(S_{max\_index}, i, num\_city)$;
    $temp2 = get\_from\_to(S_{max\_index}, 1, i)$;
    $max\_tour = merge(temp1, temp2)$;
    $min\_dist = \infty$;
    for $j = 1$ *to* $num\_city - 1$ **do**
      $tour1 = get\_from\_to(max\_tour, 1, j)$;
      $tour2 = get\_from\_to(max\_tour, j+1, num\_city)$;
      if $tour\_length(tour1) > tour\_length(tour2)$ **then**
        $longer\_tour = tour1, shorter\_tour = tour2$;
      else
        $longer\_tour = tour2, shorter\_tour = tour1$;
      end
      if $tour\_length(longer\_tour) < min\_dist$ **then**
        $min\_tour1 = longer\_tour, min\_tour2 = shorter\_tour$;
        $min\_dist = tour\_length(longer\_tour)$;
      end
    end
  end
  $S_{max\_index} = min\_tour1$; $S_h = min\_tour2$;
end
return $S = \{S_1, S_2, S_3, \ldots, S_k\}$;

Algorithm 6: Removing two edges algorithm

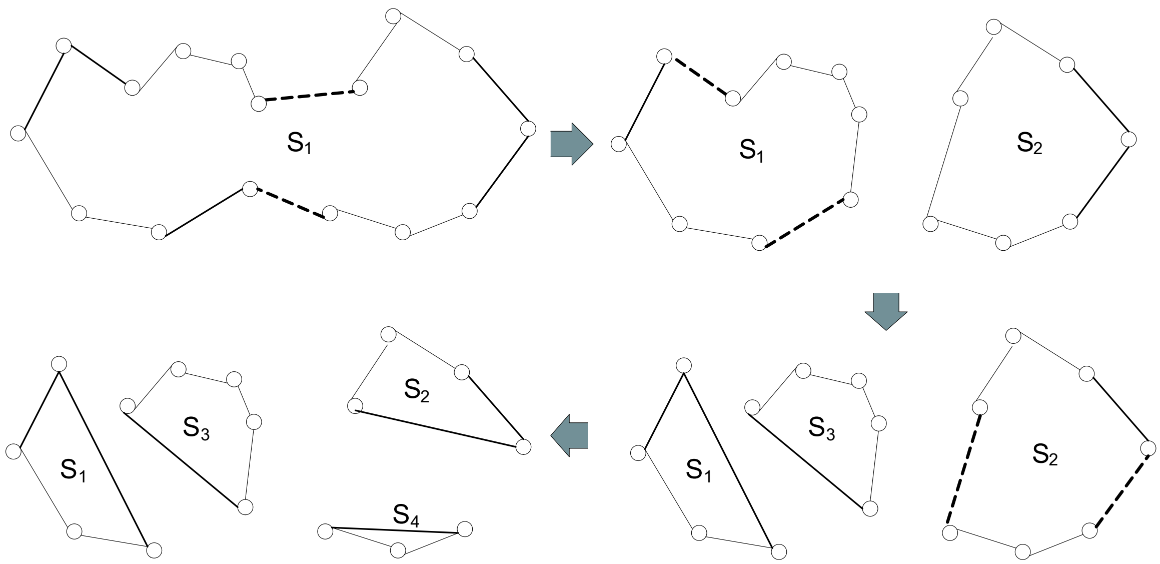


Fig. 7. Construction phase: Removing two edges algorithm.

CHAPTER III

TOUR IMPROVEMENT PHASE

Exchange heuristics are used to improve the current tour. A typical exchange heuristic performs operations (exchanges or moves) that reduce the length of the current tour until a tour is reached for which no operation yields an improvement. According to the scope of exchange, tour improvement phase can be classified into: local search within a single tour and local search between tours. Local search within a single tour is similar to that of classical TSP and hence search methods such as two-opt search and Lin-Kernighan search can be used. Local search between different tours has been used in vehicle routing problem (VRP) to deal with multiple vehicles. The *edge-exchange neighborhood* [16] is a well known method to deal with multiple routes.

A. Local Search Within a Single Tour

The objective of local search within a single tour is to minimize the length of the tour by modifying the tour sequence.

1. 2-Opt Search

The 2-opt algorithm [14] is the simplest among exchange heuristics. From the current tour, it removes two edges (not adjacent) to get two paths. To generate a tour, these paths are connected using edges different from the ones removed. Figure 8 shows typical 2-opt iterations. After applying 2-opt exchange on two non-adjacent edges selected at random from the current tour (Figure 8-b) we get another tour (Figure 8-b'). Suppose the tour distance is larger than the previous tour, this new tour is not stored as the best tour. Two other edges are removed (Figure 8-c) to generate another tour (Figure 8-c'). Suppose this new tour's distance is less than the previous

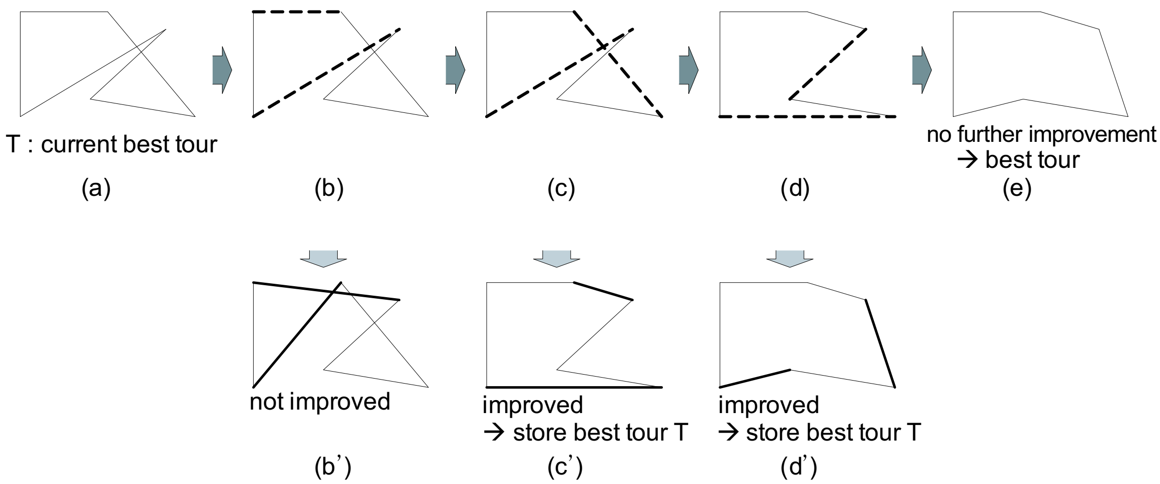


Fig. 8. 2-opt search.

tour, this tour is stored as the best tour. This procedure is repeated until there is no further improvement, say (Figure 8-e) which represents a locally optimal tour.

