國立交通大學

資訊管理研究所

碩士論文

訂單型之旅行家問題

Traveling Salesman Problem with Order Delivery

研究生：劉昱劭

指導教授：林妙聰 博士

中華民國九十七年六月

訂單型之旅行家問題

Traveling Salesman Problem with Order Delivery

研 究 生：劉昱劭　　Student: Yu-Shao Liu
指導教授：林妙聰　　Advisor: B.M.T. Lin

國立交通大學
資訊管理研究所
碩士論文

A Thesis
Submitted to Institute of Information Management
College of Management
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Business Administration
in
Information Management
June 2008
Hsinchu, Taiwan, the Republic of China

中華民國九十七年六月

誌謝

國立交通大學，一個在我高中時立志要唸的學校，終於，在碩士班的兩年把這個夢想完成了!以前的我並不愛唸書，直到唸了大學，我開始了解到讀書並不只是要對父母有所交待，而是要對自己負責，所以我用更認真的態度去面對「學習」這件事，這些過程並不輕鬆，所遇到的困難也不少，但幸運的是，我有一路扶持我的家人、老師與朋友。

首先要感謝的是我的爸媽，雖然家裡的經濟狀況不佳，但爸爸媽媽一直以來辛苦的工作為的只是讓我能衣食無缺的去上學；還要感謝我的姊姊，感謝他在我攻讀研究所時能一肩扛下養家的重擔，讓我無後顧之憂的完成學業；爺爺跟在天上的奶奶從小就對我疼愛有佳，如果沒有他們的照顧，我也無法完成這個目標。

接著要感謝的是我的指導教授林妙聰老師，他是我遇過最棒的老師，不僅讓我從對學術研究的一無所知到現在對它充滿了興趣，在做人處事方面也是非常值得我學習的對象；亞梅學姐一直以來都對我非常的照顧，能有這樣一篇通順完整的論文產生，都要歸功於她耐心的替我修改校訂；筱嵐學姐在課業及生活上給予我相當多的幫助，平時的鼓勵也是我在研究上面很大的動力；彥庭、延聰跟怡君，跟他們三個雖然只有短暫的相處，但卻是我碩班兩年中最好的朋友，是他們讓我覺得 lab 也有家的感覺；癸棠是我進研究所最早認識的人，個性剛好跟我互補，兩年來給了我很大的幫助；欣穎雖然跟我很早就認識了，但是這些日子以來才有了更多相處的機會，她心直口快的個性也為大家帶來了許多笑果；盈佑則是最乖巧稱職的學妹，隨和的好脾氣跟我很合得來；510 是個超級熱心的好學姐，雖然大我兩歲，可是相處起來就像朋友般輕鬆。最後是一路陪伴著我的巧玲，擁有這樣溫柔體貼又有趣的女朋友，讓我就算是低潮時也能勇往直前，走向未來。

Title of Thesis: Traveling Salesman Problem with Order Delivery

Name of Institute: Institute of Information Management

Student Name: Yu-Shao Liu Advisor Name: Professor B.M.T. Lin

Abstract

This thesis considers the traveling salesman problem incorporating order delivery. There is a set of nodes to be visited and each node belongs to a specific group. The completion time of a group is the moment when all nodes belonging to it have been visited. The problem is to determine a visitation sequence of the nodes such that the weighted sum of completion times over all groups is minimized. We present a binary linear programming model to formulate the studied problem and then develop an $O(n^2 2^n)$ dynamic programming algorithm for determining optimal solutions. To produce approximate solutions within an acceptable time, we design a tabu search algorithm, an iterated local search algorithm and a genetic algorithm. Computational experiments are conducted to study the performance of the proposed model and algorithms. Numerical statistics suggest that the binary linear program can reach optimal solutions faster than the existing model, and the iterated local search algorithm outperforms other approaches when the number of nodes increases.

Keywords: Traveling salesman problem, order delivery, weighted completion time, dynamic programming, approximate solution

摘要

在現今的商業行為模式下，許多服務內容都是以訂單做為計算單位，而以業主的角度來說，更關心的可能是訂單的交付時間，不是傳統的個別工作完成時間。因此，本論文考慮一個特殊型態的旅行家問題，捨棄傳統問題的目標式，而改由訂單交付時間為主要考量。在問題中，每個必須拜訪的節點隸屬於某一訂單，而每筆訂單的完成時間定義為此訂單所有節點都被拜訪完的時間點，其目的是要找出一組拜訪順序，使得所有訂單的加權平均完成時間最小。針對這個問題，我們提出了一個 0/1 線性數學式，並設計一個複雜度為 $O(n^2 2^n)$ 的動態規劃產生最佳解。此外，為了能夠在可接受的時間內產生合理的近似解，我們設計了禁忌搜尋法、迭代區域搜尋法以及基因演算法等近似演算法。為驗證本論文所提之相關演算法，我們設計一系列實驗。實驗結果顯示，迭代區域搜尋法在節點數較多之時的效能表現優於其他兩者。

關鍵字：旅行家問題，訂單交付，加權完工時間，動態規劃，近似解

Table of Contents

Chapter 1 Introduction

This thesis considers a variant of the traveling salesman problem by incorporating order delivery into the problem setting. In classical scheduling problems, the identities in the objective functions are jobs, which may consist of several tasks or operations. The concept of order delivery lies in the fact that many applications define the objective functions in terms of higher-level entities through aggregation. In the scheduling problem addressed in this thesis, jobs belonging to the same order or group will be delivered as a single batch, and the objective function considers order completion times instead of job completion times. The completion of an order is determined by the completion of the last job of that order. A set of jobs are grouped, in a disjoint sense, as different orders in a single machine scheduling environment. There exists a sequence-dependent setup time between any two consecutive jobs, and job processing times are negligible. The objective is to determine a job processing sequence so as to minimize the weighted sum of completion times over all groups.

The problem setting originates from a simplified model of the satellite imaging problem, although applications in many other areas are possible. Several requests/orders of various numbers of photographs need to be fulfilled via satellite imaging operations. Each order may consist of several photos, and is associated with a weight that characterizes the importance or the degree of urgency. The satellite should adjust the imaging angle of the camera for the next photo shooting upon the completion of the current photo shooting. Such alternation of camera lens requires great accuracy with technical concerns, and takes a much longer processing time compared with the photo shooting time. Therefore, various setup times are indispensable between any two consecutive imaging operations, while the shooting time may be ignored. For example, the Ministries of Defense, Agriculture, and Energy may place orders which demand various numbers of photos. The objective is to accomplish all requests according to a photo shooting schedule which minimizes the weighted sum of completion times over

all orders.

This model may be adopted by some other applications under different scenarios, such as in vehicle routing problems (VRP). The customers to be served may actually belong to different companies, and the service provided to a company is fulfilled only when all the subsidiary sites of that company are served. In addition, such concept involving change-over setup costs with batch delivery may also be applied as a variation of the traveling salesman problem. If the constraint of the order delivery is removed, the studied problem is equivalent to the classical deliveryman problem or the minimum latency problem, which determines the visitation sequence of nodes with the objective of minimizing the sum of completion times of nodes on the network. Nevertheless, the incorporation of order delivery is essential to certain practical applications. For example, an order may make sense to a customer only when all of the components are accomplished; or some orders may have much higher degree of urgency than the rest of the orders do. Some other transportation problems, such as VRP, may also incorporate with the concept of order delivery in order to capture and interpret complicated down-to-earth problems in real life.

