Applied Mathematics and Computation 222 (2013) 58–71

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Sequence-dependent scheduling with order deliveries

B.M.T. Lin [a,*], P.Y. Yin [b], Y.S. Liu [a]

[a] Department of Information and Finance Management, Institute of Information Management, National Chiao Tung University, Hsinchu 300, Taiwan
[b] Department of Information Management, National Chi Nan University, Puli 545, Taiwan

ARTICLE INFO

Keywords:
Sequence-dependent setup
Order delivery
Satellite imaging scheduling
Total weighted completion time
Meta-heuristics

ABSTRACT

This paper studies a sequence-dependent scheduling problem incorporating order delivery, motivated by satellite imaging scheduling. A set of jobs is to be processed on a single machine and each job belongs to a specific group. The completion time of a group is the moment when all jobs belonging to this group are completed. The problem is to determine a processing sequence of the jobs such that the sum of weighted completion times over all groups is minimized. We present a binary integer program 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 integer program and the solution algorithms. Numerical statistics suggest that the binary integer program can reach optimal solutions faster than the integer program existing in the literature, and the iterated local search algorithm outperforms other approaches when the number of jobs increases.

1. Introduction

This paper considers a single-machine scheduling problem with sequence-dependent setups and order deliveries. In most scheduling problems, the identities adopted 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 here, jobs belonging to the same order or group will be delivered in a single batch, and the objective function considers order/group completion times instead of job completion times, the latter are commonly adopted in the scheduling literature. An order is fulfilled when the last job of it is completed, i.e. the order completion time is equal to the completion time of the last job in this order. All jobs, from several different orders, are processed in a single-machine scheduling environment. A sequence-dependent setup time exists between any two consecutive jobs, and the job processing times are negligible. The objective is to determine a job processing sequence so as to minimize the sum of weighted completion times over all groups.

The problem setting originates from a simplified model of satellite imaging scheduling, although implications in many other areas are also admissible. In the scheduling problem of satellite imaging, several requests/orders of various numbers of photographs or observations need to be fulfilled via satellite imaging operations. Each order consists of several photos to take and is associated with a weight that characterizes the importance or the degree of urgency. At the completion of a photographing task, the satellite should adjust the imaging angle of the camera for the next observation spot. Such alternation of camera angle requires great accuracy with technical concerns, and takes a much longer processing time compared with the

* Corresponding author.
E-mail address: bmtlin@mail.nctu.edu.tw (B.M.T. Lin).

0096-3003/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.amc.2013.06.087

photo shooting time. Therefore, sequence-dependent setup times are indispensable between any two consecutive imaging operations, while the photographing time may be ignored. Various institutions may place orders of their own interests. The objective is to accomplish all requests according to a photographing schedule that minimizes the sum of weighted completion times over all orders.

In the domain of NASA's Earth Observing Systems [27,31,32], several critical scheduling issues arise due to the scarcity of resource. For assigning communication tasks to relay satellites, Wolfe and Sorensen [33] propose a priority dispatch heuristic and a look ahead heuristic to generate solutions (permutations of tasks) that serve to form the initial population of genetic algorithm for further improvements. A recent paper by Wang et al. [30] also addresses the scheduling problem of communication tasks of data acquisition in the satellite management domain. Lemaitre et al. [21] propose and study the selection and scheduling problem of multiple earth observation satellites. A greedy algorithm, a dynamic programming algorithm, a constraint programming approach and a local search method are adopted for a simplified model. Lin et al. [22] consider imaging scheduling incorporating reward opportunities, changeover setups, cloud coverage, and the availability of the spacecraft resource. Lagrangian relaxation, linear search and tabu search are deployed to deal with the scheduling problem. Bianchessi et al. [4] consider an imaging setting that involves multiple satellites, multiple users and multiple orbits. Tabu search is adopted to provide approximate solutions, which are evaluated through upper bounds generated by a column generation procedure. The problem setting considered in this paper is a deterministic model of imaging scheduling with the new feature of order delivery, which is commonly adopted in scheduling theory and applications.

The studied model may also be adopted by some other applications under different scenarios, such as in vehicle routing problems (VRP). Sequence-dependent scheduling has direct links to the traveling salesman problem (TSP) [17]. 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 node sequence attaining the objective of minimizing the sum of completion times of nodes on the network. Nevertheless, 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 a much higher degree of urgency than the rest of the orders. Some other transportation problems, such as VRP, also incorporate the concept of order delivery in order to capture and interpret complicated down-to-earth problems in the real world.

In this paper, we denote the studied problem by the TSP-WOCT (Weighted Order Completion Times). The TSP-WOCT conjoins the classical deliveryman problem with batch delivery scheduling on a single machine to compose a novel model. The deliveryman problem is similar to the well-known traveling salesperson problem with a difference 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 is focused instead on customer satisfaction. It is interesting that deliveryman problem has been studied under different titles, such as "the traveling repairman problem" [1,15] and "deliveryman problem" [11,24]. Some recent researches about the minimum latency problems actually cope with the same problems [5,34]. The minimum latency problem is known to be NP-hard. It can be solved in polynomial time for some specific graphs, such as paths [1,15], edge-unweighted trees [24], tree of diameter three [5]. Wu et al. [35] propose an exact algorithm by applying a dynamic programming algorithm along with a branch and bound technique for small-scale problem instances. In addition, some researches develop approximation algorithms and analyze their performances [2,16].

