# A Tree-Based Tabu Search Algorithm for the Manpower Allocation Problem with Time Windows and Job-Teaming Constraints

**Yilin Cai**[1] and **Zizhen Zhang**[2] and **Songshan Guo**[1] and **Hu Qin**[3] and **Andrew Lim**[2]
[1]Department of Computer Science, Sun Yat-Sen University
[2]Department of Management Sciences, City University of Hong Kong
[3]School of Management, Huazhong University of Science and Technology
{cylixstar, zhangzizhen}@gmail.com

## Abstract

This paper investigates the manpower allocation problem with time windows and job-teaming constraints (MAPTWTC), a practical scheduling and routing problem that tries to synchronize workers' schedules to complete all tasks. We first provide an integer programming model for the problem and discuss its properties. Next, we show that tree data structure can be used to represent the MAPTWTC solutions, and its optimal solution can be obtained from one of trees by solving a minimum cost flow model for each worker type. Consequently, we develop for the problem a novel tabu search algorithm employing search operators based on the tree data structure. Finally, we prove the effectiveness of the tabu search algorithm by computational experiments on two sets of instances.

## 1 Introduction

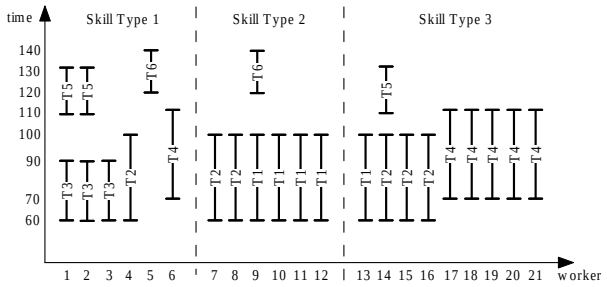
This paper investigates the manpower allocation problem with time windows and job-teaming constraints (MAPTWTC), which is a combinatorial optimization problem in the realm of scheduling and routing. The MAPTWTC tries to assign a set of tasks to a set of workers with different skills or qualifications so as to minimize the total cost. Each task is characterized by a location, a required team of workers, a processing time, and a time window. All workers are initiated at the depot and to be dispatched to fulfill a series of tasks. After finishing a task, the worker will either move to fulfill his/her next task or return to the depot. The workers are allowed to arrive at the task location at different times, but a task cannot be started until all required workers are available, which is called the *synchronization constraint*.

Table 1 shows an MAPTWTC instance involving 6 tasks and 3 worker types; this instance was constructed by [Li *et al.*, 2005]. In this table, the column "Worker" specifies the number of workers of each type required; the column "PT" gives the processing time of each task; the column "Successor" presents the possible successor tasks of each task; and the column "Distance" gives the traveling distance from the depot to the task location in the first entry, and the traveling distances from the task location in the first column to task locations in the fifth column. Figure 1 illustrates a feasible solution to the instance, where 6 type 1 workers, 6 type 2 workers and 9 type 3 workers are required.

Table 1: An MAPTWTC instance.

| Task | Worker | Time window | PT | Successor | Distance |
|---|---|---|---|---|---|
| T1 | (0, 4, 1) | [60, 200] | 40 | T4,T6 | (35; 10, 20) |
| T2 | (1, 2, 3) | [60, 110] | 40 | T5 | (10; 10) |
| T3 | (3, 0, 0) | [60, 100] | 30 | T5 | (40; 5) |
| T4 | (1, 0, 5) | [70, 150] | 40 | T1 | (30; 10) |
| T5 | (2, 0, 1) | [45, 150] | 20 | - | (20; -) |
| T6 | (1, 1, 0) | [110, 200] | 20 | - | (50; -) |

Figure 1: A feasible solution to the MAPTWTC instance.




The MAPTWTC was originated from the Port of Singapore and was first mentioned in literature by [Lim *et al.*, 2004] and [Li *et al.*, 2005]. The authors implemented simulated annealing algorithms to seek near-optimal solutions for the MAPTWTC. [Dohn *et al.*, 2009] studied the MAPTWTC from an application of scheduling ground handling tasks in Europe's major airports. They proposed a branch-and-price algorithm to exactly solve the problem instances of small- and medium-size. If the synchronization constraint is relaxed by allowing workers to start the tasks independently, the MAPTWTC reduces to the well-studied vehicle routing problem with time windows (VRPTW) [Solomon, 1987]. Synchronizing the scheduling subjects is a "hot" topic in recent years; we refer the reader to [Drexl, 2012; Bredström and Rönnqvist, 2008] for more applications. The MAPTWTC can also be viewed as a type of batching problem occurring in the area of machine scheduling [Brucker, 2007]. If each task is a *job batch* (a batch is a group of jobs which must be processed jointly), the MAPTWTC model can be used to schedule the machines to complete all batches.