2. Lin-Kernighan

The Lin-Kernighan heuristic [9] (LK search) is generally considered to be one of the most effective tour improvement methods for the TSP. LK search is similar to k-opt method, but allows for k to be changed. Several versions of Lin-Kernighan algorithm exist and the implementation used here is based on [9, 15]. Figure 9 illustrates typical LK search iterations. Starting at one vertex v_1 of current best tour T (Figure 9-a), we remove one edge v_1u_0 and add another edge from u_0w_0 such that $dist(u_0w_0) < dist(v_1u_0)$ (Figure 9-b). This new graph is called a *δ-path* due to its resemblance to the Greek letter δ. This is not a complete tour (a Hamiltonian circuit), just an incomplete path. Tours are now constructed from this *δ-path*, P_0. Vertex w_0 has three incident edges, w_0u_1, w_0u_2 and w_0u_3. Removing one edge (w_0u_1 or w_0u_2) and adding another edge (v_1u_1 or v_1u_2) results in complete tours (Figure 9-b'). If a new tour has smaller length than the previous tour, that new tour is stored as

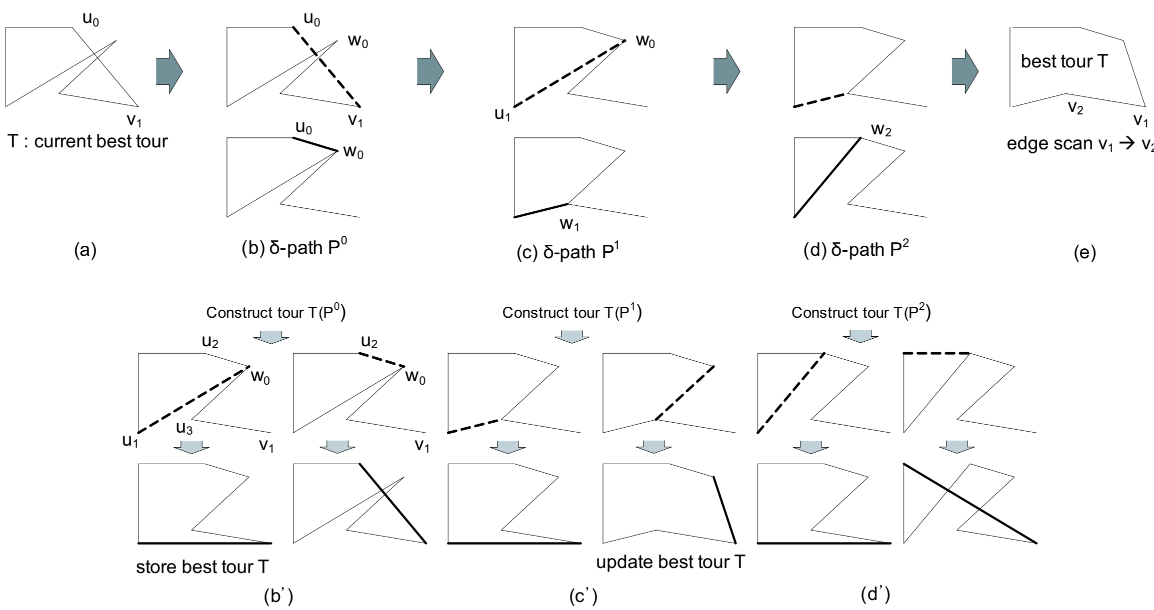


Fig. 9. Lin-Kernighan.

the best tour. Note that removing an edge w_0u_3 cannot make a Hamiltonian circuit. The next step is to construct another *δ-path* P_1 by removing u_1w_0 from the previous *δ-path* P_0. These steps are repeated until no *δ-path* starting at vertex v_1 is found (Figure 9-c,c',d,d'). Once a vertex is fully explored, we move to the next vertex v_2 from v_1 (Figure 9-e) and repeat the above steps. When all vertices have been scanned, LK search is terminated returning the current best tour.

B. Local Search Between Tours

The objective of local search between multiple tours is not only to minimize the length of each salesman's tour, but also to distribute cities among salesman as equally as possible.

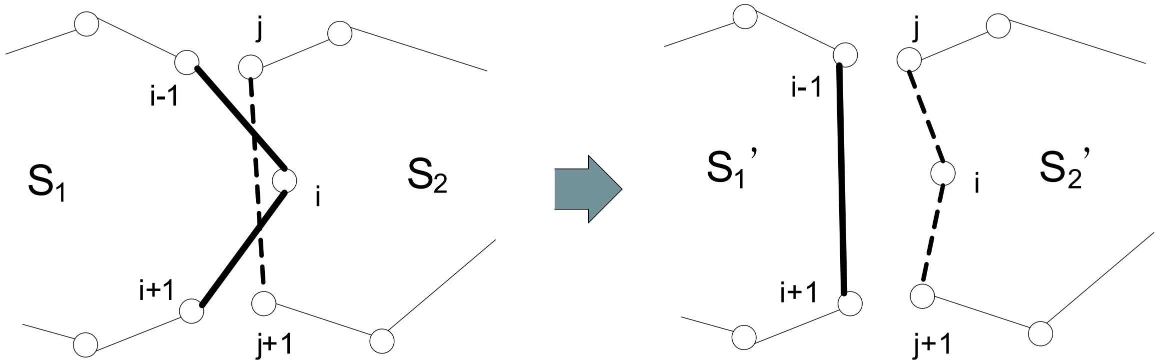


Fig. 10. Tour improvement phase: relocation.

1. Relocation

Consider two salesmen, S_1 and S_2, having the following tour sequences to visit assigned cities; $S_1 =< S_{11} - \cdots - S_{1(i-1)} - S_{1i} - S_{1(i+1)} - \cdots >$ and $S_2 =< S_{21} - \cdots - S_{2j} - S_{2(j+1)} - \cdots >$. Assume that S_1 has longer distance than S_2 and define $diff$ as ($dist(S_1) - dist(S_2)$. Suppose salesman S_1 gives one city, S_{1i} to salesman S_2. The tour of salesman S_1 becomes $S_1' =< S_{11} - \cdots - S_{1(i-1)} - S_{1(i+1)} - \cdots >$. Clearly, the tour distance of S_1 decreases. On the other hand, S_2 accepts city S_{1i} so that his tour now becomes $S_2' =< S_{21} - \cdots - S_{2j} - S_{1i} - S_{2(j+1)} - \cdots >$ which is longer than S_2. If the amount of the increase(i.e. $dist(S_2') - dist(S_2)$) is less than $diff$ (previous difference in total distance between S_1 and S_2), we have $max(dist(S_1'), dist(S_2')) < max(dist(S_1), dist(S_2))$. Under this assumption, relocation decreases the distance of the longer tour among S_1 and S_2. However, if the amount of the increase(i.e. $dist(S_2') - dist(S_2)$) is not less than $diff$, relocation operation is not executed. Figure 10 illustrates relocation procedure.

```
trialN = 0;    failN = 0;
while trialN < maxTrial & failN < maxFail do
    Select two salesmen S_src and S_tgt randomly such that
    tour_length(S_src) ≥ tour_length(S_tgt) ;
    diff = tour_length(S_src) − tour_length(S_tgt);
    for i = 1 to num_cities(S_src) do
        break_flag = 0;
        for j = 1 to num_cities(S_tgt) do
            new_dist = dist(v_j v_i) + dist(v_i v_{j+1});
            increase = new_dist − dist(v_j v_{j+1});
            if increase < diff then
                Relocate the city i to S_tgt located between j and j + 1;
                failN = 0;
                break_flag = 1;
                break;
            end
        end
        if break_flag = 1 then
            break;
        end
    end
    trialN = trialN + 1;
    failN = failN + 1;
end
```

Algorithm 7: Local search between tours : Relocation

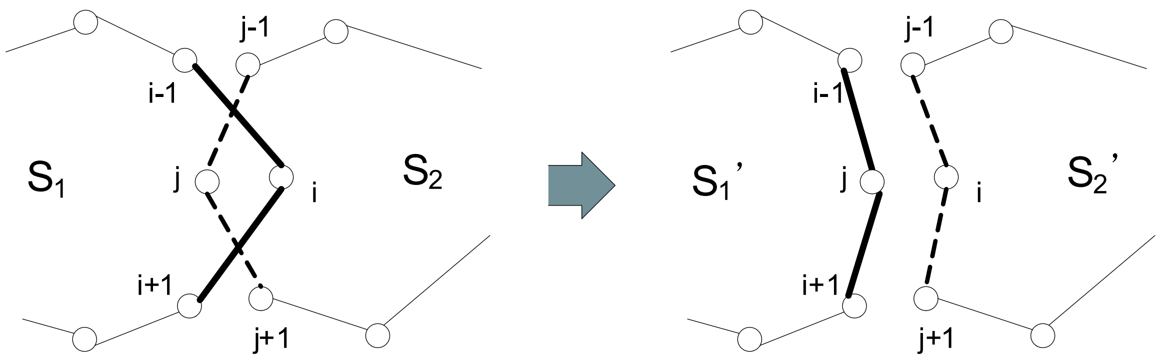


Fig. 11. Tour improvement phase: exchange.