In scheduling problems with sequential batch processing, jobs are grouped into batches and processed contiguously. A batch is completed whenever the jobs in the batch are all completed, and jobs of the same batch have a common completion time. For most scheduling problems, objective functions are defined in terms of job completion times. Some researches on the other hand focus on the concept of batch delivery and study objective functions that consider batch delivery times. Cheng and Kahbacher [8] first consider the single-machine scheduling problem with the objective function of the sum of weighted batch delivery times and the objective function of the sum of total weighted earliness and a batch delivery penalty, depending on the number of batches. Cheng and Gordon [6] solve the general version of the single machine with batch delivery scheduling problem by a dynamic programming algorithm. Cheng et al. [7] later prove the strong NP-hardness of the minimization of weighted batch delivery times and propose polynomial algorithms for two restricted cases. Cheng et al. [9] propose another objective function to the batch delivery scheduling problem by minimizing the total weighted earliness and mean batch delivery time. Yang [36] studies the single machine scheduling problems with generalized batch delivery dates and earliness penalties and prove the problems to be strongly NP-hard. Ji et al. [19] consider a scheduling problem with batch delivery to minimize the sum of weighted flow times and delivery cost. For a parallel-machine setting, Wang and Cheng [29] give NP-hardness proofs and develop a dynamic programming algorithm for producing optimal solutions. Tian et al. [28] investigate the same setting by designing and analyzing on-line algorithms. Yin et al. [37–39] incorporate batch deliveries into scheduling problems with the decision of due-date assignment. Combining batch delivery in the scheduling context with sequence-dependent setup times and the application of satellite imaging, this paper formulates and studies the scheduling problem as the latency TSP with order delivery.

The rest of this paper is organized as follows. In Section 2, we formally define the studied scheduling problem. A binary integer programming model will be presented to describe the problem. Section 3 develops an exact algorithm based on the dynamic programming approach. A special case is also investigated. In Section 4, we design a tabu search algorithm, an

iterated local search algorithm and a genetic algorithm to produce approximate solutions. Section 5 is dedicated to a computational study to evaluate the performance of the proposed algorithms. Conclusions and suggestions for further research are addressed in Section 6.

2. Problem formulation and mathematical model

In this section, we give formal statements of the scheduling problem and present mathematical models to describe the problem.

Formally, the TSP-WOCT can be described from the perspectives of scheduling theory as follows. There are K orders $\boldsymbol{O} = \{O_1, O_2, \ldots, O_K\}$ to be processed on a single-machine. Each order O_k, $1 \leqslant k \leqslant K$ consists of n_k jobs and has a non-negative weight w_k. Let $n_1 + n_2 + \cdots + n_K = n$ and $N = \bigcup_{k=1}^{K} O_k = \{J_1, J_2, \ldots J_n\}$ denote the set of all jobs of the K orders. The processing time of any job of N is negligible. A non-negative sequence-dependent setup time d_{ij} is required if job J_j is the immediate successor of job J_i, regardless of which orders they belong to. A dummy job J_0 is introduced 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 the last job of O_k is finished. The objective is to determine a permutation of the jobs, which minimizes the sum of weighted completion times $\sum_{k=1}^{K} w_k O_k$. In this paper, node, job and city are used interchangeably for describing the basic elements of our scheduling problem.

Example: There are five jobs J_1, J_2, J_3, J_4, J_5 belonging to two orders, $O_1 = \{J_1, J_2, J_3\}$ and $O_2 = \{J_4, J_5\}$. Order weights are $w_1 = 8$ and $w_2 = 5$, and 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 $S = J_2 J_1 J_4 J_3 J_5$. The completion times of the jobs are 5, 15, 21, 28 and 33, respectively. The completion times of order O_1 and orderO_2 are subsequently $C_1 = 28$ and $C_2 = 33$, respectively. The total weighted completion time is calculated as $8 \times 28 + 5 \times 33 = 389$.

In Fischetti et al. [11], an elegant integer programming formulation was proposed for the latency TSP. In this model, integer variable x_{ij} is used to indicate the inclusion 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 the kth arc in the Hamiltonian tour. The model uses $O(n^2)$ integer variables and $O(n^2)$ constraints. The model is shown below for a comparison with the models presented for the TSP_WOCT.

IP_Latency_TSP

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

Subject to

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

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

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

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

$$\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj} = \begin{cases} 1 - n & \text{if } k = 1; \\ 1 & \text{if } 2 \leqslant k \leqslant n \end{cases} \tag{5}$$

$$x_{ij} \geqslant 0, \quad 1 \leqslant i, j \leqslant n; \tag{6}$$

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

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

Program IP_Latency_TSP developed by Fischetti et al. [11] has been widely adopted in the literature for researches on TSP-related subjects because it provides theoretical structures that facilitate the development of approximation algorithms and lower bounds. In this paper, we consider another approach for formulating the latency problem. The necessity for developing a new integer program is due to the fact that knowing the position of an arc (i, j) in a Hamiltonian cycle does not reveal sufficient information for calculating the contribution that arc (i, j) makes to the objective function of total weighted order completion time.

In the new formulation, we use binary variable x_{ijl}, $0 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant n,\ 1 \leqslant l \leqslant n$, to indicate whether arc (i, j) from job J_i to job J_j is the lth edge in the TSP sequence. Recall that dummy job J_0 will be scheduled first so that job index i will start from 0. A total of n arcs rather than $n-1$ are selected to compose a complete solution.

BIP_Latency_TSP

$$\text{Minimize } Z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}(n-l+1)x_{ijl}d_{ij}$$

Subject to

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

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijl} = 1, \quad 2 \leqslant l \leqslant n; \tag{9}$$

$$\sum_{j=1}^{n}\sum_{l=2}^{n} x_{ijl} = 1, \quad 1 \leqslant i \leqslant n; \tag{10}$$

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

$$\sum_{i=0}^{n} x_{ijl} = \sum_{i=1}^{n} x_{ji(l+1)}, \quad 1 \leqslant l \leqslant n-1,\ 1 \leqslant j \leqslant n; \tag{12}$$

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

In the binary integer program BIP_Latency_TSP, constraints (8)–(13) do not explicitly embed the contribution of each arc into the decision variable x_{ijl}. Therefore, the actual contribution of the lth arc is calculated in the objective function as the arc cost multiplied by $n - l + 1$. Constraints (8–11) are self-explanatory. Constraints (12) indicate that each job J_j is the start node of the lth arc if and only if it is the end node of the $(l+1)$th arc. In our formulation, $O(n^3)$ decision variables and $O(n^2)$ functional constraints are involved. Compared with IP_Latency_TSP, our model BIP_Latency_TSP uses more decision variables. But this is compensated by the use of binary variables instead of integer ones. Another merit is the applicability to the scheduling problem with batch deliveries.