In this study, we first formulate the MAPTWTC into a network flow model. We observe that the MAPTWTC solutions

can be represented by trees, one of which must correspond to one of the optimal solutions. Therefore, we next propose a tabu search algorithm that seeks the tree leading to the optimal MAPTWTC solution. A tree can be transformed to an MAPTWTC solution by solving a couple of minimum cost flow models. Finally, we show by computational experiments that our tree-based tabu search algorithm is capable of producing high-quality MAPTWTC solutions.

## 2 Mathematical Formulation

The MAPTWTC is defined on a complete graph $G = (V, E)$, where $V = \{0, 1, \ldots, n\}$ is the vertex set and $E = \{(i, j)|i, j \in V, i \neq j\}$ is the edge set. Vertex 0 represents the depot and $V_T = \{1, \ldots, n\}$ denotes $n$ task locations. Each edge $(i, j) \in E$ has a non-negative traveling distance $d_{i,j}$, where the distance matrix $[d_{i,j}]$ satisfies the triangle inequality. Each vertex $i \in V_T$ is associated with a time window $[s_i, e_i]$ specifying the earliest and latest service starting times of task $i$, and a non-negative processing time $p_i$. For notational convenience, we set $s_0 = 0$, $e_0 = +\infty$ and $p_0 = 0$ for the depot. A worker can carry out task $j$ immediately after completing task $i$ only if the earliest service starting time of task $i$ plus $d_{i,j}$ does not exceed $e_j$. Additionally, if task $j$ is not a successor of task $i$, we can set their distance $d_{i,j}$ to be $+\infty$. Thus, the edge set $E$ can be updated by $E = \{(i, j)|s_i + p_i + d_{i,j} \leq e_j, i, j \in V\}$. We further denote by $E_T = \{(i, j)|(i, j) \in E, i, j \in V_T\}$ the set of edges between any two distinct task locations. Moreover, we use $\delta^-(i) = \{j|(j, i) \in E\}$ and $\delta^+(i) = \{j|(i, j) \in E\}$ to represent the sets of vertex $i$'s immediate predecessors and successors, respectively.

All workers are classified into $K$ types according to their skills or qualifications. We use $v_{k,i}$ $(1 \leq k \leq K)$ to represent the number of type $k$ workers required by task $i$. The goal of the MAPTWTC is to minimize the total cost incurred by the number of workers ($W$) employed and the total distance traveled ($D$). The objective function can be written as: $\alpha W + \beta D$, where $\alpha$ and $\beta$ are the labor cost per worker and the traveling cost per unit distance, respectively.

The MAPTWTC has been shown to be NP-hard in the strong sense by [Li *et al.*, 2005]. The authors also presented an integer programming (IP) model for the problem. However, we find that their IP model is very inefficient as an upper bound on the number of workers of each type has to be set in advance and each worker is associated with $|E|$ binary variables. As a result, this IP model contains a huge number of binary variables. We propose a more compact IP model for the MAPTWTC, which has a network flow structure. To present our model, we define the following three sets of decision variables, where $i, j \in V$ and $1 \leq k \leq K$.

**Decision variables:**

$w_i$: the service starting time of task $i$;

$x_{i,j}$: a binary variable which is equal to 1 if workers are allowed to move from vertex $i$ to vertex $j$, and 0 otherwise;

$y_{k,i,j}$: the number of type $k$ workers dispatched to task $j$ immediately after finishing task $i$, i.e., $y_{k,i,j}$ is the flow of type $k$ workers on edge $(i, j)$.

Using the above variables, the IP model for the MAPTWTC is as follows:

$$(IP)\ \min\ \alpha \sum_{1\leq k\leq K} \sum_{j\in V_T} y_{k,0,j} + \beta \sum_{1\leq k\leq K} \sum_{i\in V} \sum_{j\in\delta^+(i)} d_{i,j} y_{k,i,j} \tag{1}$$

$$\text{s.t.} \sum_{j\in\delta^-(i)} y_{k,j,i} = v_{k,i},\ \forall\, i \in V_T,\ 1 \leq k \leq K \tag{2}$$

$$\sum_{j\in\delta^-(i)} y_{k,j,i} - \sum_{j\in\delta^+(i)} y_{k,i,j} = 0,\ \forall\, i \in V_T,\ 1 \leq k \leq K \tag{3}$$

$$y_{k,i,j} \leq M x_{i,j},\ \forall\, (i, j) \in E,\ 1 \leq k \leq K \tag{4}$$

$$w_i + p_i + d_{i,j} \leq w_j + M(1 - x_{i,j}),\ \forall\, (i, j) \in E_T \tag{5}$$

$$d_{0,j} \leq w_j + M(1 - x_{0,j}),\ \forall\, j \in V_T,\ (0, j) \in E \tag{6}$$

$$s_i \leq w_i \leq e_i,\ \forall\, i \in V_T \tag{7}$$

$$x_{i,j} \in \{0, 1\},\ \forall\, i,\ j \in V \tag{8}$$

$$y_{k,i,j} \geq 0 \text{ and integer},\ \forall\, i,\ j \in V,\ 1 \leq k \leq K \tag{9}$$