2. Exchange

Similar to relocation, assume that salesmen S_1 and S_2 have the following tour sequences to visit assigned cities: $S_1 =< S_{11} - \cdots - S_{1(i-1)} - S_{1i} - S_{1(i+1)} - \cdots >$ and $S_2 =< S_{21} - \cdots - S_{2(j-1)} - S_{2j} - S_{2(j+1)} - \cdots >$. Exchange operation makes the salesmen exchange cities S_{1i} and S_{2j} with each other. The result of exchange operation will be as follows: $S_1' =< S_{11} - \cdots - S_{1(i-1)} - S_{2j} - S_{1(i+1)} - \cdots >$ and $S_2' =< S_{21} - \cdots - S_{2(j-1)} - S_{1i} - S_{2(j+1)} - \cdots >$. If $max(dist(S_1'), dist(S_2'))$ is less than $max(dist(S_1), dist(S_2))$, the distance of the longer tour among S_1, S_2 decreases. Otherwise, the exchange operation is not executed. Figure 11 illustrates exchange procedure.

C. Advanced Search Strategies

Local search algorithms are based on the descent property. But they are likely to be trapped in a local optimum failing to reach global optimum. Hence, advanced search methods such as Tabu Search [10], Simulated Annealing and GRASP (Greedy Randomized Adaptive Search Procedure) [12] try to overcome this drawback. In this thesis, results from Simulated Annealing approach will be compared with the descent

```
trialN = 0;   failN = 0;
while trialN < maxTrial & failN < maxFail do
    Select two salesmen S_src and S_tgt randomly such that
    tour_length(S_src) ≥ tour_length(S_tgt) ;
    for i = 1 to num_cities(S_src) do
        break_flag = 0;
        for j = 1 to num_cities(S_tgt) do
            new_src_dist = tour_length(S_src) − dist(v_{i−1}v_i) − dist(v_i v_{i+1}) +
            dist(v_{i−1}v_j) + dist(v_j v_{i+1});
            new_tgt_dist = tour_length(S_tgt) − dist(v_{j−1}v_j) − dist(v_j v_{j+1}) +
            dist(v_{j−1}v_i) + dist(v_i v_{j+1});
            max_dist = max( new_src_dist, new_tgt_dist );
            if max_dist < tour_length(S_src) then
                Exchange the city i and the city j;
                failN = 0;
                break_flag = 1;
                break;
            end
        end
        if break_flag = 1 then
            break;
        end
    end
    trialN = trialN + 1;
    failN = failN + 1;
end
```

Algorithm 8: Local search between tours : Exchange method.

1. Descent Method

General descent method accepts a solution in the neighborhood of the current solution that provides the best improvement over the current solution. It is terminated when no improving solution exists. It is simpler compared to Tabu Search and Simulated Annealing. Tabu Search has high memory requirements and both Tabu Search and Simulated Annealing take significantly longer time than the general descent method. However, general descent is prone to be trapped in a local optima. Following is is a descent method for finding a local minimum value of a real-valued function f. $N(i)$ denotes the neighborhood of solution i.

1. Choose an initial solution i;
2. Find a best $j \in N(i)$, i.e., $f(j) \leq f(k)$ for any $k \in N(i)$;
3. If $f(j) \geq f(i)$, then stop, and return solution i;
   Else set $i = j$, and go to Step2;

Algorithm 9: Descent Method

2. Simulated Annealing

Simulated Annealing (SA) is a "threshold" algorithm. It is motivated by the physics of the annealing process, the way in which a metal cools and freezes into a minimum energy crystalline structure. Kirkpatrick et al. [13] proposed the basis of this optimization technique for combinatorial problems. The basic idea of Simulated Annealing is to accept all improving solutions while probabilistically accepting worse solutions based on a control parameter that is analogous to temperature in physical annealing. Cooling schedule is a vital component of the Simulated Annealing algorithm. It determines the upper and lower limits of the temperature parameter and the rate at which the temperature is reduced. The algorithm begins at a high temperature, which corresponds to a high probability of accepting worse solutions. As the search progresses, the temperature is gradually decreased, consequently reducing the probability of accepting non-improving solutions. At temperature zero, the algorithm only accepts improving solutions. The algorithm ends when a pre-specified stopping condition is met. By allowing uphill moves (i.e. accepting worse solution) Simulated Annealing attempts to avoid local minima. The pseudo code below gives a simple example of Simulated Annealing.

```
Generate an initial solution S;
Determine the initial temperature T and decrement ratio dT;
while Not meet the Stop Criterion do
    for m = 1 to max_trial_num do
        Choose one solution J from the neighborhood of S;
        if random_number < exp( [f(S) − f(J)]/T ) then
            S = J;
        end
        T = T × dT;
    end
end
return optimal solution S;
```

Algorithm 10: Simulated Annealing

CHAPTER IV

COMPUTATIONAL RESULT

This chapter describes the instances used for testing our algorithms and reports computational results.

- Test Environment

  Tools : C++ with STL (Standard Template Library), CPLEX 9.0, AMPL 8.0

  Computer specs : Intel Pentium 4 3.06GHz Processor, 512MB RAM

  Time unit : Seconds

- Test Instances

  Two dimensional Euclidian TSP instances from TSPLIB [18] were used for testing. Number of cities in the instances are as follows: 48, 52, 100, 127, 264, and 575. Table VIII provides names of test instances, number of cities in each instance and the number of salesmen.

- Test Scenario

  Local search between multiple tours involves randomness. It is likely that different solutions are generated for each run. Hence, for each test instance, 10 runs are conducted to compute the average solution and the best solution. Figure 12 illustrates the test scenario. Six different construction algorithms are used following which both local search methods - local search in a single tour and local search between multiple tours are applied. As an advanced search strategy, we compared the solution of Simulated Annealing with that of the descent method. In Simulated Annealing method, we examine the effect on solution quality as we change control parameters such as initial temperate and temperature decrement ratio.