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

Problem TSP-WOCT

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

Subject to
Constraints (8)–(13)

$$t_j + \left(1 - \sum_{i=0}^{n} x_{ijl}\right) M \geqslant \sum_{u=0}^{n}\sum_{v=1}^{n}\sum_{r=1}^{l} x_{uvr} d_{uv}, \quad 1 \leqslant l \leqslant n,\ 1 \leqslant j \leqslant n; \tag{14}$$

$$C_k \geqslant t_j, \quad J_j \in O_k; \tag{15}$$

$$t_j \geqslant 0, \quad 1 \leqslant j \leqslant n; \tag{16}$$

In the above formulation, constraints (14) confine the completion time of job J_j, depending on if J_j is the end node of the lth arc. The variable M is a big number, as chosen appropriately. For example, we can set $M = \sum_{j=1}^{n} max_{0 \leqslant i \neq j \leqslant n}\{d_{ij}\}$. If $x_{ijl} = 1$, then t_j is the accumulated setup times before its processing. On the other hand, when $x_{ijl} = 0$, t_j could be set to an arbitrary non-negative number. Constraints (15) are given for retrieving the completion times of all orders. This program also uses $O(n^3)$ decision variables and $O(n^2)$ functional constraints.

3. Dynamic programming and a special case

In this section, 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 latency TSP has been known to be NP-hard. The time required for producing optimal solutions to the latency TSP increases exponentially when the problem size increases. The latency TSP is a special case of the TSP-WOCT where each order consists of exactly one job. In general, the TSP-WOCT is at least as hard as the latency TSP. Still, we seek to find a systematic way for generating optimal schedules to the TSP-WOCT. In this paper, we adopt the dynamic programming approach, which is one of the most widely adopted implicit enumerative methods. Following the standard design technique, Wu et al. [35] develop a dynamic programming algorithm for the minimum latency problem. The completion times of interest in the TSP-WOCT is calculated over orders. As the two problems demonstrate several similar features, we adopt the dynamic programming approach 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;
- $\prod_{J_j,N'}$: 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 $\prod_{J_j,N'}$;
- $L(S(N',J_j))$: the 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 partial sequence $S(N',J_j)$, define its contribution to the objective function by

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

The first part is the weighted cost already incurred for completed orders, and the second part reflects the cost that the sequence will incur to the unfinished orders.

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

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

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

DP_WOCT:

Initial conditions:

$$f(N',J_j) = \begin{cases} 0, & \text{if } N' = \{J_0\} \text{ and } j = 0; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursion:

$$f(N',J_j) = \min_{J_i \in N' \setminus \{J_j\}} \left\{ f(N' \setminus \{J_j\},J_i) + d_{ij} \times w_\alpha + d_{ij} \times \sum_{O_i \in \theta \setminus \theta(N')} w_k \right\},$$

where $w_\alpha = \begin{cases} w_k, & \text{if the order } O_k \text{ containing job } J_j \text{is a subset of } N' \\ 0, & \text{otherwise} \end{cases}$.

Goal:

Determine $\min_{J_j \in N}\{f(N, J_j)\}$.

Regardless of whether or not job J_i finishes the order it belongs to, when J_i is appended to the partial sequence corresponding to $f(N' \setminus \{J_j\}, J_i)$, an extra cost of $d_{ij} \times \sum_{O_i \in \theta \setminus \theta(N')} w_k$ is introduced to the remaining unfinished orders. The initial condition $f(\{J_0\}, J_0) = 0$ is defined as the boundary of recursion. The dynamic program has $O(n2^n)$ states, each of which can be computed in $O(n)$ time if $\sum_{O_i \in O \setminus \theta(N')} w_k$ 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, $1 \leqslant j \leqslant n$, 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. Consider an optimal schedule that does not satisfy the conditions of the lemma. Let order O_k be the first completed order whose jobs are not processed in consecutive positions. We shift all jobs, possibly excluding the job in the first position, of order O_k right to the positions immediately preceding the last job of this order. It is evident that the completion time of any order will not increase. The validity can be established through repeated applications of the above argument. □

Let $P_k = \sum_{J_j \in O_k} d_j$ denote the total processing length of order O_k. By assigning a specific job J_j to the first position, all the remaining jobs of $N' \setminus \{J_j\}$ can be scheduled order by order. Therefore, each order can be considered as a composite job. In single-machine scheduling of minimizing the total weighted completion time, an optimal schedule can be obtained by sequencing the jobs in non-decreasing order of the ratios of job processing time and job weight [25].

Lemma 2. *Given a job fixed in the first position, there is an optimal schedule of TSP-WOCT $d_{ij} = d_j$ that schedules the orders by the WSPT (weighted shortest processing time) rule, where P_k of order O_k are calculated on the basis that the first job is excluded.*

Proof. By Lemma 1, an order can be regarded as an aggregate job without interruption of the processing of its jobs, except for possibly the job in the first position. Under the condition of $d_{ij} = d_j$, the processing length of order O_k is the total setup time required for its jobs, i.e. $P_k = \sum_{J_j \in O_k} d_j$. Therefore, scheduling of $n - 1$ jobs reduces to scheduling of K orders, each order O_k of which is characterized by processing length P_k and weight w_k. The WSPT rule thus applies. □

Theorem 1. *The TSP-WOCT $d_{ij} = d_j$ problem is solvable in $O(n \log n)$ time.*

Proof. Consider the WSPT sequence of the K orders and calculate its total weighted completion time. Select a job, remove it from its order and schedule it as the first job in a solution. By a preprocessing procedure, re-calculating the objective function value can be done in constant time. Therefore, the overall time complexity is $O(n \log n)$. □

4. Approximation approaches

Since the TSP-WOCT is NP-hard, the running time required for producing optimal solutions will grow exceedingly fast as the problem size increases. Although a dynamic programming algorithm was developed in the previous section, it is impractical for implementation because the required memory and computing time are not affordable when the problem size is large. In this section 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

Local search or steepest descent starts with an initial solution and moves to a better solution, if available, within a defined neighborhood. This process is repeated until no further improvement is possible. The advantages of local search include simplicity and intuition in design as well as implementation. In this paper, we use a greedy method to construct an initial solution by selecting the shortest outbound arc that emits from the current node at each step. We adopt 2-Opt and 2-Exchange to define the neighborhoods of the employed local search algorithm. The two approaches are denoted by LS-2OPT and LS-2EX, respectively.