where $M$ is a sufficiently large positive number. The objective (1) is to minimize the total cost consisting of labor cost and traveling cost. Constraints (2) guarantee that the demand of workers of each type must be met for each task. Constraints (3) are flow conservation constraints. Constraints (4) ensure that if any flow $y_{k,i,j}$ is positive, the value of $x_{i,j}$ must be equal to 1. That is, $x_{i,j} = 0$ implies that no workers are allowed to traverse edge $(i, j)$. Constraints (5) and (6) state that if workers are allowed to move from task $i$ to task $j$, the earliest possible arrival time at task $j$, calculated by $w_i + p_i + d_{i,j}$, must be less than or equal to its service starting time $w_j$. Constraints (7) are the time window constraints. It is easy to see that the model $(IP)$ has feasible solutions if and only if $d_{0,i} \leq e_i$ for every $i \in V_T$.

The model $(IP)$ can produce optimal solution value for the MAPTWTC, while it does not directly generate a schedule (or a route) for each worker. We can easily convert a feasible MAPTWTC schedule that is comprised of $m$ routes into a feasible solution of the model $(IP)$. Specifically, $y_{k,i,j}$ is set to the number of times edge $(i, j)$ appearing in all routes of type $k$ workers; $x_{i,j}$ is set to 1 if there exists at least one route that traverses edge $(i, j)$, and 0 otherwise; and $w_i$ is set to the earliest service starting time of task $i$, which can be easily derived from the routes of the workers assigned to that task. On the other hand, we can generate a feasible schedule from a feasible solution of the model $(IP)$ (denoted by a vector $(\mathbf{x}, \mathbf{y}, \mathbf{w})$) using the flow decomposition algorithm described in [Ahuja *et al.*, 1993].

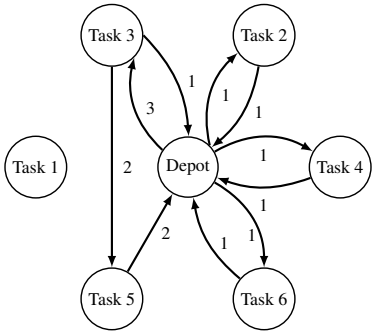
Constraints (3) reveal that the model $(IP)$ includes $K$ circulation problems ([Ahuja *et al.*, 1993], page 7) with additional constraints. According to *Flow Decomposition Theorem* ([Ahuja *et al.*, 1993], page 80), a circulation can be decomposed into a set of directed cycles (i.e., routes). The following theorem states that all these directed cycles must include the depot.

**Theorem 1** *For any feasible solution $(\mathbf{x}, \mathbf{y}, \mathbf{w})$ to the model (IP), if the flow decomposition procedure is applied to the circulation $\mathbf{y}_k$ $(1 \leq k \leq K)$, all generated directed cycles must include the depot.*

Proof. Suppose that there exists a directed cycle $C$ which does not include the depot. We start from any vertex $i$ on $C$ and sequentially visit each vertex on $C$. From Constraints (5), we know the service starting time of each vertex must be larger than that of its predecessors. Hence, we will eventually derive $w_i < w_i$, leading to a contradiction; this implies that each cycle must start from and end at the depot. □

Figure 2 illustrates an example of circulation 1 (the circulation corresponding to the schedule of type 1 workers) in a solution. In this figure, the number adjacent to each edge is the value of $y_{1,i,j}$. By the flow decomposition procedure, we obtained 6 worker schedules: (0-2-0),(0-3-0),(0-4-0),(0-6-0),(0-3-5-0) and (0-3-5-0). Note that the flow decomposition procedure may generate different MAPTWTC solutions, but their objective values must be the same.

Figure 2: The circulation corresponding to the schedule of type 1 workers in a solution.




## 3 The Tree Representation of the Solutions

The way of representing solutions is one of key issues in designing meta-heuristics. Due to the existence of the synchronization constraints, the solution representation for the MAPTWTC is more complicated than that for the VRPTW and some of other VRP variants. In this section, we show that the optimal solution of the MAPTWTC can be obtained from a tree with $n+1$ vertices. An example of using trees to represent the solutions of a traveling salesman problem variant can be found in [Tu *et al.*, 2010].

The idea of using trees to represent the MAPTWTC solutions stems from the following observations. For any feasible routing schedule to the MAPTWTC, we can shift the service starting time of all tasks as early as possible while maintaining the feasibility of the original routing tour of each worker. When no tasks can be shifted, we can identify at least one *critical path* for each task. The properties of all these critical paths can be captured by a minimum spanning tree. The optimal solutions of the problem can be obtained by thoroughly checking all spanning trees, i.e., the search space can be restricted only to all minimum spanning trees.