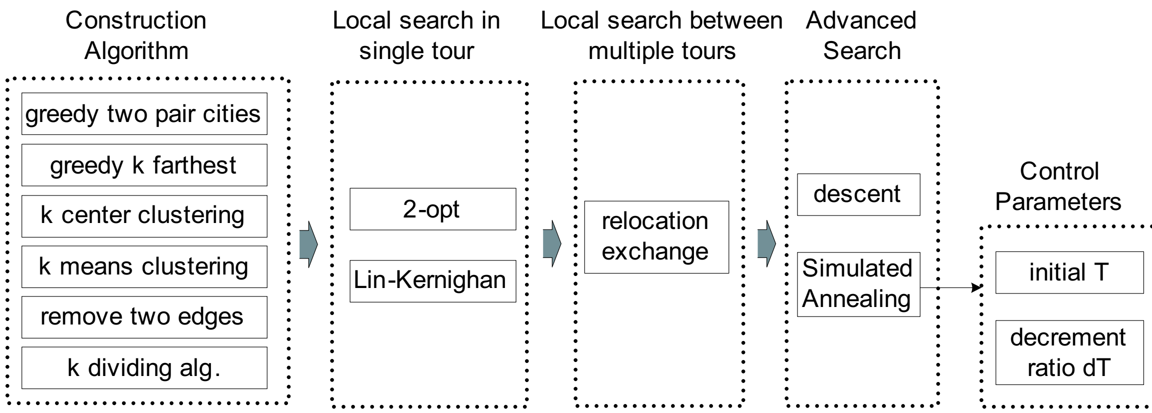


Fig. 12. Test scenario.

Table VIII. Test problems

Problem Name	Number of cities(n)	Number of Salesmen(K)
att48	48	4, 5, 6
berlin52	52	4, 5, 6
kroa100	100	4, 5, 6
bier127	127	4, 6
pr264	264	4, 6
rat575	575	4, 6

The tables in the appendix show the computational result of descent method and simulated annealing.

A. Comparison of Construction Algorithms

A comparison of running times and solution quality of construction algorithms are presented in Figure 13 and Figure 14 respectively for test instance *bier127* with four salesmen. Local search uses Lin-Kernighan neighborhood in a general descent method. In the Figures 13 and 14: Gr2City, Greedy, Kcenter, KMeans, RmvTwo and Rt1st stand for *greedy selecting k pairs of closest cities* algorithm, *greedy selecting k cities located farthest* algorithm, *k-center clustering*, *k-means clustering*, *removing*

Init method	Running Time
Gr2City	5.611
Greedy	5.504
Kcenter	2.308
KMeans	1.949
RmvTwo	5.262
Rt1st	15.438

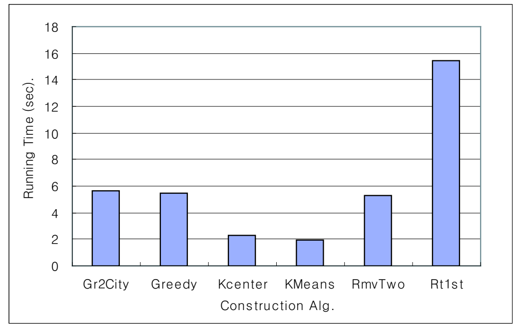


Fig. 13. Construction algorithms vs running time (bier127).

Init method	Init Solution	L.S. Solution
Gr2City	45,298	38,584
Greedy	42,797	35,552
Kcenter	44,570	36,704
KMeans	39,781	35,218
RmvTwo	35,173	32,758
Rt1st	34,597	33,266

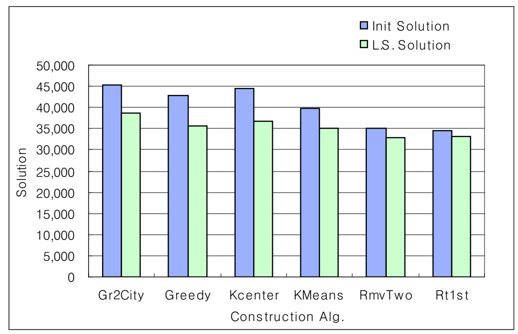


Fig. 14. Construction algorithms vs solution (bier127).

two edges algorithm and *k dividing* algorithm respectively.

Cluster first and route second algorithms such as *k-center clustering* and *k-means clustering* took shorter time while *k dividing* algorithm took the longest time. On the other hand, *k dividing* algorithm produced the best initial solution and *removing two edges* algorithm generated best local search solution. The solution of *greedy selecting k pairs of closest cities* algorithm was worst both in the initial solution and local search solution.

B. Comparison Between Descent Method And SA

Table IX presents a comparison between descent method and simulated annealing in terms of solution quality and running time on *att48* test instance with four salesmen. When the initial temperature was 50 and the temperature decrement ratio was 0.7,

Table IX. Comparison between descent method and SA (att48)

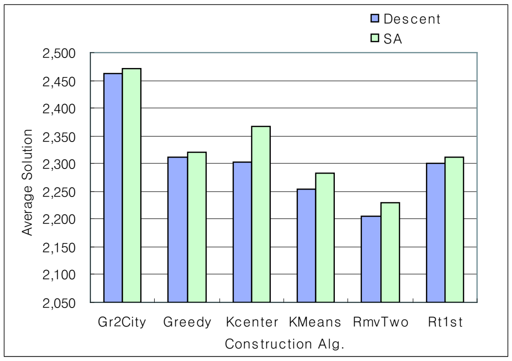
init method	avg running time		average solution		best solution	
	DC	SA	DC	SA	DC	SA
Gr2City	0.574	1.130	10298.4	9988.5	9315	9103
greedy	0.383	0.614	9249.8	9316.6	9103	9103
Kcenter	0.128	0.363	9389.7	9384.6	9318	9103
KMeans	0.122	0.348	9140.3	9352	9103	9103
RmvTwo	0.403	0.650	9129.7	9222.5	9103	9103
Rt1st	0.809	1.049	9702.8	9487.2	9521	9103

Table X. Comparison between descent method and SA (berlin52)

init method	avg running time		average solution		best solution	
	DC	SA	DC	SA	DC	SA (initT, Tratio)
Gr2City	0.595	1.140	2,463.0	2,470.5	2359	2107(100,0.9)
greedy	0.492	0.903	2,311.0	2,321.0	2221	2137(100,0.9)
Kcenter	0.334	0.888	2,302.5	2,367.7	2221	2161(3000,0.8)
KMeans	0.250	0.590	2,254.9	2,281.7	2161	2126(60,0.8)
RmvTwo	0.509	0.904	2,204.3	2,229.2	2182	2118(50,0.6)
Rt1st	1.128	1.461	2,299.6	2,310.8	2231	2118(100, 0.6)

all construction solutions of simulated annealing were better than those of the descent method. But, the average solutions were not always better than those of the descent method. In general, Simulated Annealing took longer time than the descent method.

Table X presents a comparison between descent method and simulated annealing in terms of solution quality and running time on *berlin52* test instance with four salesmen. Under certain specific control parameters, Simulated Annealing failed to generate better solutions than the descent method. However, changing the control parameters, enabled Simulated Annealing to produce better solutions than the descent method. The average solutions were still worse than those of descent method. Figure 15 and Figure 16 are graphical representations of *berlin52*'s comparison.



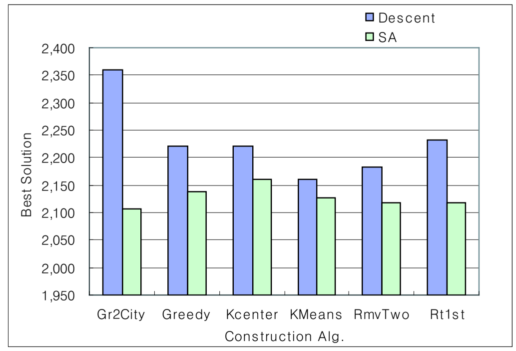


Fig. 15. Solution comparison between SA and descent (berlin52).