4.2. Tabu search

The tabu search approach was first proposed by Glover [12,13] and has been used to solve numerous hard combinatorial optimization problems. The principle of tabu search is to impose some restrictions so as to guide a search course to cover a wide range of candidate solutions and to avoid being trapped by local optima [14]. In the search process, it usually starts from a selected initial solution S_0 and generates a set of neighbors $\{S'\}$ by a heuristic swapping. The objective value is evaluated for each neighbor S' and the best neighbor solution replaces the current solution. The next iteration will be triggered based on the current solution. The procedure is repeated until certain stopping conditions are satisfied. Moreover, a "tabu list", the essence in tabu search, is maintained 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 iterations. The size of a tabu list is defined according to some experiential rules. For different designs customized to various problems, specific features of implementations need to be articulated. In what follows, we describe the major components of our tabu search algorithm.

4.2.1. Initial solution

The initial solution is constructed by a greedy method, same as that used by LS-2OPT and LS-2EX.

4.2.2. Neighborhood structure

We also adopt the two types of swaps, 2-Opt and 2-Exchange, for defining the neighborhood structure, and the resulting tabu search algorithms are denoted by TS-2OPT and TS-2EX, respectively. It is more likely to locate the global optimum if the size of neighbor structure is larger; the running time nevertheless will on the other hand increase. Therefore, we determine the size of the neighborhood through preliminary computational experiences to compromise the quality of the obtained solutions and the elapsed execution time.

4.2.3. Selection of the best neighbor and tabu list

In the search process, a set of neighbors of the current solution is generated during each move session. First, we compute the objective value for each neighbor and record it in the memory. Second, these objective values are sorted in non-decreasing order. Then we can easily locate the best eligible neighbor solution. Moreover, we check the tabu list in order to prevent the search from cycling. If the best neighbor solution is already in the tabu list, we will skip the solution and choose the next neighbor with the best objective value among the rest of all neighbors. Finally, the selected neighbor solution replaces the current one and is recorded in the tabu list in order to avoid being revisited in its tabu tenure.

The size of the tabu list can be fixed or variable. In particular, we adopt the fixed size strategy in our implementation and the size is set to be *seven* based on a preliminary study. When the tabu list is full and a new solution is selected, the earliest solution recorded in the tabu list will be deleted and this new solution will be appended into the list. Therefore, the tabu list is maintained by a queue with the first-in-first-out mechanism.

4.2.4. Aspiration criterion

The aspiration criterion is introduced to let a tabu search algorithm probe diversified regions of the solution space for locating 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 the new solution encountered is better than the incumbent.

4.2.5. Local search

Since the tabu search approach uses a random sampling method in creating a set of neighbor solutions, its performance may exhibit fluctuations. Therefore, we embody a deterministic method to make its performance more consistent in the execution course. When it reaches a solution that is better than the incumbent, the local search method is invoked. Using this design, we can guarantee that the solution found by our tabu search method is the best solution in its local neighborhood area.

4.2.6. 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 section.

4.3. Genetic algorithm

In the early 1970s, Holland [18] introduces the concept of genetic algorithms (GA) 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. These algorithms iteratively improve the quality of solutions and finally produce near-optimal ones. Merits of genetic algorithms include simplicity for implementations, global search capability, and being adaptive to changing conditions in the problem.

Despite of these advantages, genetic algorithms do not use unequivocal rules of how to search for the solutions, resulting in a slower solution finding process. For this reason, we can adopt hybrid methods that combine genetic algorithms with

other techniques. In this paper, a hybrid method combining genetic algorithm and local search is proposed. Local search plays the role of the mutation operator in the proposed genetic algorithm. To prevent the algorithm from getting trapped at the local optima, a head–tail crossover operator is used to enhance the capability for escaping from the local optima. The algorithm consists of the following components.

4.3.1. Initialization

A population of chromosomes is generated in random. We evaluate the fitness of each chromosome. The size of population is denoted by α.

4.3.2. Natural selection

Set the crossover probability P_c. Within each generation, $\alpha \times (1 - P_c)$ 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 that can be added to the new population.

4.3.3. Reproduction

Randomly select $\alpha \times P_c$ chromosomes from the current population and produce an offspring from each pair of the selected individuals.

4.3.4. Mutation by local search

Set the mutation probability P_m. Choose $\alpha \times P_m$ chromosomes randomly and improve on 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.

Step 1. Initialization

Chromosome coding and representation are the most crucial to the design of genetic algorithms. It is not trivial to encode the problem solution into a chromosome neither to design an easy-to-compute fitness function. Because the studied problem is a variant of the TSP, we adopt the permutation representation to encode the chromosome. An example of a chromosome with 7 genes is given by (2, 7, 5, 4, 3, 1, 6).

With the chromosome coding scheme, an initial population comprised of α chromosomes is generated. In order to increase the diversity of the initial population and to avoid premature convergence to local optima, half of the members of the initial population are produced in random, while the other half of the members are produced by a greedy method to ensure that there is fertile gene information existing in the population for evolution.

Step 2. Natural selection

We first set the crossover probability P_c. Following the notion of elitist strategy, from each generation we reserve the best $\alpha \times (1 - P_c)$ chromosomes, with reference to their fitness values, for inclusion into the next population without performing crossover. The remaining $\alpha \times P_c$ chromosomes are selected based on the selection probability and are subject to the crossover operation that would produce offspring. The selection probability is proportional to the merit of the fitness value. The elitist strategy guarantees that the best chromosomes will not be dropped by stochastic errors

Step 3. Crossover in reproduction

Traditional local search can improve the quality of a solution by leading it to a local optimum, but it also lacks the capability for further exploring other promising regions. Crossover operators provide a mechanism for the candidate solutions to escape from the local optima. Consider two local optimal chromosomes, each of which contains quality genes at some positions of the chromosome. It is beneficial to explore the uncharted region between local optima by combining the two chromosomes because it is very likely that this region contains better solutions that have not yet been explored.