In this thesis, we denote the studied by the TSP-WOCT (Weighted Order Completion Times). The TSP-WOCT conjoins the classical deliveryman problem with batch delivery scheduling on single machine to compose a new model. The deliveryman problem is similar to the well-known traveling salesperson problem with differences in the objective functions. The total delay over all nodes is deliberated in the deliveryman problems while the TSP considers only the total length. The TSP is studied from an aspect of internal efficiency and the deliveryman problem focused instead on customer satisfaction. It is interesting that deliveryman problem has been studied under different titles, such as “the traveling repairman problem” (Afrati et al., 1986; Garcia et al., 2002) and “deliveryman problem” (Minieka, 1989; Fischetti et al., 1993). Some recent researches about the minimum latency problems actually cope with the same problems (Blum et al., 1994; Wu 2000). Such minimum latency problems are also proven to be NP-hard, and the polynomial time algorithms can only be applied for some specific graphs, such as paths (Afrati et al., 1986, Garcia et al., 2002), edge-unweighted trees

(Minieka, 1989), tree of diameter 3 (Blum et al., 1994). Wu et al. (2004) proposed an exact algorithm by applying a dynamic programming algorithm along with branch and bound technique for the small-scale problems. In addition, some researches developed approximation algorithms for the minimum latency problems (Archer and Williamson, 2003; Goemans and Kleinberg, 1998).

In single machine scheduling with batch delivery, a set of jobs are to be scheduled and the jobs may be grouped as batches, and a batch contains contiguously scheduled jobs. All jobs in the same batch are delivered to the customer as a whole upon the completion of the last job in the batch. Cheng and Kahbacher (1993) first considered the problem with the objective to minimize the sum of the total weighted batch delivery time. They also proposed another objective of minimizing the total weighted earliness and a batch delivery penalty depending on the number of batches. The general version of the proposed problem was demonstrated to be ordinary NP-hard in the same article. For the same problem, Cheng et al. (1996) later proved that it is strongly NP-hard. They also proposed polynomial algorithms for special cases with equal processing times or equally weighted batches. Although a dynamic programming algorithm was applied by Cheng and Gordon (1994) to solve the general version of the single machine with batch delivery scheduling problem, Cheng et al. (1996) further clarified that this problem can be formulated as a classical parallel machine scheduling problem. Therefore, the exact/approximate algorithms and complexity analyses for the corresponding parallel machine scheduling problem can be easily extended to the problem. Cheng et al. (1997) proposed another objective function to the batch delivery scheduling problem by minimizing the total weighted earliness and mean batch delivery time. They proved the strong NP-hardness of the problem and provided polynomial algorithms for some special cases. Yang (2000) studied the single machine scheduling problems with generalized batch delivery dates and earliness penalties, which is proven to be strongly NP-hard. He also suggested a polynomial time solvable case, even for general earliness penalty function, when all processing times are equal. An exact algorithm for such case, while weighted earliness functions being considered, was provided by Yang in the same

article. Recently, Ji et al. (2007) considered a scheduling problem with batch delivery property, that is, jobs are delivered in batches and the delivery date of a batch determined by the completion time of the last job in the batch. The objective is to minimize the sum of the total weighted flow time and delivery cost. They proved the problem to be strongly NP-hard and applied a dynamic programming algorithm that runs in pseudo-polynomial time for the cases with bounded numbers of batches. They also proposed optimal algorithms for two special cases.

The rest of this thesis is organized as follows. In Chapter 2, we formally define the studied scheduling problem. A binary integer programming model will be presented to describe the problem. In Chapter 3, an exact algorithm is developed by the dynamic programming approach. A special case is also studied. In Chapter 4, we design a tabu search algorithm, an iterated local search algorithm and a genetic algorithm to produce approximate solutions within an acceptable period of time. In Chapter 5, a series of computational experiments are conducted to scrutinize the performance of the proposed algorithms. Finally, some conclusions and suggestions for further research are addressed in Chapter 6.

Chapter 2 Problem Formulation and Mathematical Model

In this chapter, we present formal statements of the scheduling problem. An binary integer programming model is also proposed to given a mathematical formulation.

Formerly, the TSP_WOCT can be described from the perspectives of scheduling theory as follows. There are K orders $\theta = \{O_1, O_2 \ldots, O_K\}$ to process on a single-machine. Each order O_k, $1 \le k \le K$ consists of n_k jobs and has a non-negative weight w_k. Let $n_1 + n_1 + \cdots + n_K = n$ and $N = \bigcup_{k=1}^{K} O_k = \{J_1, J_2, ..., J_n\}$ denote the set of all jobs from the K orders. The processing time of any job of N is negligible. However, there is a non-negative sequence-dependent setup time d_{ij} if job J_j immediately follows job J_i, regardless of which orders they belong to. An initial job J_0 is given to specify the machine status for defining the setup required by the job scheduled first. Given a processing sequence of all jobs, the completion time C_k of order O_k is the moment when its last job is finished. The objective is to minimize the weighted completion times $\sum_{k=1}^{K} w_k O_k$ over all orders.

Example: There are five jobs J_1, J_2, J_3, J_4, J_5 belonging to two orders O_1 and O_2 such that $O_1 = \{J_1, J_2, J_3\}$ and $O_2 = \{J_4, J_5\}$. Order weights are $w_1 = 8$ and $w_2 = 5$. The sequence-dependent setups are given in the following table.

d_{ij}	J_1	J_2	J_3	J_4	J_5
J_0	3	5	2	3	4
J_1	0	3	8	6	2
J_2	10	0	4	2	12
J_3	7	6	0	8	5
J_4	9	4	7	0	7
J_5	13	5	11	4	0

Consider the processing sequence $S = J_2J_1J_4J_3J_5$. The completion times of the jobs are 5, 15, 21, 28 and 33, respectively. The completion times of order O_1 and order O_2 are subsequently $C_1 = 28$ and $C_2 = 33$, respectively. Thus, the weighted completion time is 8×28+5×33=389.

In Fischetti *et al*. (1993), an integer programming formulation was proposed for the latency TSP. In their model, integer variable x_{ij} is used to indicate the status of the arc from city i to city j. If $x_{ij} = 0$ then arc (i, j) is not used. If $x_{ij} = n-k+1$, then it is in the k-th position on the Hamiltonian tour. There are $O(n^2)$ integer variables and $O(n^2)$ constraints. The model is shown below.

IP_Latency_TSP

Minimize $\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}$

Subject to

$$\sum_{j=1}^{n} y_{ij} = 1, \qquad 1 \le i \le n; \tag{1}$$

$$\sum_{i=1}^{n} y_{ij} = 1, \qquad 1 \le j \le n; \tag{2}$$

$$y_{ij} \in \{0,1\}, \qquad 1 \le i, j \le n; \tag{3}$$

$$\sum_{i=1}^{n} x_{i1} = 1, \tag{4}$$

$$\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj} = \begin{cases} 1-n & (k=1) \\ 1 & (2 \le k \le n) \end{cases} \tag{5}$$

$$x_{ij} \ge 0, \qquad 1 \le i, j \le n; \tag{6}$$

$$x_{ij} \le r_{ij}y_{ij}, \qquad 1 \le i, j \le n; \tag{7}$$

Where $r_{ij} = \begin{cases} 1, & \text{if } j = 1; \\ n, & \text{if } i = 1; \\ n-1, & \text{otherwise.} \end{cases}$

The program IP_Latency_TSP has been widely adopted in the literature for related researches on TSP-related problems because it provides theoretical structures that facilitate the development of approximation algorithms and lower bounds. In this thesis, we consider another alternative for formulating the latency problem because knowing the position of an arc (*i*, *j*) in a Hamiltonian cycle does not provide sufficient information to calculate the contribution arc (*i*, *j*) makes to the objective function.