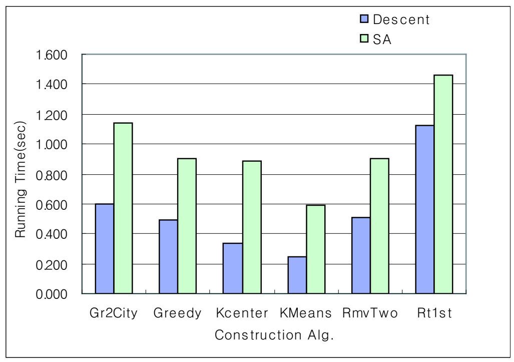


Fig. 16. Time comparison between SA and descent (berlin52).

C. Number of Cities Vs Running Time of Construction Algorithms

The running time of construction algorithms vary with the number of cities in an instance. Table XI and Figure 17 demonstrate this effect when the number of salesmen was four. The running time of k *dividing* algorithm significantly increased as the number of cities increased, whereas the increase in running time of k-*center clustering* and k-*means clustering* was slower. The growth in running time of the two greedy algorithms was in between.

Table XI. The number of cities vs the running time of construction

Init Method	48	52	100	127	264	575
Gr2City	0.36	0.35	1.93	4.16	37.73	501.52
Greedy	0.29	0.35	1.91	3.97	47.65	502.97
Kcenter	0.04	0.07	0.17	0.40	1.11	9.15
KMeans	0.02	0.03	0.11	0.25	0.60	4.75
RmvTwo	0.33	0.37	2.45	4.13	22.38	180.69
Rt1st	0.70	0.90	6.08	14.15	128.90	2,071.35

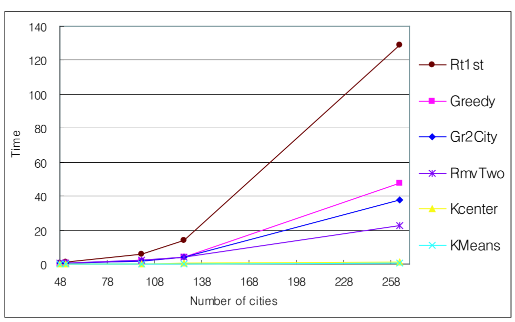


Fig. 17. The number of cities vs the running time of construction.

TRatio	Avg Solution	Best Solution
0.50	2,255.5	2,221
0.60	2,240.5	2,161
0.70	2,255.5	2,161
0.80	2,276.2	2,161
0.90	2,279.2	2,161

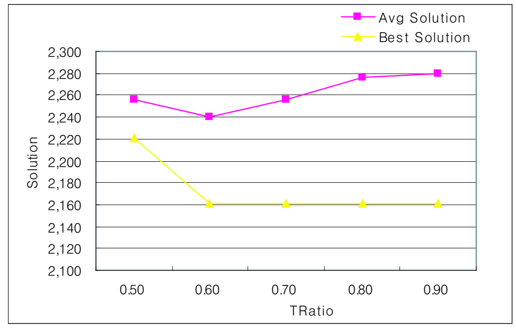


Fig. 18. Comparison by SA's temperature decrement ratio.

D. Comparison of SA's Cooling Schedules

The performance of Simulated Annealing method depends heavily on the choice of cooling schedule and tuning the associated parameters is an important task. The effect of the cooling schedule on solution quality was studied on *berlin52* instance with four salesman and *k-means clustering* as the construction algorithm. First, the initial temperature was fixed at 1000 and the temperature decrement ratio was varied. Figure 18 shows that the average solution was the best when decrement ratio was 0.6. The average solution tends to increase after 0.6. The best solution obtained was consistently the best for 0.6, 0.7, 0.8 and 0.9. Next, the temperature decrement ratio was fixed at 0.70 and the initial temperature was varied. Figure 19 shows that when initial temperature was 1000, the average solution and the best solution were the best.

Initial T	Avg Solution	Best Solution
50	2,306.6	2,235
100	2,295.8	2,235
1,000	2,255.5	2,161
3,000	2,289.5	2,235

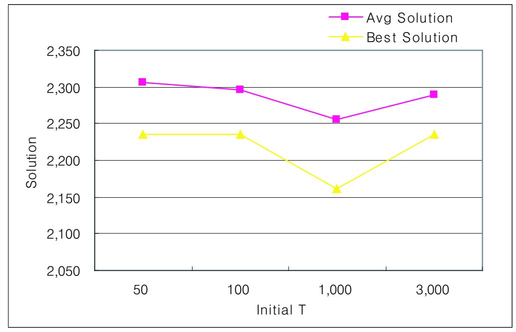


Fig. 19. Comparison by SA's initial temperature.

Table XII. Heuristic solutions vs optimal solutions by CPLEX

	Time		Solution	
	Heuristic	CPLEX	Heuristic	CPLEX
node10	0.053	163.034	3417	3417
node12	0.062	213.452	1496	1496
node15	0.073	7903.394	7264	7264
node20	0.068	out of memory	-	-

E. Optimal Solution by CPLEX

To evaluate the performance of the proposed heuristic algorithms, we compare with the optimal solution obtained by CPLEX in Table XII. Instances with small number of cities were generated. When the number of cities were 10, 12, and 15, the heuristic solutions and CPLEX solutions were same. But CPLEX took longer time than the heuristic run. When the number of cities were more than 20, CPLEX was unable to terminate optimally and due to lack of memory. APPENDIX D shows the no-depot minmax k-TSP AMPL code for solution by CPLEX.

CHAPTER V

DISCUSSION AND CONCLUSION

The proposed algorithms have shown a good performance in terms of running time. When the number of cities is less than 100, it took less than one second. Though the solution and running time varied depending on the construction heuristics and test instances, in general removing two edges algorithm generated best initial and local search solution. A kind of *route first and cluster second* algorithm, *k dividing method* also exhibited good performance. Whereas, greedy algorithms and *cluster first and route second* algorithms did not work as well.

For a larger number of cities, the running time of *k dividing* algorithm significantly increased and the greedy algorithms took longer time. Since these algorithms include the local search for a single tour in the construction phase, this caused the increase in running time. Meanwhile, *k-center clustering* and *k-means clustering* have fast running time even when the number of cities is large. In the aspect of running time, these *cluster first and route second* algorithms will be effective for the instances of large number of cities.

The Simulated Annealing can produce better solution than descent method if we properly set the control parameters. For some instances like *att48*, simulated annealing yielded the best solution in all construction algorithms. For other instances like *berlin52*, only if we set the appropriate control parameters, we can get a better solution than for the descent method. Also, in terms of average solution, Simulated Annealing did not have better performance than the descent method. In the cooling schedule experiment of Simulated Annealing, the solution can be affected by control parameters, such as the initial temperature, temperature decrement ratio, and stopping criterion. Hence, it is hard to say which condition results in the best schedule.

More experiments need to be executed by changing the control parameters, so that the effects of changes on the solutions can be observed.

Since no computational results on instances available in public domain were previously published in the literature for the studied problem, it was difficult to compare the proposed approaches to those by other authors. There are two other ways to evaluate the performance of the proposed heuristic algorithms. First is to compare the obtained solutions to lower bounds on optimal solution, such as the *Held and Karp lower bound* [17] for the single TSP. However, tight lower bounds for the minmax k-TSP are not easy to obtain and further investigation of this issue is required. Second, the heuristic solutions can be compared directly with the optimal solution. Since the minmax k-TSP is an NP-hard problem, in general, it is feasible to find the optimal solutions only for small-size instances of the problem. For instances with up to 20 vertices considered in this thesis, the heuristic approach produced the exact solution. However, CPLEX could not handle instances with more than 20 cities. To solve larger instances to optimality, cutting plane techniques and branch-and-cut algorithms could be used. Thus, a detailed polyhedral study of the problem of interest is an interesting direction in future research of the minmax k-TSP.