In order to preserve the solution feasibility for the addressed problem, we propose a modified version of the head–tail crossover (HTC) [20]. We illustrate the HTC by the following example. Let $g_a = (1, 3, 2, 4, 6, 5, 7)$ and$g_b = (2, 7, 5, 4, 3, 1, 6)$ be the parent chromosomes. Denote the offspring to be generated by g_c. We define the head of a chromosome as its first gene and the tail the last gene. The HTC crossover will select the genes in g_a from head to tail and the genes in g_b from tail to head, and combine them together to form the offspring g_c. If the selected gene has been contained in the offspring, the HTC would discard the selection and proceed to the next gene in the respective order. This crossover combination process can not only avoid an infeasible solution but also retain the basic building blocks within the parent chromosomes. The algorithm of the HTC crossover operator is given in the following pseudo codes.

Algorithm: head–tail crossover

```
Parent chromosomes: g_a = (a_0, a_1, a_2, ..., a_{n-1}), g_b = (b_0, b_1, b_2, ..., b_{n-1});
Offspring chromosome: g_c = (c_0, c_1, c_2, ..., 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 < n
      begin
        if (round mod 2) = 0 then
          while g_a[a_index] has existed in g_c
            a_index++
          g_c[c_head] = g_a[a_index]
          a_index++
          c_head++
        else
          while g_b[b_index] has existed in g_c
            b_index−−
          g_c[c_tail] = g_b[b_index]
          b_index−−
          c_tail−−
          round++
      end
  end
```

Step 4. Mutation by local search

The mutation operator commonly used in the design of genetic algorithms plays the role of diversification strategy which may enhance the diversity of the current population and avoids premature convergence. In our proposed hybrid method, however, the mutation operator is implemented using the 2-Opt as aforementioned in local search and tabu search. The 2-Opt mutation operator leads the solutions to local optima which will be combined in the next generation by the head–tail crossover for exploring other promising regions. As the design of crossover and mutation operators is different from that of traditional genetic algorithms, the setting of parameter values, such as the crossover rate and the mutation rate, will be determined based on preliminary experiments.

4.4. Iterated local search

Iterated local search (ILS), probably first due to Stützle [26], is a simple and powerful meta-heuristic that iteratively applies local search to improve on the current solution. It has been applied to flow shop scheduling problems [26]. The motivation for designing ILS is based on the observation that local search tends to get trapped in local optima. Two major mechanisms in ILS, i.e. *perturbation* and *acceptance* criterion, are deployed to resolve this problem. Instead of restarting local search with a random initial solution when the quality of the current solution cannot be improved further, perturbation is introduced to assist local search escape from the local optimum by moving to a new neighbor solution which is then led to another local optimum by the local search process. The acceptance criterion is used to determine which of the local optimal solutions should be retained. The perturbation and acceptance procedures are iterated until the termination condition is satisfied. The following are the pseudo codes of the ILS algorithm. For detail introduction, the reader is referred to Lourenço et al. [23].

```
Procedure Iterated Local Search
  Let S_0 be an initial solution.
  Let S* be the local optimum solution obtained by applying a local search process
on S_0., i.e., S* = LocalSearch (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
```

Next, we describe the detail implementation of all operators used in the ILS algorithm to tackle the TSP-WOCT.

4.4.1. Initial solution

The initial solution is constructed by the greedy method, same as that used in the approximation algorithms noted in previous sections

4.4.2. Local search

In general, any local search algorithm can be used, but the choice will affect the solution quality obtained and the computation time required by the ILS algorithm. Preliminary experimental results manifest that 2-Opt is better than 2-Exchange, so we use 2-Opt to generate neighbor solutions in local search. For large-scale problems, the computing time for applying local search is not acceptable. To handle this difficulty, we propose a simplified version called the best-improvement local search algorithm. First, an edge (or, a pair of nodes) is randomly chosen. We examine all possibilities for swapping with this edge and if one or more better neighbors are found, the one with the greatest improvement is retained.

4.4.3. Perturbation

The adoption of perturbation is important to iterated local search since it enables the search process to leave the local optimum and find a good neighbor as the initial solution to be led to another local optimum that is better than the previous one. For the studied problem, perturbation is a simple modification to the solution structure. It performs two instances of swap-move and one instance of 2-Exchange. The swap-move swaps the jobs at positions i and $i+1$, where i is randomly selected.

4.4.4. Acceptance criterion

We consider the standard acceptance criterion that the ILS algorithm only accepts a local optimum that is better than the one obtained in the previous iteration.

5. Computational experiments

This section is dedicated to appraisal on 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 2.8 GHz CPU and 512 MB memory running Microsoft Windows XP. The test data were obtained by randomly generating points with real value coordinates on a 100×100 square plane. The setup times between jobs are thereby defined as the Euclidean distance (rounded to integer), and thus the setup times between two jobs are symmetric. Euclidean space is adopted to reflect the application origin of satellite imaging scheduling.

To analyze the efficiency of our binary integer program, we use CPLEX to implement the two formulations *IP_Latency_TSP* and *BIP_Latency_TSP* of the deliveryman problem presented in Section 2. For each problem size, five independent test instances were randomly generated. The results are shown in Table 1. The row entitled "optimal" indicates the number of instances for which optimal solutions were successfully found, out of the five independent runs. The row entitled "Avg Time" gives the average execution time over the five instances. It is clear that the binary formulation took a shorter execution time than the integer program. Although the binary integer program is computationally faster than the integer program, we do not claim its absolute superiority. As mentioned in Section 2, the integer program inherits good structures to facilitate the deployment of other research techniques. Another merit of the *BIP_Latency_TSP* model is its applicability for modeling the studied model involving batch deliveries.

The second part is to investigate the efficiency of the dynamic programming algorithm and the effectiveness of the proposed approximation approaches. The number of orders was a given constant and the weight of each order w_i was randomly generated from the uniform interval [1,10]. All jobs were randomly assigned to the orders.