In the new formulation, we use binary variable x_{ijl}, $1 \le i \ne j \le n, 2 \le l \le n$, to indicate whether arc (*i*, *j*) from job J_i to J_j is the *l*-th edge in the TSP sequence and variable u_{0j}, $1 \le j \le n$, indicates whether job J_j is scheduled first or not. Let auxiliary variable t_j denote the completion time of job J_j.

BIP_Latency_TSP

Minimize $Z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}[x_{ijl} \times (n-l+1) \times c_{ij}]$

Subject to

$$\sum_{j=1}^{n} x_{0j1} = 1, \tag{8}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijl} = 1, \qquad 1 \le l \le n; \tag{9}$$

$$\sum_{j=1}^{n}\sum_{l=1}^{n} x_{ijl} = 1, \qquad 1 \le i \le n; \tag{10}$$

$$\sum_{i=1}^{n}\sum_{l=1}^{n} x_{ijl} = 1, \qquad 1 \le j \le n; \tag{11}$$

$$\sum_{i=1}^{n} x_{ijl} = \sum_{i=1}^{n} x_{ji(l+1)}, \qquad 1 \le l \le n-1, \ 1 \le j \le n; \tag{12}$$

$$x_{ijl} \in \{0,1\}, \qquad 1 \le i, j, l \le n; \tag{13}$$

The binary integer program **BIP_Latency_TSP** (8-13) does not explicitly embed the contribution of each arc in the decision variable x_{ijl}. Therefore, the actual contribution is implicitly calculated in the objective function by multiplying $n-j+1$. Constraints (12) indicate each job J_j is the start node of the l-th arc if and only if it is the end node of the (l+1)th arc. No arc and no job will be ignored by constraints (8-11). In our formulation, $O(n^3)$ decision variables and $O(n^2)$ constraints are involved.

With the new binary integer program, we can accordingly design a binary integer program in the following. Auxiliary variables t_j are used to specify the completion time of job J_j and C_k the completion time of order O_k.

Problem TSP-WOCT

Minimize $Z = \sum_{k=1}^{K} w_k C_k$

Subject to

$$\sum_{i=0}^{n}\sum_{j=0}^{n} x_{ijl} = 1, \qquad 0 \le l \le n; \tag{14}$$

$$\sum_{l=0}^{n} x_{ijl} = 1, \qquad 0 \le i \ne j \le n; \tag{15}$$

$$\sum_{i=1}^{n} x_{ijl} = \sum_{i=1}^{n} x_{ji(l+1)}, \qquad 1 \le l \le n-2, \ 1 \le j \le n; \tag{16}$$

$$\sum_{j=0}^{n}\sum_{l=0}^{n} x_{ijl} = 1, \qquad 0 \le i \le n; \tag{17}$$

$$\sum_{i=0}^{n}\sum_{l=0}^{n} x_{ijl} = 1, \qquad 0 \le j \le n; \tag{18}$$

$$\sum_{j=1}^{n} x_{0j0} = 1, \tag{19}$$

$$\sum_{i=1}^{n} x_{i0n} = 1, \tag{20}$$

$$t_j + (1 - \sum_{i=0}^{n} x_{ijl})M \geq \sum_{u=1}^{n}\sum_{v=1}^{n}\sum_{r=1}^{l} x_{uvr} d_{uv}, \qquad 0 \leq l \leq n-1,\ \ 1 \leq j \leq n; \tag{21}$$

$$C_k \geq t_j, \qquad J_j \in O_k; \tag{22}$$

$$t_j \geq 0, \qquad 1 \leq j \leq n; \tag{23}$$

$$C_k \geq 0, \qquad 1 \leq k \leq K; \tag{24}$$

$$x_{ijl} \in \{0,1\}, \qquad 1 \leq i, j \leq n, 1 \leq l \leq n-1; \tag{25}$$

In the above formulation, constraint set (14) specifies that each position can accommodate exactly one edge, and constraint sets (15) on the other hand restricts each edge to be assigned to at most one position. In constraints (16), if node J_j is the end point of the l-th edge of the schedule, then it must be the start node of the (l+1)-th edge. Constraints (17) and (18) are given to ensure each node will be the start node of some arc and the end node of anther arc exactly once. Constraints (19) and (20) are defined the initial state and the final state. Constraint set (21) confines the completion time of job J_j. If it is scheduled as the end node of the l-th edge, then its completion time is no less than the sum of travel distance passing through its predecessors. Constrain set (22) defines the completion times of all orders. The remaining constraints dictate the characteristics of the decision and auxiliary variables. This program also uses $O(n^3)$ decision variables and $O(n^2)$ constraints.

Chapter 3 Dynamic Programming and Special Case

In this chapter, we discuss a special case that can be solved in polynomial time. For the general NP-hard problem, a dynamic programming algorithm will be developed for producing optimal solutions.

The TSP-WOCT is similar to the deliveryman problem and the minimum latency problem, which are known to be NP-hard. It is very likely to require a long computing time for producing optimal solutions. Still, we seek to find a systematic way to the generation of optimal schedules. In this thesis, we adopt the dynamic programming approach, which is one of the most widely adopted implicit enumerative methods. Following the standard design technique, Wu (2004) designed a dynamic programming algorithm for the minimum latency problem. However, the completion times of interest in the TSP-WOCT is calculated over orders. As the two problems demonstrate many similar features, we adapt the existing dynamic programming algorithm to develop an exact solution method for the TSP-WOCT. The dynamic programming algorithm is described in the following. Notation used in the algorithm is defined first.

Notation

- $N' \subseteq N$: a subset of scheduled jobs;
- $\prod_{N',J_j}$: the set of all sequences of jobs of N' with job $J_j \in N'$ scheduled last;
- $S(N', J_j)$: a particular sequence in $\prod_{N',J_j}$;
- $L(S(N', J_j))$: length of sequence $S(N', J_j)$.
- $\theta(N')$: the subset of orders that are already completed in N' ;
- $C_k(S(N', J_j))$: the completion time of order O_k in $S(N', J_j)$ for $O_k \in \theta(N')$;
- For each partial sequence $S(N', J_j)$, define its contribution to the objective function by

$$W(S(N^{'},J_j)) = \sum_{O_k \in \theta(N^{'})} w_k C_k(S(N^{'},J_j)) + L(S(N^{'},J_j)) \times \sum_{O_k \in \theta \setminus \theta(N^{'})} w_k \ .$$

The first part is the cost already incurred, and the second part is due to the cost that the partial sequence will make to the unfinished orders.

$S_1 = J_2 J_4 J_3 J_1$	$S_1^{'} = J_2 J_4 J_3 J_1 J_5$	$W(S_1^{'}) = W(S_1) + d_{15} \times \sum_{O_k \in \theta \setminus \theta(N^{'})} w_k$
$S_2 = J_4 J_3 J_2 J_1$	$S_2^{'} = J_4 J_3 J_2 J_1 J_5$	$W(S_2^{'}) = W(S_2) + d_{15} \times \sum_{O_k \in \theta \setminus \theta(N^{'})} w_k$

As the table shows, if we know that $W(S_1) \leqq W(S_2)$, we can also confirm that $W(S_1^{'}) \leqq W(S_2^{'})$. So we can say that S_2 can be eliminated by S_1. It is because when we add the job to the partial sequence every recursion. The extra contribution to the objective function of the new jobs only depends on the last job of original partial sequence. For S_1 and S_2, we say that they have the same configuration ($N^{'}$, J_1). For each possible configuration, we only need to keep one partial sequence. For example, for $N^{'} = \{J_1, J_2, J_3, J_4, J_5\}$, it has five configurations $S(N^{'},1), S(N^{'},2), S(N^{'},3), S(N^{'},4), S(N^{'},5)$.