We define $e(\mathbf{x}) = \{(i,j)|x_{i,j} = 1, (i,j) \in E\}$ as the set of edges that the workers are allowed to traverse when vector variable $\mathbf{x}$ is fixed. Analogously, when vector variable $\mathbf{w}$ is fixed, we can derive another edge set $e(\mathbf{w}) = \{(i,j)|w_i+p_i+d_{i,j} \leq w_j, (i,j) \in E_T\} \cup \{(0,j)|d_{0,j} \leq w_j, (0,j) \in E\})$.

If vector variable $\mathbf{x}$ is fixed, the model (*IP*) can be decomposed into two separate parts. The first part includes $K$ circulation problems (namely objective (1) together with Constraints (3) and (4)) and Constraints (2). This part can be converted into $K$ *minimum cost flow* problems, which will be discussed in detail in Section 4. The second part includes Constraints (5) – (7) that constitute a *system of difference constraints* ([Ahuja *et al.*, 1993], page 103). When proper values are assigned to vector variable $\mathbf{x}$, we can find feasible values for vector variable $\mathbf{w}$ in the system of difference constraints, and obtain a feasible MAPTWTC solution by solving $K$ minimum cost flow problems. Note that the MAPTWTC might not have feasible solutions when $\mathbf{x}$ is set to some values. The vector $\mathbf{x}$ that results in a feasible $\mathbf{w}$ in the system of difference constraints is called feasible. The above observations reveal that the optimal MAPTWTC solution can be obtained from a certain vector $\mathbf{x}$. For any two feasible vectors $\mathbf{x_1}$ and $\mathbf{x_2}$, we have:

**Lemma 1** *For any two feasible vectors* $\mathbf{x_1}$ *and* $\mathbf{x_2}$*, if* $e(\mathbf{x_1}) \subseteq e(\mathbf{x_2})$*, then* $opt_x(\mathbf{x_2}) \leq opt_x(\mathbf{x_1})$*, where* $opt_x(\mathbf{x})$ *is the objective value of the best MAPTWTC solution derived from vector* $\mathbf{x}$*.*

Proof. Both $\mathbf{x_1}$ and $\mathbf{x_2}$ are feasible for the system of difference constraints (5) – (7). Since $\mathbf{x_2}$ provides more choices to worker flows ($\mathbf{y}$) than $\mathbf{x_1}$ (see Constraints (4)), the solution space defined by $\mathbf{x_2}$ is larger than or equal to that defined by $\mathbf{x_1}$. This can directly derive the lemma. □

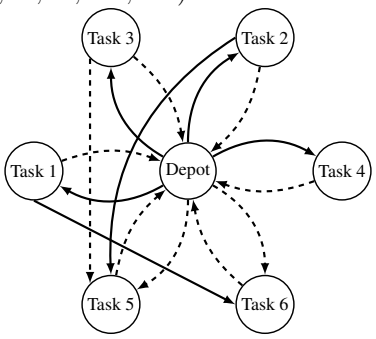
The procedure of computing $opt_x(\mathbf{x})$ will be elaborated in Section 4. We next turn to discuss the property of the model (*IP*) when vector variable $\mathbf{w}$ is fixed. We say vector $\mathbf{w}$ is feasible if $s_i \leq w_i \leq e_i$ for all $i \in V_T$. When $d_{0,i} \leq e_i$ for all $i \in V_T$, a feasible vector $\mathbf{w}$ can lead to a set of feasible MAPTWTC solutions. From Constraints (5) and (6), we know that to make the model (*IP*) feasible, $x_{i,j}$ must be set to 0 if $w_i + p_i + d_{i,j} > w_j$ or $d_{0,j} > w_j$; otherwise, $x_{i,j}$ can be set to either 1 or 0. However, we always set $x_{i,j} = 1$ if it has two choices. This is because by Lemma 1 such setting of $\mathbf{x}$ leads to the best MAPTWTC solution (when $\mathbf{w}$ is fixed). Obviously, $e(\mathbf{w})$ is in fact equivalent to the largest possible $e(\mathbf{x})$ with the given $\mathbf{w}$. Figure 3 depicts a graph $G_w = (V, e(\mathbf{w}))$, where $\mathbf{w}$ corresponds to the service starting times of all tasks in the solution shown in Figure 1, i.e. $\mathbf{w} = (w_1, \ldots, w_6) = (60, 60, 60, 70, 110, 120)$. Similarly, we have the following lemma related to $e(\mathbf{w})$.