In the computational study, we used various numbers of jobs$n$ and various numbers of orders$K$. One hundred instances were generated for each combination of$n$ and K, and the average performance from these one hundred independent runs was presented as the computational result. The experiment analysis basically consists of three parts. The first part reports the relative performance between the dynamic programming algorithm and all the tested approximation methods. Two performance indices are used: the elapsed running time and the relative error. The relative error is defined as $100\% \times (apprx\text{-}opt)/apprx$, where *opt* and *apprx* respectively denote the objective values of the optimal solution and an approximate solution [3]. Due to the difficulty for the dynamic programming algorithm to report optimal solutions of large instances within a reasonable time, the second and the third parts only consist of the performance comparisons among the three meta-heuristic approaches. Numerical statistics are summarized in Tables 2–9.

Small-scale instances include the combinations of $n = 10, 14, 18, 20, 22$, or 24 and $K = 3$ or 5. The computational results are shown in Tables 2 and 3 with $K = 3$ and 5, respectively. The statistics include the average running times the different algorithms required to solve each instance. Since the dynamic programming algorithm provided optimal solutions for these small-scaled problems, we can compute the hit frequency and the error ratio of approximate solutions with reference to the optimal ones as depicted in the tables. We observe that LS-2EX and LS-2OPT tend to be easily trapped in the local optima

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

n	8		10		12		14		16		18		20		22	
	IP	BIP	IP	BIP	IP	BIP	IP	BIP	IP	BIP	IP	BIP	IP	BIP	IP	BIP
Optimal	5	5	5	5	0	5	0	5	0	5	0	5	0	5	0	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, number of orders = 3.

n	DP	LS-2EX			LS-2OPT			TS-2OPT			TS-2EX			GA			ILS		
	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	3.7	0.00	52	1.7	3.86	100	0.0	3.61	100	0.0	0.02	99	0.0	0.05	99	0.0
14	0.04	0.00	15	5.7	0.00	35	2.3	4.26	100	0.0	3.88	89	0.2	0.08	99	0.0	0.15	97	0.0
18	1.87	0.01	10	8.1	0.01	24	3.7	4.46	100	0.0	4.07	65	0.5	0.26	86	0.1	0.33	88	0.2
20	12.37	0.02	2	7.6	0.01	10	4.8	4.65	99	0.0	4.27	43	0.8	0.46	85	0.1	0.46	86	0.1
22	76.19	0.01	3	9.4	0.01	13	4.9	4.94	98	0.0	4.54	32	1.3	0.64	82	0.1	0.60	79	0.2
24	386.91	0.01	1	9.8	0.01	0	6.5	5.02	94	0.0	4.61	17	2.0	1.08	65	0.3	0.85	72	0.3

Table 3
Small size, number of orders = 5.

n	DP	LS-2EX			LS-2OPT			TS-2OPT			TS-2EX			GA			ILS		
	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	4.0	0.00	53	1.5	3.92	100	0.0	3.67	100	0.0	0.02	99	0.0	0.06	99	0.0
14	0.04	0.00	13	5.6	0.00	32	2.3	4.37	100	0.0	4.01	94	0.1	0.10	98	0.0	0.17	94	0.1
18	1.86	0.01	2	8.6	0.01	11	4.5	4.72	100	0.0	4.32	61	0.6	0.35	87	0.1	0.38	91	0.1
20	12.08	0.02	4	7.9	0.01	5	5.2	4.78	100	0.0	4.39	54	0.5	0.58	91	0.0	0.56	86	0.2
22	75.78	0.01	4	5.9	0.01	6	9.3	5.27	100	0.0	4.80	24	1.4	0.97	70	0.2	0.74	77	0.3
24	395.02	0.01	1	9.7	0.01	3	6.2	5.00	89	0.0	4.61	25	1.6	1.52	61	0.3	0.91	72	0.3

Table 4
Medium size, number of orders = 3.

n	TS-2OPT			GA			ILS		
	Time	Win	Deviation (%)	Time	Win	Deviation (%)	Time	Win	Deviation (%)
30	5.17	72	0.13	3.27	52	0.42	1.58	62	0.49
40	5.84	37	0.67	15.05	25	0.76	3.80	61	0.61
50	6.48	22	1.25	20.47	16	1.52	7.68	66	0.43
60	7.23	14	2.12	34.96	10	2.28	13.39	77	0.22

as compared to TS-2OPT, TS-2EX, GA and ILS. Consequently, the average hit frequencies and error ratios of LS-2EX and LS-2OPT appear to be inferior to those of other approximation methods. In addition, if examining the computational results in more detail, we may see the superiority of the 2-Opt strategy over the 2-Exchange strategy for constructing the neighborhood structures in both the local search and tabu search approaches. This phenomenon is also evident in all of the other experiments conducted in this research. It is worthwhile to further study the comparative performance of the two neighborhood strategies applied in other contexts. In the remainder of the experiment, we used the TS_2OPT strategy as the prevailing tabu search approach.

The numerical results in Tables 2 and 3 also evince that TS_2OPT has the highest hit frequency and the lowest error ratio among all meta-heuristics for small-scale problems. TS_2OPT could report optimal solutions for most of the test cases, thus achieving negligible error ratios. As the number of jobs (n) exceeds 18, the dynamic programming algorithm requires a computation time significantly longer than all meta-heuristics. In particular, the error ratios resulted by TS-2OPT, GA, and ILS are below 0.3 percents for all tested small size instances. This implies that the number of orders (K) does not substantially affect the performance of these algorithms when the problem size is small.