With the above notation, we can then design a dynamic programming algorithm. For each subset $N^{'}$ and job $J_j \in N^{'}$, define function

$$f\left(N^{'},J_j\right) = \min_{S(N^{'},J_j) \in \prod_{N^{'},J_j}} \left\{W\left(S(N^{'},J_j)\right)\right\}.$$

The value of $f\left(N^{'},J_j\right)$ can be computed by a recursive formulation if we know the values $f\left(N^{'} \setminus \{J_j\}, J_i\right)$ for all jobs $J_i \in N^{'} \setminus \{J_j\}$. The dynamic programming algorithm and figure is given as follows:

DP_WOCT:

Initial conditions:

$$f\left(N^{'},J_{j}\right)=\begin{cases}0, \text{ if } N^{'}=\left\{J_{0}\right\} \text{ and } j=0;\\ \infty, \text{ otherwise.}\end{cases}$$

Recursion:

$$f\left(N^{'},J_{j}\right)=\min_{J_{i}\in N^{'}\setminus\{J_{j}\}}\left\{f\left(N^{'}\setminus\left\{J_{j}\right\},J_{i}\right)+d_{ij}\times\sum_{O_{k}\in\theta\setminus\theta(N^{'})}w_{k}\right\}.$$

Goal:

Determine $\min_{J_{j}\in N}\left\{f\left(N,J_{j}\right)\right\}$.

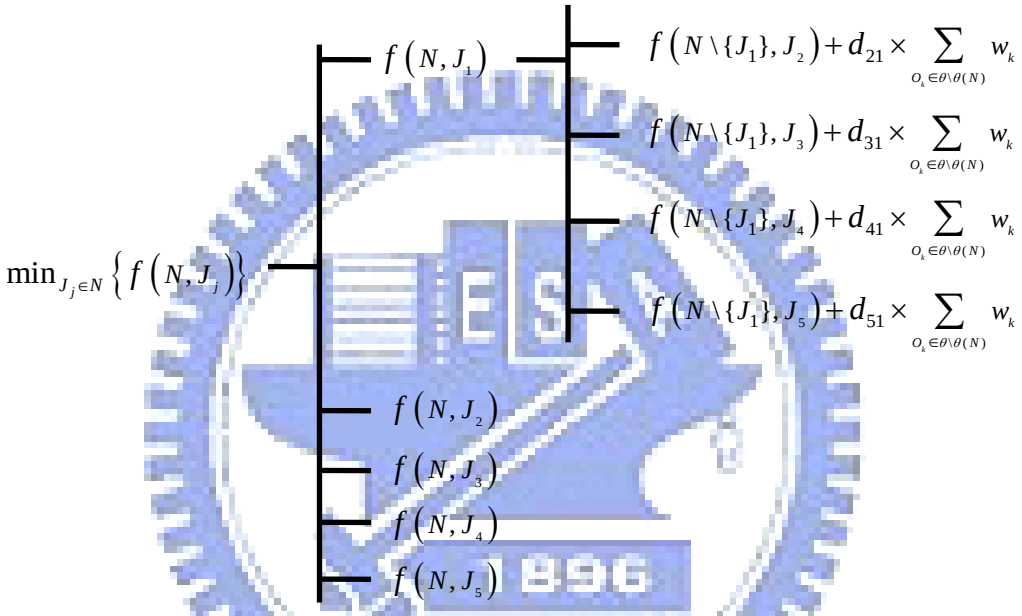

Figure 1: Recursion of the DP

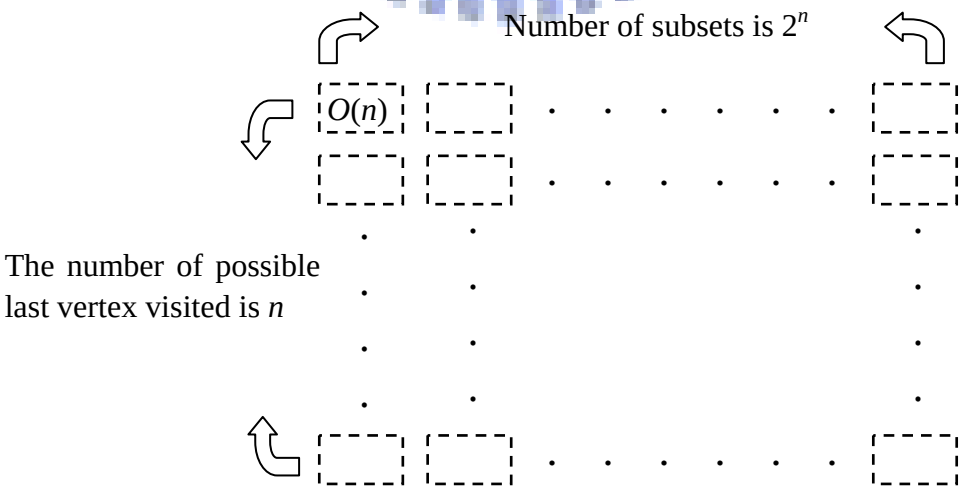

Figure 2: Complexity of the DP

Regardless of whether job J_i finishes its order when it is appended to the sequence of $f(N'\backslash\{J_j\},J_i)$, an extra cost of $d_{ij} \times \sum_{O_k \in \theta\backslash\theta(N')} w_k$ is introduced to the remaining unfinished orders. With the initial condition $f(\{J_0\},J_0) = 0$, we want to find $\min_{J_j \in N} \left\{ f\left(N, J_j\right) \right\}$. As the figure2 shows that there are $O(n2^n)$ entries, each of which can be computed in $O(n)$ time if $\sum_{O_k \in \theta\backslash\theta(N')} w_k$ for each $N' \subseteq N$ is known in advance. Therefore, the overall computing time is $O(n^2 2^n)$.

While the general version of the studied problem is NP-hard, we consider a restricted case where the setup time for each job J_j, $1 \le j \le n$, is independent of its predecessor, i.e. $d_{ij} = d_j$. Denote this case by TSP-WOCT$_{d_{ij}=d_j}$. In this case, we can treat the setup time for each job as its processing time as in classical scheduling theory.

Lemma 1: In an optimal schedule of TSP-WOCT$_{d_{ij}=d_j}$, except for the job processed first, jobs belonging to the same order are scheduled consecutively.

Proof: The validity can be established by the standard job-interchange argument.

Let $P_k = \sum_{J_j \in O_k} d_j$, the total processing length of order O_k. By assigning a specific job J_j to the first position, all the remaining jobs of $N - \{J_j\}$ can be scheduled order by order. Therefore, each order can be considered as a composite job and the following property follows.

Lemma 2: There is an optimal schedule of TSP-WOCT$_{d_{ij}=d_j}$ that schedules the orders by the WSPT (weighted shortest processing time) rule, i.e. the orders are sequenced by non-increasing $\frac{w_k}{P_k}$.

Theorem 1: The TSP-WOCT $d_{ij} = d_j$ problem can be solved in $O(n \log n)$ time.

PROOF: The time complexity is dominated by the sorting stage, which takes $O(n \log n)$ time.

Chapter 4 Approximate Solutions

As the TSP-WOCT is NP-hard, when the problem size increases, the required run time will grow exceedingly fast for producing optimal solutions. Although a dynamic programming algorithm was developed in the previous chapter, it is impractical for implementation because the required memory and computing time are not affordable when the problem size is large. In this chapter we consider several approximation methods that can produce quality solutions in a reasonable time. The approximation algorithms include local search, tabu search, genetic algorithm and iterated local search.