**Lemma 2** *For any two feasible vectors* $\mathbf{w_1}$ *and* $\mathbf{w_2}$*, if* $e(\mathbf{w_1}) \subseteq e(\mathbf{w_2})$*, then* $opt_w(\mathbf{w_2}) \leq opt_w(\mathbf{w_1})$*, where* $opt_w(\mathbf{w})$ *is the objective value of the best MAPTWTC solution derived from vector* $\mathbf{w}$*.*

Proof. For a given feasible vector $\mathbf{w}$, there always exists a vector $\mathbf{x}'$ such that $e(\mathbf{w}) = e(\mathbf{x}')$. Moreover, the best objective value $opt_w(\mathbf{w})$ is equal to $opt_x(\mathbf{x}')$. Hence, the statement can be directly deduced from Lemma 1. □

We divide $e(\mathbf{w})$ into two edge sets, namely *critical* and *non-critical* edges. An edge $(i,j) \in e(\mathbf{w})$ is called critical if it satisfies at least one of the following conditions: (1) $w_j = w_i + p_i + d_{i,j}$ for $i, j \in V_T$, (2) $w_j = d_{i,j}$ for $i = 0, j \in V_T$, (3) $w_j = s_j$ for $i = 0, j \in V_T$; otherwise, it is called non-critical. In Figure 3, the critical and non-critical edges

Figure 3: An example of the graph $G_w = (V, e(\mathbf{w}))$, where $\mathbf{w} = (60, 60, 60, 70, 110, 120)$.




are denoted by solid and dash lines, respectively. We define $\Delta_j(\mathbf{w}), j \in V_T$ as:

$$\Delta_j(\mathbf{w}) = \min \begin{cases} w_j - (w_i + p_i + d_{i,j}), \forall\, i \in V_T, (i,j) \in e(\mathbf{w}) \\ w_j - d_{0,j} \\ w_j - s_j \end{cases} \tag{10}$$

Obviously, a feasible vector $\mathbf{w}$ must have $\Delta_j(\mathbf{w}) \geq 0$ for all $j \in V_T$. The vector $\mathbf{w}$ is called *tight* if and only if $\Delta_j(\mathbf{w}) = 0,\ \forall\ j \in V_T$, i.e., there exists at least one critical incoming edge to each vertex $j$. For instance, the vector $\mathbf{w}$ shown in Figure 3 is tight. The following theorem shows that we can always obtain a tight vector $\mathbf{w^t}$ from a feasible vector $\mathbf{w}$, where $opt_w(\mathbf{w^t})$ must be less than or equal to $opt_w(\mathbf{w})$.

**Theorem 2** *There always exists a tight vector* $\mathbf{w^t}$ *for any feasible vector* $\mathbf{w}$ *such that* $opt_w(\mathbf{w^t}) \leq opt_w(\mathbf{w})$.

Proof. We construct a system of difference constraints as follows.

$$\bar{w}_j - \bar{w}_i \geq p_i + d_{i,j},\ \forall\, i,j \in V_T \text{ and } (i,j) \in e(\mathbf{w}) \tag{11}$$
$$\bar{w}_j - 0 \geq d_{0,j},\ \forall\, j \in V_T \text{ and } (0,j) \in e(\mathbf{w}) \tag{12}$$
$$\bar{w}_i - 0 \geq s_i,\ \forall\, i \in V_T \tag{13}$$
$$0 - \bar{w}_i \geq -e_i,\ \forall\, i \in V_T \tag{14}$$

where $\bar{\mathbf{w}} = (\bar{w}_1, \ldots, \bar{w}_n)$ is a vector variable. Since $\mathbf{w}$ is a feasible solution to the above system, this system of difference constraints must have feasible solutions. We can apply a shortest path algorithm, e.g., Bellman-Ford algorithm, to identify the shortest traveling time (denoted by ${w_i}^t$) from the depot to each vertex $i$ on the associated *constraint graph*. Meanwhile, by the shortest path algorithm we can also obtain a *shortest path tree* (note that the shortest path tree may not be unique). According to the property of the shortest path tree, we can easily observe that $\mathbf{w^t}$ is tight, where $\mathbf{w^t} = (w_1^t, \ldots, w_n^t)$. Since $\mathbf{w^t}$ is a feasible solution to the system of difference constraints, all edges $(i,j) \in e(\mathbf{w})$ must also be included in the edge set $e(\mathbf{w^t})$, leading to $e(\mathbf{w}) \subseteq e(\mathbf{w^t})$. By Lemma 2 we therefore have $opt_w(\mathbf{w^t}) \leq opt_w(\mathbf{w})$. □

Suppose that $\mathbf{T}$ is the set of all trees spanning the graph $G$. According to the proof of Theorem 2, we can define a function $w2t : \mathbf{R}^n \rightarrow \mathbf{T}$ to express the construction of a spanning tree (i.e., shortest path tree) from a vector $\mathbf{w}$. On the other hand, we can also derive a vector $\mathbf{w}$ from a tree $\mathbf{t} \in \mathbf{T}$ using a function $t2w : \mathbf{T} \rightarrow \mathbf{R}^n$. For a given tree $\mathbf{t}$, its corresponding vector $\mathbf{w} = t2w(\mathbf{t})$ can be obtained by:

$$w_j = \max \begin{cases} s_j \\ d_{0,j},\ father(j) = 0 \\ w_i + p_i + d_{i,j}, i = father(j) \text{ and } i \neq 0 \end{cases} \tag{15}$$

where $father(j)$ represents the father of vertex $j$ on tree $\mathbf{t}$. We say $\mathbf{t}$ is feasible if its corresponding vector $\mathbf{w}$ is feasible, namely $s_j \leq w_j \leq e_j$ for all $j \in V_T$.