Tables 4–6 exhibit the performances obtained through medium-scale instances, where the number of jobs (n) is set to 30, 40, 50, or 60, and the number of orders (K) = 3, 6, or 15. Since the dynamic programming algorithm failed to produce optimal

Table 5
Medium size, number of orders = 6.

n	TS-2OPT			GA			ILS		
	Time	Win	Deviation (%)	Time	Win	Deviation (%)	Time	Win	Deviation (%)
30	5.55	60	0.22	6.08	32	0.46	1.90	55	0.74
40	6.46	27	1.04	18.76	23	1.16	4.62	55	0.71
50	6.82	23	1.64	34.93	16	1.89	9.22	63	0.48
60	7.58	14	3.13	60.92	9	3.915	15.99	84	0.25

Table 6
Medium size, number of orders = 15.

n	TS-2OPT			GA			ILS		
	Time	Win	Deviation (%)	Time	Win	Deviation (%)	Time	Win	Deviation (%)
30	6.23	69	0.15	4.88	24	0.77	2.57	53	0.62
40	7.14	28	1.02	12.82	18	1.76	6.16	58	0.53
50	7.88	14	2.03	27.25	14	2.82	12.69	71	0.46
60	8.89	10	2.87	62.97	8	3.39	21.74	81	0.34

Table 7
Large size, number of orders = 6.

n	TS-2OPT			GA			ILS		
	Time	Win	Deviation (%)	Time	Win	Deviation (%)	Time	Win	Deviation (%)
70	8.51	7	4.03	35.34	2	5.15	26.78	91	0.14
80	9.15	7	4.68	64.67	6	4.93	40.61	88	0.16
90	10.1	0	5.77	53.92	1	6.10	56.77	99	0.01
100	10.84	3	6.07	82.32	2	6.27	80.65	95	0.04

Table 8
Large size, number of orders = 15.

n	TS-2OPT			GA			ILS		
	Time	Win	Deviation (%)	Time	Win	Deviation (%)	Time	Win	Deviation (%)
70	9.37	9	4.04	66.28	1	5.47	33.39	90	0.11
80	10.18	3	5.52	100.77	2	6.28	50.42	94	0.11
90	11.07	3	6.33	100.36	1	7.24	72.60	96	0.06
100	12.15	0	6.95	93.07	0	7.52	100.88	100	0.00

Table 9
Large size, number of orders = 25.

n	TS-2OPT			GA			ILS		
	Time	Win	Deviation (%)	Time	Win	Deviation (%)	Time	Win	Deviation (%)
70	10.29	7	4.59	82.93	2	5.75	41.23	91	0.13
80	11.38	5	6.61	133.82	0	5.75	61.22	95	0.02
90	11.95	1	9.31	163.79	0	6.57	86.86	99	0.00
100	12.77	0	10.55	197.84	1	6.93	21.74	99	0.00

solutions within a reasonable time and the solution quality obtained by the local search methods (LS-2EX and LS-2OPT) is not acceptable, only the performances among the TS-2OPT, GA, and ILS are examined. The columns entitled "win" in these tables indicate the number of tested instances for which the corresponding method provides the best approximate solution among all the competitors. The columns entitled "deviation" summarize the relative error ratios produced by each individual method by replacing the reference to the optimal solution with the best solution generated by the winner approach indicated in the "win" column. Some observations for various values of n and K are summarized.

1. Observations on different numbers of jobs, n: The values of "win" decrease as n increases for TS-2OPT and GA; however, ILS shows an ascending value of "win" as n increases. This indicates that a higher probability exists for ILS to play the role of winner among the three approaches when tackling medium-scale problems with larger values of n. More precisely, TS-2OPT was able to produce the best approximate solutions for most instances with $n = 30$. ILS becomes the leading method when $n \geqslant 40$.
2. Observations on different numbers of orders, K: The running times and deviations for all of the three competing methods generally grow along with the increase of the number of orders. GA relatively provided a less number of "win" solutions as the value of K increases, implying that GA performs comparatively worse with larger K when compared with TS-2OPT and ILS.

Tables 7–9 summarize the statistics for the large-scale cases. We specified four different numbers of jobs ($n = 70$, 80, 90, or 100) with three different numbers of orders ($K = 6$, 15, or 25). It is clear that ILS outperformed TS-2OPT and GA because it produced the largest number of "win" solutions when coping with large-scale instances. GA not only failed to beat the other two methods in most of the large-scaled problem cases, but also required a much longer execution time. TS-2OPT seems to be the most efficient method in the aspect of required computing time. However, TS-2OPT failed to produce more "win" solutions even when the number of iterations was increased to 100,000, implying that TS-2OPT might not be able to escape from local optima when the problem size increases. Therefore, ILS could a viable method when considering the trade off between solution quality and elapsed running time.

6. Conclusions

This paper considered a variant of the latency TSP incorporating order delivery. We first gave a binary integer program that is different from the integer program known in the literature. The binary model runs faster than the existing integer one. 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 identified. By the complexity nature of the problem, three meta-heuristics, including a local search, a tabu search and a genetic algorithm, were designed for producing approximate solutions. From the computational statistics, it is clear that the iterated local search outperforms all other approximation approaches in terms of solution quality. The dominance becomes more significant when the number of jobs increases. If we consider the time elapsed for producing solutions, tabu search dominates the other two meta-heuristic approaches. Comparing different neighborhood structures, we find that 2-Opt is better than 2-Exchange. Genetic algorithm seems to be inferior to other meta-heuristic approaches in all settings. We need to emphasize that the experiments are not designed and conducted to claim the superiority of one method over others for the studied problem, 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 derivation of tighter lower bounds also provides theoretical insights into the studied problem and gives underestimates of optimal objective values for evaluating approximation approaches. The concept of order delivery suggests considerable room for further research on other subjects when aggregation of individual entities (jobs, nodes, elements) is in effect.

Acknowledgements

The authors are grateful for the reviewers' comments that have improved the presentation of earlier versions of this paper. This research was partially supported by the National Science Council of Taiwan under project grants NSC 100-2410-H-009 -015 -MY2 and NSC 101-2410-H-260 -004 -MY2.