4.1 Local Search

We design the first approximation method **H1-LS** based on local search with **2-Opt**. The original concept is to first randomly construct an initial solution S_0, followed by swapping two edges in every iteration in order to reach a better solution. If the new solution S_1 has a better objective value, the local search approach retains this solution S_1, and the same process will proceed to the next solution S_2 based on solution S_1. If solution S_1 does not have an objective value better than S_0, this solution is discarded. The process iterates until no more improvement through swapping is possible. In order to get better solutions to the problem, we modify some steps as follows. We construct an initial solution by a greedy method; that is, the algorithm selects some node as the first one, and successively determines the next node, from among the unscheduled jobs, with the minimum setup time until all nodes are scheduled. Having the initial solution, the algorithm starts the local search phase. The search process will terminate after all feasible pairs of edges have been examined. When the local search stops, the algorithm decides another initial job and constructs another initial solution again. Following that, the process begins local search by 2-Opt again until all feasible initial solutions have been examined.

Algorithm H1-LS:

Set incumbent value $z = \infty$.

For j = 1 to n do the following three steps

Step1: Designate job J_j to be scheduled first.

Step2: Construct initial solution S_0 by the greedy method. Set $z = Z(S_0)$.

Step3: For each feasible pair of edges in S_0

3.1: Swap the two edges and generate new solution S_1.

3.2: If $Z(S_1) < z$ then $S_0 = S_1$ and $z = Z(S_1)$.

The second heuristic method **H2-LS** is similar to H1-LS. It is also based on local search, but utilizes 2-Exchange as the swapping strategy. This approach swaps the sequence of two nodes in each improvement iteration. Except this alteration, other procedure details are identical to H1. Both 2-Opt and 2-Exchange have been commonly applied to the TSP. We design heuristic H1-LS and H2-LS not only to study their effectiveness but will also deploy the two approaches in the design of tabu search to define the neighborhood structures.

4.2 Tabu Search

Tabu Search approach was first proposed by Glover (1989) and has been used to solve numerous combinatorial optimization problems. The principle of tabu search is to impose some restrictions so as to guide a search process to cover a wide domain of possible solutions and avoid being trapped by local optimum. In the search process, it usually starts from a selected initial solution S_0 and generate a set of neighbors $\{S'\}$ by a heuristic swapping. The objective value is evaluated for each S' and the best neighbor becomes a new current solution. The next iteration will be triggered based on the current solution. These procedures are repeated until certain stopping conditions are satisfied. Moreover, a “tabu list”, the essence in tabu search, is generated in order to avoid cycling in the search process. A tabu list contains solutions that have been considered as a “current solution” in the recent past. The size of a tabu list is defined according to some experience rules. For different designs for different problems,

specific details need to be specified. The following is the components of our tabu search algorithm.

Initial solution

The initial solution is constructed by a greedy method, same as that used by H1-LS and H2-LS.

Neighborhood structure

In this thesis, we respectively adopt two types of swaps, 2-Opt and 2-Exchange. We can generate a set of neighbors by the swapping operations. However, even though the probability of finding the global optimum is larger as the number of neighbors increases, the run time nevertheless also increases. Therefore, we determine the number of neighbors by computational experience to compromise between the search quality and the execution time.

Select the best neighborhood solution and the tabu list

In the search process**,** a set of neighbors is generated for iteration**.** First, we compute the objective function for each neighbor and record it in the memory. Second, we sort these objective values in non-decreasing order. Then we can find the best neighborhood solution easily. Moreover, we should check the tabu list in order to bypass local optima. If the best neighborhood solution is already in the tabu list, we will choose the next one. Finally, the selected neighborhood solution should be recorded in the tabu list.

The size of the tabu list can be fixed or variable. In general, the size is set to be *seven* by experience and we adopt the size in our implementation. When the tabu list is full and a new solution is encountered, the earliest solution recorded in the tabu list will be deleted and the next new solution will be appended to the list. Therefore, the tabu list is maintained by a queue with the first-in-first-out mechanism.

Aspiration criterion

The aspiration criterion is important for helping a tabu search method probe the solution space to locate better solutions. It is used to determine when the tabu restriction can be overridden. Our search procedure utilizes the standard form of aspiration criterion, called “global aspiration”, in which the search process overrides the restriction of the tabu list when a new solution encountered is better than the current one.

Local search

Since the tabu search approach uses a random search method, its performance may exhibit fluctuation. Therefore, we add a deterministic method to keep its performance more consistent. When it reaches a solution that is better than the current best solution, the local search method is invoked. This step can guarantee the solution we find by tabu search is the best solution in its local neighborhood area.

Stopping criterion

We adopt a simple stopping criterion. The search process is terminated after a certain number of iterations. The number of iterations is determined by preliminary experiments. We will introduce the details in the next chapter.

4.3 Genetic Algorithm

In the early 1970s, John Holland introduced the concept of genetic algorithms borrowing the principle of evolution from the nature. The result of such a simulation is a series of optimization algorithms, usually based on a simple set of rules. Optimization iteratively improves the quality of solutions until an optimal, or at least feasible, solution is found. Some merits of GA are that it can be implementing easily, search the solution globally, and adapt to the changing conditions in the problem.

Despite of these advantages, since GA does not use unequivocal rules of how to search for the solutions, it is often slower than conventional methods such as heuristic

methods or local search methods. For this reason, we can adapt hybrid methods that combine GA with other conventional techniques. In this thesis, we adapt a method that combines GA and Local Search method [9]. The Local Search method used to be the mutation operator of GA; but the local search methods easily let the solutions falls into local optimum. To avoid the situation occupied, the crossover operator provides the capability of jumping out form the local optimum. The algorithm consists of the following steps.

Initialization

Generate a population of chromosomes and calculate the fitness of each chromosome. The size of population is denoted by M.

Natural Selection

Set the crossover probability P_c. Within each generation, $M*(1 - P_c\%)$ chromosomes are selected to stay in the new population without crossover. Chromosome selection is based on their fitness values. The others that are not selected are going to produce the offspring which can be added to the new population.

Reproduction

Choose $M*P_c\%$ chromosomes randomly and produce an offspring from each pair of individuals.

Mutation by local search

Set the mutation probability P_m. Then choose $M*P_m\%$ chromosomes randomly and improve them by 2-Opt. The individuals that stay in the new population by natural selection are always chosen. This can help the search process find the optimum more quickly.

The followings describe the detail of each of the above steps.

Initialization

Chromosome coding and representation are the most crucial to the design of genetic algorithms. Clear represent the problem solution in chromosomes and easy-to-compute definition of fitness functions are not trivial to achieve. Because the studied problem is a variant of the TSP, we adapt the sequence representation to encode the chromosome. Examples are given in the following.

1	3	2	4	6	5	7

2	4	3	1	7	5	6

After the chromosome coding, we need to generate the initial population:

Initial population :

For greedy method

- Pick up the initial nodes of each solution randomly
- Construct an initial solution by greedy fashion
- Generate M chromosomes

To avoid the solution fall into local optimum too early, we generate the half of initial population randomly.

Natural Selection

Each generation we should pick up M*(1-P_c%) chromosomes to stay in the new population without crossover. The chance of survival is defined by the fitness value. It can avoid dropping the chromosome which is better than other ones. We first sort the individuals in fitness value order and then compare the fitness values and pick up the first M*(1-P_c%) chromosomes. Each selected chromosome is improved by the mutation operator.