REFERENCES

[1] P.M. Franka, M. Gendreau, G. Laporte and F. Muller, “The m-traveling salesman problem with minmax objective,” *Transportation Science*, vol.29, 1995, pp. 267–275.

[2] B.L. Golden, G. Laporte and E.D. Taillard, “An adaptive memory heuristic for a class of vehicle routing problems with minmax objective,” *Computers and Operations Research*, vol.24, 1997, pp. 445–452.

[3] C.E. Miller, A.W. Tucker, and R.A. Zemlin, “Integer programming formulations and traveling salesman problems,” *Journal of the ACM*, vol.7, 1960, pp. 326–329.

[4] V. Bugera , “Properties of no-depot minmax 2-traveling salesmen problem,” in *Recent Developments in Cooperative control and Optimization*. S. Butenko, R. Murphey and P. Pardalos, Eds., Norwell, Massachusetts, Kluwer Academic Publishers, 2004, pp. 45–59.

[5] M. Bellmore and S. Hong, “Transformation of multisalesman problem to the standard traveling salesman problem,” *Journal of the ACM*, vol.21(3), 1974, pp. 500–504.

[6] G.N. Frederickson, M.S. Hecht and C.E. Kim, “Approximation algorithms for some routing problems,” *SIAM Journal on Computing*, vol.7, 1978, pp. 178–193.

[7] M.J. Spriggs, “Spanning forests and multi-traveling salesman tours with a minmax objective,” Master’s thesis, University of Saskatchewan, Saskatoon, Saskatchewan, March 2000.

[8] D. Sofge, A. Schultz and K. De Jong, "Evolutionary Computational Approaches to Solving the Multiple Traveling Salesman Problem Using a Neighborhood Attractor Schema," *Lecture Notes in Computer Science*, vol.2279, Springer 2002, pp. 153–162.

[9] S. Lin, and B.W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol.21, No.2, 1973, pp. 498–516.

[10] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol.13, 1986, pp. 533–549.

[11] E.H.L. Aarts, J.H.M. Korst and P.J.M. Laarhoven, "Simulated annealing", in *Local Search in Combinatorial Optimization*. E.H.L. Aarts and J.K. Lenstra, Eds., Chichester, UK: John Wiley & Sons, 1997, pp. 91–120.

[12] T.A. Feo and M.G.C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol.8, 1989, pp. 67–71.

[13] S. Kirkpatrick, C.D. Jr. Gerlatt, and M.P. Vecchi, "Optimization by simulated annealing," *Science*, vol.220, 1983, pp. 671–680.

[14] G.A. Croes, "A method for solving traveling salesman problems," *Operations Research*, vol.6, 1958, pp. 791–812.

[15] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*. New York, Wiley-Interscience, 1998, chap. 7, pp. 247–251.

[16] G.A.P. Kindervater and M.W.P. Savelsbergh, "Vehicle routing: handling edge exchanges," in *Local Search in Combinatorial Optimization*. E.H.L. Aarts and J.K. Lenstra, Eds., Chichester, UK: John Wiley & Sons, 1997, pp. 337–360.

[17] M. Held and R. Karp, "The traveling salesman problem and minimum spanning trees, Part II," *Mathematical Programming*, vol.6, 1971, pp. 62–88.

[18] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

APPENDIX A

Table XIII. Result of descent method

test instance	K	init method	running time(average)			solution		
			init	L.S.	total	init	L.S.	best
att48	4	Gr2City	0.358	0.216	0.574	17254	10298.4	9315
		Greedy	0.287	0.095	0.383	10216	9249.8	9103
		Kcenter	0.036	0.092	0.128	10524	9389.7	9318
		KMeans	0.022	0.100	0.122	10407	9140.3	9103
		RmvTwo	0.328	0.075	0.403	9321	9129.7	9103
		Rt1st	0.697	0.113	0.809	10416	9702.8	9521
	5	Gr2City	0.887	0.164	1.051	12810	8164.4	7766
		Greedy	0.274	0.072	0.345	8271	7689.5	7652
		Kcenter	0.030	0.117	0.147	9497	7738.3	7454
		KMeans	0.022	0.101	0.123	10407	7903.4	7621
		RmvTwo	0.336	0.184	0.520	9103	8127.3	7621
		Rt1st	0.606	0.105	0.711	8507	7690.3	7454
	6	Gr2City	0.245	0.155	0.400	10216	6769.7	6220
		RmvTwo	0.351	0.121	0.472	8820	6591.1	6220
		Greedy	0.250	0.073	0.323	7026	7026	7026
		Kcenter	0.025	0.298	0.323	8598	6510.3	6226
		KMeans	0.014	0.084	0.099	7452	6862.1	6456
		Rt1st	0.567	0.102	0.669	6999	6286.9	6226
berlin52	4	Gr2City	0.350	0.246	0.595	3369	2463	2359
		Greedy	0.352	0.141	0.492	2321	2311	2221
		Kcenter	0.075	0.260	0.334	3650	2302.5	2221
		KMeans	0.031	0.219	0.250	2500	2254.9	2161
		RmvTwo	0.372	0.138	0.509	2251	2204.3	2182
		Rt1st	0.904	0.224	1.128	2482	2299.6	2231
	5	Gr2City	0.320	0.216	0.536	2711	1913.9	1807
		Greedy	0.317	0.172	0.489	2229	1943.1	1910
		Kcenter	0.076	0.237	0.314	2828	1933.7	1815
		KMeans	0.041	0.141	0.181	2500	1944.2	1884
		RmvTwo	0.388	0.153	0.541	2197	1878.2	1814
		Rt1st	0.814	0.162	0.976	2048	1739.7	1713
	6	Gr2City	0.294	0.177	0.470	2478	1624.1	1547
		Greedy	0.327	0.133	0.459	1751	1585	1531