The relationship between the functions $w2t$ and $t2w$ is as follows. If the tight vector $\mathbf{w^t}$ is obtained from a feasible vector $\mathbf{w}$ by the shortest path algorithm, then we must have $t2w(w2t(\mathbf{w})) = \mathbf{w^t}$. As every feasible $\mathbf{w}$ is associated with a feasible tree, we can achieve the optimal solution by identifying the best spanning tree on graph $G$; this reduces the search space of the original problem significantly. With a given tree $\mathbf{t}$, we can first apply the function $t2w$ to get the corresponding tight vector $\mathbf{w^t}$ and then generate the best MAPTWTC solution with regard to $\mathbf{w^t}$ by solving $K$ minimum cost models (see Section 4). We can straightforwardly present the following theorem.

**Theorem 3** *If the MAPTWTC has feasible solutions, then there must exist a feasible tree* $\mathbf{t} \in \mathbf{T}$ *such that* $opt_w(t2w(\mathbf{t}))$ *is the optimal solution value of the MAPTWTC.*

## 4 Minimum Cost Flow Model

For a given feasible tree $\mathbf{t}$, we first obtain the vector $\mathbf{w}$ using function $t2w(\mathbf{t})$. Next, we identify an $\mathbf{x}$ with the largest possible $e(\mathbf{x})$ so that the tree can be evaluated by $opt_w(t2w(\mathbf{t})) = opt_x(\mathbf{x})$. In this section, we show how to calculate $opt_x(\mathbf{x})$ by solving minimum cost flow models. Recall that when vector variable $\mathbf{x}$ is fixed, the objective (1) together with Constraints (2) – (4) and (9) can be decomposed into $K$ separate circulation problems with inflow requirement constraints (call it *circulation problem* for short). The circulation problem can be further transformed into a minimum cost flow model. The value of $opt_x(\mathbf{x})$ can be obtained by summing up the optimal solution values of all $K$ circulation problems.

The circulation problem associated with each worker type $k$ is given as:

$$(CP_k) \quad \min\ \alpha \sum_{j \in V_T} y_{k,0,j} + \beta \sum_{i \in V} \sum_{j \in \delta^+(i)} d_{i,j} y_{k,i,j} \tag{16}$$
$$\text{s.t.} \sum_{j \in \delta^-(i)} y_{k,j,i} = v_{k,i}, \forall\, i \in V_T \tag{17}$$
$$\sum_{j \in \delta^-(i)} y_{k,j,i} - \sum_{j \in \delta^+(i)} y_{k,i,j} = 0,\ \forall\, i \in V_T \tag{18}$$
$$y_{k,i,j} = 0,\ \forall\, (i,j) \notin e(\mathbf{x}) \tag{19}$$
$$y_{k,i,j} \geq 0 \text{ and integer},\ \forall\, i,\ j \in V \tag{20}$$

We denote by vector variable $\mathbf{y_k}$ the flows on a graph $G_k = (V_k, E_k)$, where $V_k = V$ and $E_k = e(\mathbf{x})$. A feasible vector $\mathbf{y_k}$ must satisfy the demand required by each task (see Constraints (17)). In the following context, we construct a new graph $G'_k = (V'_k, E'_k)$ from $G_k$. The vertex set $V'_k$ is $\{0, 1, \ldots, n\} \cup \{0', 1', \ldots, n'\}$ and the edge set $E'_k$ is con-

structed as:

$$\forall\,(0,i)\in E_k, i\in V_T \leftrightarrow (0,i)\in E'_k,\ cap(0,i)=v_{k,i} \quad (21)$$

$$\forall\,(i,j)\in E_k, i,j\in V_T \leftrightarrow (i,j')\in E'_k,\ cap(i,j')=+\infty \quad (22)$$

$$\forall\,(j,0)\in E_k, j\in V_T \leftrightarrow (j',0')\in E'_k,\ cap(j',0')=v_{k,j} \quad (23)$$

where $cap(i,j)$ denotes the flow capacity on edge $(i,j)$. We can uniquely transform $\mathbf{y'_k}$ (the flows on the graph $G'_k$) to $\mathbf{y_k}$ by the following equations:

$$\begin{aligned} &y_{k,i,j}=y'_{k,i,j'},\ \forall\,(i,j)\in E_T\\ &y_{k,i,0}=v_{k,i}-\sum_{j\in\delta^+(i)-\{0\}} y'_{k,i,j'},\ \forall\, i\in V_T\\ &y_{k,0,i}=v_{k,i}-\sum_{j\in\delta^-(i)-\{0\}} y'_{k,j,i'},\ \forall\, i\in V_T \end{aligned} \quad (24)$$