Crossover in Reproduction

When we apply the local search to a solution, it often falls into a local optimum. It this needs some method to jump out from the local optimum. Let's consider the two chromosomes which have fallen into local optimum. Each of

them may have some best genes for different parts of the chromosome. It is possible to get a better chromosome if we combine the two chromosomes which have fallen into local optimum. Beside it may get better chromosome, the search process also can have the capacity jumping out the local optimum. Although we can not sure which parts of chromosome is good, we can expect the solution to be located in the valley of the global optimum.

We propose a new crossover operator named “head-tail crossover (HTC)”. By using the HTC, we can not only keep some sub-gene of the origin chromosomes but also jump out from the local optimum. In the HTC, for example, the chromosome of parents are g_a = (1, 3, 2, 4, 6, 5, 7), g_b = (2, 7, 5, 4, 3, 1, 6). The process of crossover is shown in Fig.2 and algorithm as following.

Firstly, we pick up the gene from g_a in position 0, and then add the selected gene to new chromosome from head to tail. On the other hand, we pick up the gene form g_a in position 6, and then add the selected gene to new chromosome from tail to head. If the selected gene which have add into the new chromosome, discard the selection and pick up next gene from its process. This kind of construction skill can not only avoid to get a infeasible solution but also remain the partial structure of the parents.

Algorithm: head-tail crossover

Chromosomes of parents $g_a = (a_0, a_1, a_2, \ldots, a_{n-1})$, $g_b = (b_0, b_1, b_2, \ldots, b_{n-1})$, the offspring chromosome $g_c = (c_0, c_1, c_2, \ldots, c_{n-1})$.

Procedure crossover (g_a, g_b, g_c)

begin

a_index = 0
b_index = *n*-1
c_head = 0
c_tail = *n*-1
round = 0
While round less than *n*

begin
if (round mod 2) = 0 **then**
while a_{a_index} does not have placed on g_c
a_index++
$c_{c\_head} = a_{a\_index}$
a_index++
c_head++
else
while b_{b_index} doesn't have placed on g_c
b_index--
$c_{c\_tail} = b_{b\_index}$
b_index--
c_tail--
round++
end
end

Mutation by Local Search

The 2opt we have introduced in the local search and tabu search. In the GA, we still need to use the local search to sure the solution is the best of its local area. In genera, GA use the crossover operator to keep the good parts of the parents that search process can find the better solution; then use the Mutation operator to try jumping out the local optimum. But in the hybrid method we use, the mutation by local search is the main idea to find local optimum. We use the crossover to avoid the search process fall into local optimum. Therefore, when we define the parameter of GA, the crossover rate and the mutation rate is not like the traditional design. We should define the parameter by the computational experiments.

4.4 Iterated Local Search

Iterated local search (ILS) is a simple and powerful meta-heuristic that iteratively applies local search to modify the current solution. The original concept is based on the observation that local search are trapped in local optimum easily. To resolve this problem, there are two major mechanisms in ILS, i.e. *perturbation* and *acceptance* criterion. Instead of restarting the local search from a random initial solution while it can't improve

any more, ILS moved to a neighbor solution by some random rule in the search process. Perturbation is introduced to help local search escape from local optima and does not always start from an independent random initial solution. The acceptance criterion is used to determinate which solution can be kept and then generate the next solution by the perturbation. Following is the pseudo code of an iterated local search procedure. Then we will show the detail for all operators used for the ILS to the TSP-WOCT.

Procedure *Iterated Local Search*
Let S_0 be an initial solution.
Let S^* =be the solution obtained from applying local search on S_0.
Repeat
S' = Perturbation (S^*)
$S^{*}{}'$ = LocalSearch (S')
If $Z(S^{*}{}')<Z(S^*)$ then $S^* = S^{*}{}'$ and $Z(S^*) = Z(S^{*}{}')$ /* Acceptance Criteria */
Until termination condition is met
End

Initial Solution

The initial solution is constructed by a greedy method, same as that used above.

Local search

In general, any local search algorithm can be used, but the chosen will affect solution quality and computing time of the ILS algorithm. Preliminary experiments suggest that 2-Opt is better than 2-exchange in computation, so we choose 2-opt to be the local search algorithm. For large-scale problems, the computation time for local search still grows fast. To resolve this difficulty, we use a modified local search strategy to replace the original algorithm. The new strategy is called best-improvement local search algorithm; that is, for each randomly chosen edge we examine all possibilities for swapping and if one or more better neighbors are found, we chose from among them the one with the greatest improvement and execute the swapping.

Perturbation

The adoption of perturbation is important, since it should be strong enough to allow the search process leave the local optimum and to find another better one. Moreover, it also should be control to keep some characteristics of the current optimum. It is not only allow the search process to run fast but also avoid the perturbation of the current optimum is so large that the search process is similar to starting from a new, randomly solution.

For perturbation on our problem, we consider a simple modification to change the solution structure. It contains two swap-move and one 2-exchange. The swap-move will swap the jobs at positions i and i+1, where i is randomly selected.

Acceptance Criterion

For acceptance criterion, we consider the traditional choice that the ILS applications only accept the better local optimum.

Chapter 5 Computational Experiments

This chapter is to examine the efficiency and effectiveness of the proposed algorithms. We designed a series of computational experiments. The algorithms were implemented in Java and tested on a personal computer with a Pentium D 2.8GHz CPU and 512 MB memory running Microsoft Windows XP. The test data were obtained by randomly generating points with real coordinates on a 100×100 square plane. The costs (setup time between jobs) are thereby defined in the Euclidean distance (rounded to integer).

To analyze the efficiency of our binary integer program, we use CPLEX to implement the two formulations **IP_Latency_TSP** and **BIP_Latency_TSP** to delivery man problem which mentioned in Chapter 2. For each problem size, five instances were executed. The results are shown on the Table1. The “optimal” row indicates the number of instances out of each five that were optimally solved, and the “Avg Time” row gives the average execution time for each five instances. It is clear that the binary formulation permits shorter execution times than the integer program. Although our binary program is faster than the integer program, we do not claim absolute superiority. As mentioned in Chapter 2, the integer program provides good structures for other research purposes.

The second part of our experiments is to investigate the efficiency and effectiveness of the dynamic programming algorithm and the proposed approximation approaches. The number of the orders is a given constant and the weight of each order w_i is generated from the uniform interval [1, 10]. Each job is also randomly distributed to the orders.

In the computational study, there are different numbers of nodes represented by n and different numbers of orders represented by K. One hundred instances were generated for each case, and the average of these one hundred instances was presented as the computational result. The experiment analysis basically consists of three parts of comparisons. The first part is the performance comparison between the dynamic

programming algorithm and all other meta-heuristic methods. Two performance indices are used: the elapsed run time and the relative errors (|opt-apprx|/max{opt,apprx}) (Ausiello et al., 2005). Due to the difficulty of obtaining optimal solutions, within a reasonable time frame, by the dynamic programming algorithm as the problem size increase, the second and third parts only conduct the performance comparisons among the specified meta-heuristic methods. The numerical statistics are displayed through Tables 2 to 9.

Small-scaled problems, n = 10, 14, 18, 20, 22, or 24 and K = 3 or 5, are shown in Table 2 and Table 3. The tables illustrate the average run time of each case by different algorithms adopted in the research. Because the exact dynamic programming algorithm could provide the optimal solutions within a reasonable time frame for small problems, error ratio and hit frequency of approximate algorithms are also depicted in the tables. The local search methods, both H1_LS and H2_LS, tend to be easily trapped in the local optima in the experiments. Consequently, the overall hit frequencies and error ratios of the local search approaches appear to be inferior to other methods. In addition, if we examine the computational results closely, we may observe the superiority of the 2-Opt strategy for constructing neighborhood structures in the local search approaches. Similar verification could also be applied to the tabu search algorithm. Therefore, for simplification, we only adopt TS_2Opt to represent tabu search for the rest of all the computational analyses.