Table XIII. Continued

test instance	K	init method	running time(average)			solution		
			init	L.S.	total	init	L.S.	best
		Kcenter	0.047	0.467	0.514	2350	1616.7	1573
		KMeans	0.028	0.342	0.370	2207	1604.6	1569
		RmvTwo	0.397	0.450	0.847	2089	1643.1	1476
		Rt1st	0.744	0.347	1.091	2034	1644.1	1586
kroa100	4	Gr2City	1.934	0.581	2.515	6799	6225	5979
		Greedy	1.914	0.584	2.498	6578	6096.7	5955
		Kcenter	0.167	0.908	1.075	9133	6899.4	6475
		KMeans	0.108	0.995	1.103	7418	6377.4	6085
		RmvTwo	2.455	0.619	3.073	6835	6164.9	6042
		Rt1st	6.079	0.670	6.750	6617	6133.6	5990
	5	Gr2City	1.676	0.367	2.044	5442	5281.8	5231
		Greedy	1.619	0.513	2.131	5239	5044.3	4959
		Kcenter	0.136	0.831	0.967	6690	5064.4	4688
		KMeans	0.078	0.497	0.575	5558	5208.4	4947
		RmvTwo	2.467	0.734	3.201	6157	5051.7	4790
		Rt1st	4.998	0.463	5.461	5612	5025.9	4629
	6	Gr2City	1.506	0.572	2.078	4976	4383.1	4242
		Greedy	1.439	0.472	1.911	4675	4394.8	4268
		Kcenter	0.103	0.640	0.744	6009	4429.4	4200
		KMeans	0.063	0.284	0.347	4480	4234.6	4230
		RmvTwo	2.500	0.691	3.190	6062	4578.2	4158
		Rt1st	4.425	0.325	4.750	4631	4285.7	4235
bier127	4	Gr2City	4.156	1.454	5.611	45298	38584.3	37174
		Greedy	3.965	1.539	5.504	42797	35552.3	34677
		Kcenter	0.401	1.907	2.308	44570	36703.6	34081
		KMeans	0.246	1.703	1.949	39781	35217.7	33352
		RmvTwo	4.135	1.127	5.262	35173	32757.5	32423
		Rt1st	14.150	1.288	15.438	34597	33265.7	32712
	6	Gr2City	2.676	2.128	4.805	36305	24567.8	23244
		Greedy	4.754	1.558	6.312	29195	23667.3	23169
		Kcenter	1.052	3.366	4.417	62489	25343.9	23736
		KMeans	0.238	3.833	4.070	36441	24431.5	23244
		RmvTwo	3.853	3.153	7.006	31990	24602.2	22884
		Rt1st	9.638	0.570	10.208	24089	23071.7	22815
pr264	4	Gr2City	37.732	9.956	47.688	27558	23476.6	23189
		Greedy	47.655	11.699	59.354	20914	13705.5	13009
		Kcenter	1.113	7.026	8.139	16461	14792	14792

Table XIII. Continued

test instance	K	init method	running time(average) init	running time(average) L.S.	running time(average) total	solution init	solution L.S.	solution best
		KMeans	0.605	6.986	7.590	17550	15000	15000
		RmvTwo	22.378	3.266	25.644	12875	12705	12705
		Rt1st	128.898	7.308	136.206	14182	13693	13693
	6	Gr2City	26.533	8.530	35.063	23742	19522.2	18733
		Greedy	28.509	10.436	38.945	13758	10095.7	9295
		Kcenter	0.544	7.767	8.311	10579	9392.5	8568
		KMeans	0.316	6.371	6.686	10140	9131.6	8526
		RmvTwo	23.305	7.258	30.563	12531	9051.6	8739
		Rt1st	75.305	6.399	81.704	10663	9613.5	9208
rat575	4	Gr2City	501.522	35.656	537.178	2240	2034.8	2016
		Greedy	502.972	36.772	539.744	2140	2015.4	1971
		Kcenter	9.153	61.947	71.100	2448	1991.2	1918
		KMeans	4.750	47.206	51.957	2239	1924.2	1876
		RmvTwo	180.685	16.907	197.592	1890	1867.8	1865
		Rt1st	2,071.351	32.097	2,103.448	1889	1852.4	1849
	6	Gr2City	300.549	39.231	339.780	1586	1438.8	1407
		Greedy	286.350	36.768	323.118	1520	1393.4	1371
		Kcenter	5.242	41.425	46.667	1831	1428.5	1374
		KMeans	2.305	17.480	19.784	1286	1245.1	1232
		RmvTwo	175.215	49.759	224.973	1868	1436.8	1389
		Rt1st	1,027.612	16.386	1,043.998	1283	1251.4	1247

APPENDIX B

Table XIV. Result of simulated annealing - berlin52 (k=4, Lin-Kernighan)

Init Method	Initial T	Tratio	Time	Avg Soln	Best Soln	Overall Best
Gr2City	50	0.50	1.333	2,502.7	2,321	2,107
		0.60	1.083	2,455.5	2,288	
		0.70	0.972	2,442.6	2,274	
		0.80	1.619	2,441.3	2,210	
	100	0.50	1.384	2,438.6	2,288	
		0.60	1.064	2,416.4	2,321	
		0.70	0.916	2,415.9	2,321	
		0.80	1.202	2,472.6	2,321	
		0.90	0.995	2,375.0	2,107	
	1,000	0.50	1.097	2,445.0	2,189	
		0.60	1.108	2,446.4	2,221	
		0.70	1.073	2,494.8	2,285	
		0.80	0.930	2,510.5	2,429	
		0.90	0.941	2,477.4	2,321	
	3,000	0.50	0.925	2,479.7	2,299	
		0.60	1.008	2,521.2	2,321	
		0.70	1.450	2,530.6	2,281	
		0.80	1.172	2,588.2	2,375	
		0.90	1.398	2,485.3	2,362	
Greedy	50	0.50	0.855	2,362.2	2,321	2,137
		0.60	0.855	2,302.6	2,207	
		0.70	0.850	2,312.4	2,235	
		0.80	1.181	2,313.6	2,211	
	100	0.50	0.841	2,308.5	2,229	
		0.60	0.852	2,308.0	2,191	
		0.70	0.845	2,319.2	2,229	
		0.80	0.956	2,321.0	2,321	
		0.90	1.063	2,349.8	2,137	
	1,000	0.50	0.845	2,321.0	2,321	
		0.60	0.841	2,321.0	2,321	
		0.70	0.853	2,321.0	2,321	
		0.80	0.837	2,321.0	2,321	
		0.90	0.831	2,312.4	2,235	

Table XIV. Continued

Init Method	Initial T	Tratio	Time	Avg Soln	Best Soln	Overall Best
	3,000	0.50	0.947	2,321.0	2,321	
		0.60	0.962	2,321.0	2,321	
		0.70	0.900	2,321.0	2,321	
		0.80	0.992	2,321.0	2,321	
		0.90	0.844	2,321.0	2,321	
Kcenter	50	0.50	0.614	2,412.4	2,174	2,161
		0.60	0.636	2,466.8	2,161	
		0.70	0.800	2,363.9	2,235	
		0.80	0.630	2,382.8	2,202	
	100	0.50	0.755	2,412.4	2,163	
		0.60	0.648	2,463.2	2,228	
		0.70	0.606	2,354.8	2,161	
		0.80	3.675	2,320.3	2,161	
		0.90	0.872	2,297.5	2,181	
	1,000	0.50	0.616	2,311.5	2,186	
		0.60	0.616	2,295.2	2,186	
		0.70	1.134	2,403.5	2,304	
		0.80	0.606	2,360.9	2,321	
		0.90	0.809	2,351.1	2,161	
	3,000	0.50	0.623	2,353.8	2,271	
		0.60	1.211	2,431.1	2,215	
		0.70	0.700	2,315.7	2,161	
		0.80	0.694	2,318.9	2,161	
		0.90	0.628	2,371.0	2,186	
KMeans	50	0.50	0.553	2,302.4	2,235	2,126
		0.60	0.558	2,295.5	2,235	
		0.70	0.564	2,306.6	2,235	
		0.80	0.566	2,287.8	2,126	
	100	0.50	0.564	2,304.6	2,245	
		0.60	0.581	2,297.4	2,235	
		0.70	0.566	2,295.8	2,235	
		0.80	0.577	2,290.8	2,224	
		0.90	0.569	2,304.7	2,235	
	1,000	0.50	0.573	2,255.5	2,221	
		0.60	0.581	2,240.5	2,161	
		0.70	0.584	2,255.5	2,161	
		0.80	0.572	2,276.2	2,161	
		0.90	0.575	2,279.2	2,161	