Reversely, Equations (25) transform $\mathbf{y_k}$ to $\mathbf{y'_k}$.

$$\begin{aligned} &y'_{k,i,j'}=y_{k,i,j},\ \forall\,(i,j)\in E_T\\ &y'_{k,i',0'}=\sum_{j\in\delta^-(i)-\{0\}} y_{k,j,i},\ \forall\, i\in V_T\\ &y'_{k,0,i}=\sum_{j\in\delta^+(i)-\{0\}} y_{k,i,j},\ \forall\, i\in V_T \end{aligned} \quad (25)$$

Note that if we substitute $\mathbf{y'_k}$ in Equations (25) into Equations (24), we can get $y_{k,i,j}=y_{k,i,j},\ \forall\,(i,j)\in E$; this implies that there is a one-to-one correspondence between $\mathbf{y_k}$ and $\mathbf{y'_k}$. By substituting Equations (24) into (16), we can rewrite the objective function of the model $(CP_k)$ as:

$$\begin{aligned} &\alpha\sum_{j\in V_T} y_{k,0,j}+\beta\sum_{i\in V}\sum_{j\in\delta^+(i)} d_{i,j}y_{k,i,j}\\ =&\alpha\sum_{j\in V_T} y_{k,0,j}+\beta\sum_{j\in V_T} d_{0,j}y_{k,0,j}+\beta\sum_{(i,j)\in E_T} d_{i,j}y_{k,i,j}\\ &+\beta\sum_{i\in V_T} d_{i,0}y_{k,i,0}\\ =&\sum_{j\in V_T}(\alpha+\beta d_{0,j})y_{k,0,j}+\sum_{(i,j)\in E_T}\beta d_{i,j}y_{k,i,j}+\sum_{i\in V_T}\beta d_{i,0}y_{k,i,0}\\ =&\sum_{i\in V_T}(\alpha+\beta d_{0,i})(v_{k,i}-y'_{k,i',0'})+\sum_{(i,j)\in E_T}\beta d_{i,j}y'_{k,i,j'}\\ &+\sum_{i\in V_T}\beta d_{i,0}(v_{k,i}-y'_{k,0,i})\\ =&\sum_{i\in V_T} v_{k,i}(\alpha+\beta d_{0,i}+\beta d_{i,0})+\sum_{i\in V_T}-(\alpha+\beta d_{0,i})y'_{k,i',0'}\\ &+\sum_{(i,j)\in E_T}\beta d_{i,j}y'_{k,i,j'}+\sum_{i\in V_T}-\beta d_{i,0}y'_{k,0,i} \end{aligned} \quad (26)$$

Equation (26) enables us to associate each edge $(i,j)\in E'_k$ with a cost, denoted by $c_k(i,j)$, where:

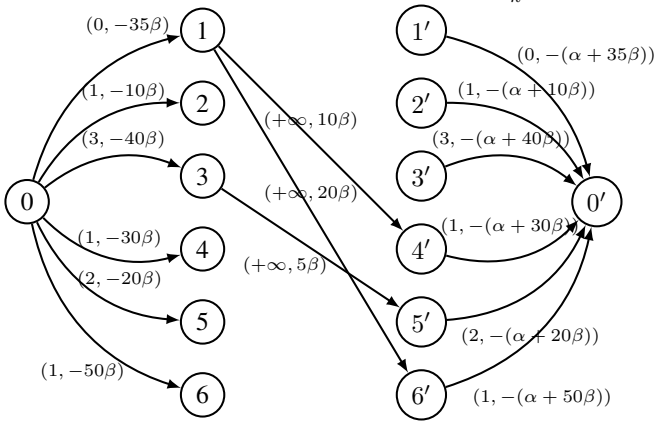
$$\begin{aligned} &c_k(0,i)=-\beta d_{i,0},\ \forall\,(0,i)\in E'_k, i\in V_T\\ &c_k(i,j')=\beta d_{i,j},\ \forall\,(i,j')\in E'_k, i,j\in V_T\\ &c_k(j',0')=-(\alpha+\beta d_{0,j}),\ \forall\,(j',0)\in E'_k, j\in V_T \end{aligned}$$

The optimal solution of the model $(CP_k)$ can be achieved by solving a minimum cost flow model on $G'_k$. Denoting by $mc_k$ the optimal solution value of the model $(CP_k)$, we can finally get:

$$opt_x(\mathbf{x})=\sum_{1\le k\le K}\Big(\sum_{i\in V_T} v_{k,i}(\alpha+\beta d_{0,i}+\beta d_{i,0})+mc_k\Big) \quad (27)$$