Tabu search method demonstrates the ascendancy, in hit frequency, for small-scaled problems. Optimal solutions could be obtained for most of the test cases by the tabu search method. On the other hand, for GA and ILS, they both exhibit high hit frequencies with low error ratios, but they work less efficiently than the tabu search in terms of runtime. As the number of jobs rises (n exceeds 18), the exact dynamic programming algorithm requires a longer run time than all approximation methods. The low error ratios are below 0.3 percent for all heuristic methods. This implies that the number of orders does not affect the performance of these algorithms when the problem size is small.

Tables 4 to 6 exhibit some medium-scaled problems, as the number of nodes is greater than twenty (n = 30, 40, 50, or 60, K = 3, 6, or 15). Because the dynamic programming algorithm fails to produce optimal solutions within a desirable time frame, the performances of the heuristic methods are compared with each other. The "win" columns in the tables indicate how may times a heuristic method is able to provide the best approximate solution. The "deviation" columns display the deviation of each individual approximation solution away from the "best" one. Some observations, for various n and various K, are listed as follows.

1. Different numbers of nodes, n: The numbers recorded in the "win" columns drop as n increases for GA and tabu search methods; on the contrary, ILS has the ascendancy with n. When n = 30, tabu search was able to obtain the best approximation most often, while GA and ILS could also provide the solution more than 50 times with the deviation bounded within 1%. When $n \geq 40$, ILS ascended as the leader among all tested heuristic methods. Such phenomenon is aggrandized as the number of nodes increases.
2. Different numbers of orders, K: The run times and deviations for all three heuristic methods increase with the number of orders. Moreover, GA provided less "best" solutions as the number of orders increases. This implies that GA performs worse when more orders are received. On the contrary, such performance deterioration along with ascending K is not evident for either tabu search or ILS.

Tables 7 to 9 summarize the statistics for the large-scaled cases. We examined four different numbers of nodes (n = 70, 80, 90, or 100), along with various numbers of orders (K = 6, 15, or 25). It is easy to see that ILS outperforms all other meta-heuristic methods when coping with large-scaled cases. While genetic algorithm failed to improve the solution qualities for large scaled problems, it also required a much longer execution time in comparison with other meta-heuristic approaches adopted in the research. Tabu search seems to work efficiently by accomplishing one hundred jobs in

ten seconds. However, tabu search also failed to improve the solution even if the number of iteration is raised to 100,000 times. This fact implies that the tabu search might not find a better neighbor solution when the problem size increases. Therefore, compared with the tabu search approach, ILS effectively improves the solution quality with a moderate compromise of elapsed run time.

Chapter 6 Conclusion

In this thesis, we considered a variant of the latency TSP by including order delivery. We first gave a binary integer programming model that is different from the integer program known in the literature. To produce optimal solutions, a dynamic programming algorithm was devised. The complexity is $O(n^2 2^n)$, which still exhibits an exponential growth of computing time with respect to the problem size. A special case that can be solved in polynomial time was also identified. By the complexity nature of the problem, several approximation approaches, including local search, tabu search and genetic algorithms, were designed for producing approximate solutions in an acceptable time. From the statistics, it is clear that iterated local search outperforms all other approaches in terms of solution quality. The superiority becomes stronger when the number of jobs increases. If we consider the time elapsed for producing solutions, tabu search dominates other approaches. Comparing different neighborhood structures, we find that 2-OPT is better than 2-Exchange. Genetic algorithm seems to be inferior in all settings. We need to emphasize that the experiments are not designed and conducteded to claim the superiority of one method over others for the studied, but to provide preliminary investigations of the deployment of these methods.

For further research, seeking quality lower bounds for the development of branch-and-bound algorithms to optimally solve the TSP-WOCT can be an interesting direction. Moreover, the existence of sharp bounds will also provide theoretical insights into the studied problem and give underestimates of optimal objective values for evaluating the approximation approaches. The concept of order delivery also exhibits significant room of research for incorporating aggregation of individual entities (jobs, nodes, elements) in numerous optimization problems.

References

[1] Afrati, F., Cosmadakis, S., Papadimitriou, C., Papageorgiou, G., Papakostantinou, N., (1986). “The complexity of the traveling repairman problem,” Theoretical Informatics and Applications, 20, 79-87.

[2] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M., (1999). “Complexity and Approximation,” Springer, chapter 3.

[3] Archer, A., Williamson, D., (2003). “Faster approximation algorithms for the minimum latency problem,” Society for Industrial and Applied Mathematics, 88-96

[4] Blum, A., Chalasani, P., Coppersmith, D.,Pulleyblank, B., Raghavan, P., Sudan, M., (1994). “The minimum latency problem,” the 26th ACM Symposium on the Theory of Computing, 163-171.

[5] Minieka, E., (1989). “The delivery man problem on a tree network.” Annals of Operations Research, 18, 261-266.

[6] Fischetti, M., Laporte, G., and Martello, S., (1993). “The delivery man problem and cumulative matroids,” Operations Research, 41, 1055–1064.

[7] Glover, F., (1989). “Tabu search - Part I,” ORSA Journal on Computing, 1, 190-206.

[8] Glover, F., (1990). “Tabu search - Part II,” ORSA Journal on Computing, 2, 4-32.

[9] Glover, F., (1990). Laguna, M., “Tabu Search,” Kluwer Academic Publishers, Hingham, M.A.

[10] Garcia, A., Jodr’a, P., Tejel, J., (2002). “A note on the traveling repairmen problem,” Networks, 40, 27-31.

[11] Goemans, M., and Kleinberg, J., (1998). “An improved approximation ratio for the minimum latency problem,” Math. Prog., 82, 111-124.

[12] Holland, J.H, (1975). “Adaptation in natural and artificial systems,” University of Michigan Press.

[13] Lin, S., Kernighan, B. W., (1973). “An effective heuristic algorithm for the

traveling salesman problem," Operations Research, 21, 498-516.

[14] Li, W. (2005). "Dynamics of search trajectory in traveling salesman problem," Journal of Heuristic, 11, 507-524.

[15] Ji, M., He, Y., and Cheng, T.C.E, (2007). "Batch delivery scheduling with batch delivery cost on a single machine," European Journal of Operational Research, 176, 745-755.

[16] Sengoku, H., Yoshihara, I., (1993). "A fast TSP solution using GA on java,".

[17] Cheng, T.C.E., Kahlbacher, H.G., (1993). "Scheduling with delivery and earliness penalties," Asia-Pacific Journal of Operational Research, 10, 145-152.

[18] Cheng, T.C.E., and Gordon, S., (1994). "Batch delivery scheduling on a single machine," Journal of the Operational Research Society, 45, 1211-1215.

[19] Cheng, T.C.E., Gordon, S., and Kovalyov, M.Y., (1996). "Single machine scheduling with batch deliveries," European Journal of Operational Research, 94, 277-283.

[20] Cheng, T.C.E., Kovalyov, M.Y., Lin, B.M.T., (1997). "Single machine scheduling to minimize batch delivery and job earliness penalties," SIAM Journal on Optimization, 7, 547-559.

[21] Wu, B.Y, (2000). "Polynomial time algorithms for some minimum latency problems," Information Processing Letters, 225-229.

[22] Wu, B.Y., Huang, Zhan, Z.-N., (2004). "Exact algorithms for the minimum latency problem," Information Processing Letters, 92, 303-309.

[23] Yang, X., (2000). "Scheduling with generalized batch delivery dates and earliness penalties," IIE Transactions, 32, 735-741.