Table XIV. Continued

Init Method	Initial T	Tratio	Time	Avg Soln	Best Soln	Overall Best
	3,000	0.50	0.669	2,256.5	2,161	
		0.60	0.611	2,256.7	2,161	
		0.70	0.594	2,289.5	2,235	
		0.80	0.791	2,275.3	2,161	
		0.90	0.563	2,281.1	2,163	
RmvTwo	50	0.50	0.884	2,310.4	2,182	2,118
		0.60	0.889	2,306.5	2,118	
		0.70	0.887	2,299.9	2,197	
		0.80	0.930	2,314.6	2,204	
	100	0.50	0.877	2,230.9	2,182	
		0.60	0.870	2,204.7	2,182	
		0.70	0.881	2,237.4	2,182	
		0.80	0.903	2,258.8	2,183	
		0.90	0.888	2,225.8	2,135	
	1,000	0.50	0.870	2,194.3	2,182	
		0.60	0.870	2,197.0	2,182	
		0.70	0.870	2,198.2	2,182	
		0.80	0.877	2,202.4	2,182	
		0.90	0.873	2,197.8	2,182	
	3,000	0.50	0.911	2,193.8	2,182	
		0.60	0.913	2,198.2	2,182	
		0.70	1.002	2,194.8	2,182	
		0.80	1.123	2,194.8	2,182	
		0.90	0.865	2,195.2	2,182	
Rt1st	50	0.50	1.434	2,344.6	2,211	2,118
		0.60	1.434	2,304.4	2,123	
		0.70	1.424	2,334.9	2,157	
		0.80	1.531	2,324.3	2,157	
	100	0.50	1.417	2,312.8	2,198	
		0.60	1.428	2,292.6	2,118	
		0.70	1.425	2,286.7	2,157	
		0.80	1.448	2,313.3	2,161	
		0.90	1.434	2,285.8	2,221	
	1,000	0.50	1.416	2,345.8	2,221	
		0.60	1.422	2,302.5	2,157	
		0.70	1.419	2,281.3	2,182	
		0.80	1.424	2,318.1	2,208	
		0.90	1.406	2,301.3	2,157	

Table XIV. Continued

Init Method	Initial T	Tratio	Time	Avg Soln	Best Soln	Overall Best
	3,000	0.50	1.536	2,346.8	2,251	
		0.60	1.550	2,296.2	2,157	
		0.70	1.608	2,286.0	2,198	
		0.80	1.586	2,302.3	2,157	
		0.90	1.416	2,326.3	2,251	

APPENDIX C

Table XV. Result of simulated annealing - att48 (k=4, Lin-Kernighan)

Init Method	Initial T	Tratio	Time	Avg Soln	Best Soln	Overall Best
Gr2City	50	0.7	1.130	9988.5	9103	9103
		0.8	0.969	10035.9	9103	
		0.9	1.356	9928.3	9103	
	100	0.8	1.125	10309.9	9355	
	300	0.8	1.706	10741.3	9103	
	1000	0.8	1.814	10429.2	9103	
	3000	0.5	1.439	10613.6	9103	
		0.6	1.805	10648.7	9798	
		0.7	3.894	11040.6	9515	
		0.8	1.278	10310	9555	
		0.9	1.995	10861	9827	
Greedy	50	0.7	0.614	9316.6	9103	9103
		0.8	0.614	9247.9	9103	
		0.9	0.623	9576.2	9103	
	100	0.8	0.613	9305.2	9103	
	300	0.8	0.659	9230.7	9103	
	1000	0.8	0.803	9267.7	9103	
	3000	0.5	0.799	9276.5	9103	
		0.6	0.745	9347.3	9176	
		0.7	0.753	9249.3	9103	
		0.8	0.773	9261.9	9103	
		0.9	0.642	9308.5	9103	
Kcenter	50	0.7	0.363	9384.6	9103	9103
		0.8	0.356	9414.5	9103	
		0.9	0.362	9489.9	9103	
	100	0.8	0.352	9466.4	9222	
	300	0.8	0.388	9502.6	9222	
	1000	0.8	0.436	9431.7	9385	

Table XV. Continued

Init Method	Initial T	Tratio	Time	Avg Soln	Best Soln	Overall Best
	3000	0.5	0.491	9428.7	9222	
		0.6	0.489	9378.6	9318	
		0.7	0.492	9415.4	9318	
		0.8	0.459	9407	9222	
		0.9	0.380	9432.8	9385	
KMeans	50	0.7	0.348	9352	9103	9103
		0.8	0.345	9294	9103	
		0.9	0.350	9331.7	9103	
	100	0.8	0.341	9188.8	9103	
	300	0.8	0.375	9240.4	9103	
	1000	0.8	0.434	9198.9	9103	
	3000	0.5	0.472	9219.1	9103	
		0.6	0.472	9144.7	9103	
		0.7	0.477	9199.3	9103	
		0.8	0.472	9158.5	9103	
		0.9	0.355	9261.9	9103	
RmvTwo	50	0.7	0.650	9222.5	9103	9103
		0.8	0.645	9180.4	9103	
		0.9	0.847	9418.5	9103	
	100	0.8	0.650	9162.3	9103	
	300	0.8	0.734	9120.8	9103	
	1000	0.8	0.903	9120.8	9103	
	3000	0.5	0.791	9160.3	9103	
		0.6	0.781	9155.3	9103	
		0.7	0.808	9142.5	9103	
		0.8	0.873	9156.4	9103	
		0.9	0.764	9120.8	9103	
Rt1st	50	0.7	1.049	9487.2	9103	9103
		0.8	1.278	9533.4	9103	
		0.9	1.058	9390.1	9103	
	100	0.8	1.049	9597.7	9103	
	300	0.8	1.131	9701.3	9481	
	1000	0.8	1.411	9732.5	9521	
	3000	0.5	1.197	9793.7	9521	
		0.6	1.186	9700.4	9521	

Table XV. Continued

Init Method	Initial T	Tratio	Time	Avg Soln	Best Soln	Overall Best
		0.7	1.189	9736.9	9521	
		0.8	1.345	9724.6	9521	
		0.9	1.078	9669.5	9521	

APPENDIX D

AMPL modeling

```
set I;
set K;

param C{I,I} >= 0;
param N>=0;
param KK>=0;

var X{I,I,K} binary;
var V{I} binary;
var U{I};
var Y;

minimize longest_dist: Y;

subject to constraint0 {k in K}:
sum{i in I, j in I} C[i,j]* X[i,j,k] <= Y;

subject to constraint1 {j in I}:
sum {k in K} sum {i in I} X[i,j,k] = 1;

subject to constraint2 {i in I}:
sum {k in K} sum {j in I} X[i,j,k] = 1;

subject to constraint3 {i in I, k in K}:
sum {j in I} X[i,j,k] = sum {j in I} X[j,i,k];

subject to constraint4  :
sum{ i in I} V[i] = KK-1;

subject to constraint5 {i in 1..N-1} :
2 <= U[i]+ V[i] ;

subject to constraint6 {i in 1..N-1} :
U[i] <= N ;

subject to constraint7 {i in 1..N-1, j in 1..N-1} :
U[i]-U[j]-N*(V[i]+V[j])+1<=(N-1)*(1-sum{k in K} X[i,j,k] );

subject to constraint8 {i in I, k in  K}:
X[i,i,k] = 0;
```

VITA

Name: Byungsoo Na

Address: Department of Industrial and Systems Engineering
Zachry Engineering Center, Texas A&M University
College Station, TX 77843-3131

Email Address: byungsoo.na@gmail.com

Education: B.S., Industrial Engineering,
Pohang University of Science and Technology, 2000
Pohang, Korea

M.S., Industrial Engineering,
Texas A&M University, 2006