References

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, N. Papakostantinou, The complexity of the traveling repairman problem, Theor. Inf. Appl. 20 (1986) 79–87.
[2] A. Archer, D.P. Williamson, Faster approximation algorithms for the minimum latency problem, in: The 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 88–96.
[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability, Springer, New York, 1999.
[4] N. Bianchessi, J.-F. Cordeau, J. Desrosiers, G. Laporte, V. Raymond, A heuristic for the multi-satellite, multi-orbit and multi-user management of earth observation satellites, Eur. J. Oper. Res. 177 (2007) 750–762.
[5] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, M. Sudan, The minimum latency problem, in: The 26th ACM Symposium on the Theory of Computing, 1994, pp. 163–171.
[6] T.C.E. Cheng, V.S. Gordon, Batch delivery scheduling on a single machine, J. Oper. Res. Soc. 45 (1994) 1211–1215.
[7] T.C.E. Cheng, V.S. Gordon, M.Y. Kovalyov, Single machine scheduling with batch deliveries, Eur. J. Oper. Res. 94 (1996) 277–283.
[8] T.C.E. Cheng, H.G. Kahlbacher, Scheduling with delivery and earliness penalties, Asia-Pac. J. Oper. Res. 10 (1993) 145–152.
[9] T.C.E. Cheng, M.Y. Kovalyov, B.M.T. Lin, Single machine scheduling to minimize batch delivery and job earliness penalties, SIAM J. Optim. 7 (1997) 547–559.
[10] X. Dong, H. Huang, P. Chen, An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion, Comput. Oper. Res. 36 (2009) 1664–1669.
[11] M. Fischetti, G. Laporte, S. Martello, The delivery man problem and cumulative matroids, Oper. Res. 41 (1993) 1055–1064.

[12] F. Glover, Tabu search – part I, ORSA J. Comput. 1 (1989) 190–206.
[13] F. Glover, Tabu search – part II, ORSA J. Comput. 2 (1990) 4–32.
[14] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, MA, 1997.
[15] A. Garcia, P. Jodr'a, J. Tejel, A note on the traveling repairmen problem, Networks 40 (2002) 27–31.
[16] M. Goemans, J. Kleinberg, An improved approximation ratio for the minimum latency problem, Math. Program. 82 (1998) 111–124.
[17] G. Gutin, A.P. Punnen, The Traveling Salesman Problem and Its Variations, Springer, New York, 2007.
[18] J.H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, 1992.
[19] M. Ji, Y. He, T.C.E. Cheng, Batch delivery scheduling with batch delivery cost on a single machine, Eur. J. Oper. Res. 176 (2007) 745–755.
[20] K.Y. Lee, F.F. Yang, Optimal reactive power planning using evolutionary algorithms: a comparative study for evolutionary programming, evolutionary strategy, genetic algorithm, and linear programming, IEEE Trans. Power Syst. 13 (1998) 101–108.
[21] M. Lemaitre, G. Verfaillie, F. Jouhaud, J.-M. Lachiver, N. Bataille, Selecting and scheduling observations of agile satellites, Aerosp. Sci. Technol 6 (2002) 367–381.
[22] W.C. Lin, D.Y. Liao, C.Y. Liu, Y.Y. Lee, Daily imaging scheduling of an earth observation satellite, IEEE Trans. Syst. Man Cybern. Part A Syst. Humans 35 (2005) 213–223.
[23] H. Lourenço, O. Martin, T. Stützle, A gentle introduction to iterated local search, in: Proceedings of MIC'2001 – The 4th Metaheuristics International Conference, Porto, Portugal, 2001.
[24] E. Minieka, The delivery man problem on a tree network, Ann. Oper. Res. 18 (1989) 261–266.
[25] M. Pinedo, Scheduling Theory, Algorithms and Systems, 4th ed., Springer, New York, 2012.
[26] T. Stützle, Applying Iterated Local Search to the Permutation Flow Shop Problem, Technical Report, AIDA-98-04, FG Intellektik, TU Darmstadt, 1998.
[27] G. Syswerda, J. Pamucci, The application of genetic algorithms to resource scheduling, in: The 4th International Conference on Genetic Algorithms, San Diego, California, USA, January 1991, pp. 501–508.
[28] J. Tian, T.C.E. Cheng, C.T. Ng, J.J. Yuan, An improved online algorithm for single parallel-batch machine scheduling with delivery times, Discrete Appl. Math. 160 (2012) 1191–1210.
[29] G. Wang, T.C.E. Cheng, Parallel machine scheduling with batch delivery costs, Int. J. Prod. Econ. 68 (2000) 177–183.
[30] P. Wang, G. Reinelt, P. Gao, Y. Tan, A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation, Comput. Ind. Eng. 61 (2011) 322–335.
[31] W.J. Wolfe, Spacebased Resource Planning: Phase I. Summary Report for Houghes Information Technology Systems research grant #SG-258799SJP, Aurora, Colorado, USA, January 2004.
[32] W.J. Wolfe, Spacebased Resource Planning: Phase II. Summary Report for Houghes Information Technology Systems research grant #SG-258799SJP, Aurora, Colorado, USA, January 2005.
[33] W.J. Wolfe, S.E. Sorensen, Three scheduling algorithms applied to the earth observing systems domain, Manage. Sci. 46 (2000) 148–166.
[34] B.Y. Wu, Polynomial time algorithms for some minimum latency problems, Inf. Process. Lett. 75 (2000) 225–229.
[35] B.Y. Wu, Z.N. Huang, F.-J. Zhan, Exact algorithms for the minimum latency problem, Inf. Process. Lett. 92 (2004) 303–309.
[36] X. Yang, Scheduling with generalized batch delivery dates and earliness penalties, IIE Trans. 32 (2000) 735–741.
[37] Y.Q. Yin, T.C.E. Cheng, S.R. Cheng, C.C. Wu, Single-machine batch delivery scheduling with an assignable common due date and controllable processing times, Comput. Ind. Eng. 65 (2013) 652–662.
[38] Y.Q. Yin, T.C.E. Cheng, C.J. Hsu, C.C. Wu, Single-machine batch delivery scheduling with an assignable common due window, Omega 41 (2013) 216–225.
[39] Y.Q. Yin, T.C.E. Cheng, D.H. Xu, C.C. Wu, Common due date assignment and scheduling with a rate-modifying activity to minimize the due date, earliness, tardiness, holding, and batch delivery costs, Comput. Ind. Eng. 63 (2012) 223–234.