APPENDIX

Table 1: CPLEX implementations of **IP_Latency_TSP** and **BIP_Latency_TSP**.

n	8		10		12		14		16		18		20		22	
	integer	binary	Integer	binary	integer	binary	integer	binary	integer	binary	integer	binary	integer	binary	integer	binary
optimal	5/5	5/5	5/5	5/5	0/5	5/5	0/5	5/5	0/5	5/5	0/5	5/5	0/5	5/5	0/5	5/5
Avg time	0.418	0.038	20.366	0.366	-	0.466	-	3.818	-	4.764	-	17.95	-	58.294	-	148.872

Table 2 Small size, order=3

	DP	LS-2exchange			LS-2opt			TS-2opt			TS-2exchange			GA			ILS		
n	time	time	hit	error	time	hit	error	time	hit	error	time	hit	error	time	hit	error	time	hit	error
10	0.00	0.00	36.0%	3.7%	0.00	52.0%	1.7%	3.86	100.0%	0.0%	3.61	100.0%	0.0%	0.02	99.0%	0.0%	0.05	99.0%	0.0%
14	0.04	0.00	15.0%	5.7%	0.00	35.0%	2.3%	4.26	100.0%	0.0%	3.88	89.0%	0.2%	0.08	99.0%	0.0%	0.15	97.0%	0.0%
18	1.87	0.01	10.0%	8.1%	0.01	24.0%	3.7%	4.46	100.0%	0.0%	4.07	65.0%	0.5%	0.26	86.0%	0.1%	0.33	88.0%	0.2%
20	12.37	0.02	2.0%	7.6%	0.01	10.0%	4.8%	4.65	99.0%	0.0%	4.27	43.0%	0.8%	0.46	85.0%	0.1%	0.46	86.0%	0.1%
22	76.19	0.01	3.0%	9.4%	0.01	13.0%	4.9%	4.94	98.0%	0.0%	4.54	32.0%	1.3%	0.64	82.0%	0.1%	0.60	79.0%	0.2%
24	386.91	0.01	1.0%	9.8%	0.01	0.0%	6.5%	5.02	94.0%	0.0%	4.61	17.0%	2.0%	1.08	65.0%	0.3%	0.85	72.0%	0.3%

Table 3 Small size, order=5

	DP	LS-2exchange			LS-2opt			TS-2opt			TS-2exchange			GA			ILS		
n	time	time	hit	error	time	hit	error	time	hit	error	time	hit	error	time	hit	error	time	hit	error
10	0.00	0.00	33.0%	4.0%	0.00	53.0%	1.5%	3.92	100.0%	0.0%	3.67	100.0%	0.0%	0.02	99.0%	0.0%	0.06	99.0%	0.0%
14	0.04	0.00	13.0%	5.6%	0.00	32.0%	2.3%	4.37	100.0%	0.0%	4.01	94.0%	0.1%	0.10	98.0%	0.0%	0.17	94.0%	0.1%
18	1.86	0.01	2.0%	8.6%	0.01	11.0%	4.5%	4.72	100.0%	0.0%	4.32	61.0%	0.6%	0.35	87.0%	0.1%	0.38	91.0%	0.1%
20	12.08	0.02	4.0%	7.9%	0.01	5.0%	5.2%	4.78	100.0%	0.0%	4.39	54.0%	0.5%	0.58	91.0%	0.0%	0.56	86.0%	0.2%
22	75.78	0.01	4.0%	5.9%	0.01	6.0%	9.3%	5.27	100.0%	0.0%	4.80	24.0%	1.4%	0.97	70.0%	0.2%	0.74	77.0%	0.3%
24	395.02	0.01	1.0%	9.7%	0.01	3.0%	6.2%	5.0	89.0%	0.0%	4.61	25.0%	1.6%	1.52	61.0%	0.3%	0.91	72.0%	0.3%

Table 4 mid size, order=3

	TS-2opt			TS-2exchange			GA			ILS		
n	time	win	deviation	Time	win	deviation	time	win	deviation	time	win	deviation
30	5.17	72	0.13%	4.66	0	3.78%	3.27	52	0.42%	1.58	62	0.49%
40	5.84	37	0.67%	5.31	0	6.71%	15.05	25	0.76%	3.80	61	0.61%
50	6.48	22	1.25%	5.85	0	8.97%	20.47	16	1.52%	7.68	66	0.43%
60	7.23	14	2.12%	6.52	0	10.45%	34.96	10	2.28%	13.39	77	0.22%

Table 5 mid size, order=6

	TS-2opt			TS-2exchange			GA			ILS		
n	time	win	deviation	time	win	deviation	time	win	deviation	time	win	deviation
30	5.55	60	0.22%	5.04	0	3.47%	6.08	32	0.46%	1.90	55	0.74%
40	6.46	27	1.04%	5.85	0	5.66%	18.76	23	1.16%	4.62	55	0.71%
50	6.82	23	1.64%	6.22	0	7.89%	34.93	16	1.89%	9.22	63	0.48%
60	7.58	14	3.13%	6.84	0	10.21%	60.92	9	3.915	15.99	84	0.25%

Table 6 mid size, order=15

	TS-2opt			TS-2exchange			GA			ILS		
n	time	win	deviation	time	win	deviation	time	win	deviation	time	win	deviation
30	6.23	69	0.15%	5.79	4	3.07%	4.88	24	0.77%	2.57	53	0.62%
40	7.14	28	1.02%	6.63	1	4.99%	12.82	18	1.76%	6.16	58	0.53%
50	7.88	14	2.03%	7.23	1	7.57%	27.25	14	2.82%	12.69	71	0.46%
60	8.89	10	2.87%	8.33	1	8.92%	62.97	8	3.39%	21.74	81	0.34%

Table 7 large size, order=6

	TS-2opt			TS-2exchange			GA			ILS		
n	time	win	deviation	time	win	deviation	time	win	deviation	time	win	deviation
70	8.51	7	4.03%	7.59	0	11.45%	35.34	2	5.15%	26.78	91	0.14%
80	9.15	7	4.68%	8.18	0	12.01%	64.67	6	4.93%	40.61	88	0.16%
90	10.1	0	5.77%	8.90	0	13.01%	53.92	1	6.10%	56.77	99	0.01%
100	10.84	3	6.07%	9.64	0	14.23%	82.32	2	6.27%	80.65	95	0.04%

Table 8 large size, order=15

	TS-2opt			TS-2exchange			GA			ILS		
n	time	win	deviation	time	win	deviation	time	win	deviation	time	win	deviation
70	9.37	9	4.04%	8.62	0	10.91%	66.28	1	5.47%	33.39	90	0.11%
80	10.18	3	5.52%	9.25	1	12.02%	100.77	2	6.28%	50.42	94	0.11%
90	11.07	3	6.33%	9.99	0	12.82%	100.36	1	7.24%	72.60	96	0.06%
100	12.15	0	6.95%	10.81	0	14.59%	93.07	0	7.52%	100.88	100	0.00%

Table 9 large size, order=25

	TS-2opt			TS-2exchange			GA			ILS		
n	time	win	deviation	time	win	deviation	time	win	deviation	time	win	deviation
70	10.29	7	4.59%	9.45	0	14.56%	82.93	2	5.75%	41.23	91	0.13%
80	11.38	5	6.61%	10.27	0	17.06%	133.82	0	5.75%	61.22	95	0.02%
90	11.95	1	9.31%	10.73	0	19.89%	163.79	0	6.57%	86.86	99	0.00%
100	12.77	0	10.55%	11.37	0	19.99%	197.84	1	6.93%	21.74	99	0.00%