Figure 4 pictorially demonstrates the conversion from $G_k$ into $G'_k$, where $k=1$ and $E_k$ is the set of all edges in Figure 3. Since $\{(1,4),(1,6),(3,5)\}\in E_T$, the edges $(1,4'),(1,6'),(3,5')$ appear in $G'_k$. The two numbers in the parenthesis adjacent to each edge are its cost $c_k(i,j)$ and capacity $cap(i,j)$, respectively. If we set $\beta=0$ and $\alpha=1$, the minimum cost flows on $G'_k$ shown in Figure 4 are $y'_{k,0,3}=y'_{k,3,5'}=y'_{k,5',0'}=2$ and $y'_{k,i,j}=0$ for other edges $(i,j)$ in $E'_k$. Hence, from Expression (24) we know that the optimal solution value of the model $(CP_1)$, namely the minimum number of type 1 workers employed, is equal to $(0+1+3+1+2+1)-2=6$.

Figure 4: An example of the graph $G'_k$.




## 5 A Tree-based Tabu Search Algorithm

Tabu search (TS) algorithm is a meta-heuristic that combines local search with a memory scheme in order to avoid the traps of local optima. It has been successfully applied to a wide variety of vehicle routing problems, such as the VRPTW [Potvin *et al.*, 1996], the VRP with soft time windows [Taillard *et al.*, 1997], the split delivery VRP [Archetti *et al.*, 2006] and the three-dimensional loading capacitated VRP [Zhu *et al.*, 2012]. In most of the existing literature on the VRP variants, solutions are represented by lists (or sequences) of vertices complying with some additional constraints.

We refer the reader to [Glover and Laguna, 1998] for more details on the TS algorithm. We adopt the standard framework of the TS algorithm for the MAPTWTC. Recall from Section 3 that there must exist an optimal MAPTWTC solution that can be derived from a tree. Consequently, in our tree-based TS (TTS) algorithm, the search space is restricted to a set of feasible trees, from which the optimal solution can be obtained. The overview of our TTS algorithm is presented in Algorithm 1.

**Algorithm 1:** The tree-based tabu search algorithm.

```
1  Randomly generate a feasible tree t ; Set tabuList ← ∅ and
   best ← t ; while termination criteria are not satisfied do
2      N(t) ← all admissible and feasible neighbors of t ;
       t' ← arg min_{t''∈N(t)} {opt_w(t2w(t''))} ;
3      if the search process is trapped in a local optimum then
4          t ← diversify(t) ;
5          tabuList ← ∅ ;
6      else
7          Update tabuList for the current move ;
8          t ← t' ;
9          if opt_w(t2w(t)) < opt_w(t2w(best)) then best ← t ;
10     end
11 end
```

For a given feasible tree $\mathbf{t}$, $N(\mathbf{t})$ is the set of its feasible and admissible neighbors (i.e., the neighbors are not in the tabu list or allowed by the aspiration criteria). The neighborhood is generated by two tree-based operators, namely *vertex-relocate operator* and *subtree-relocate operator*. Before performing the operators on a tree $\mathbf{t}$, we apply the $t2w$ function to obtain its corresponding vector $\mathbf{w}$, which can help efficiently check the feasibility of its neighboring trees.

The vertex-relocate operator $vertex\_relocate(\mathbf{t}, i, j)$ works as follows. A non-root vertex $i$ is first relocated to be a child of vertex $j$ and its children are then linked to its father vertex such that a new tree $\mathbf{t}'$ is created. We stipulate that the vertex $j$ cannot be the descendant of vertex $i$. After the vertex-relocate operation, the resultant tree $\mathbf{t}'$ is feasible if $w_j + p_j + d_{j,i} \leq e_i$. Note that all infeasible trees are excluded from $N(\mathbf{t})$.

The subtree-relocate operator $subtree\_relocate(\mathbf{t}, i, j)$ is analogous to the vertex-relocate operator except that the relocation is applied to subtrees instead of vertices. We first remove the subtree rooted at vertex $i$ from $\mathbf{t}$ and then link vertex $i$ to be a child of vertex $j$. The infeasible resultant trees are also excluded from $N(\mathbf{t})$. In order to check the feasibility of the new tree efficiently, we pre-calculate the latest time $l(i)$ at which the task at vertex $i$ can be started while the subtree rooted at vertex $i$ is still feasible. The value of $l(i)$ can be recursively calculated by:

$$l(i) = \min \begin{cases} e_i \\ l(j) - p_i - d_{i,j},\ \forall\ j \text{ and } father(j) = i \end{cases} \quad (28)$$

Then, the operator $subtree\_relocate(\mathbf{t}, i, j)$ generates a feasible tree if $w_j + p_j + d_{j,i} \leq l(i)$.

The diversification phase in line **??** is implemented as follows. Firstly, a subset of vertices in the tree is extracted. Secondly, these vertices are selected in a random order and are sequentially reinserted into the partial tree at a random position. Each relocation must ensure the feasibility of the intermediate partial trees. Since $d_{0,i} \leq e_i$, it is guaranteed that all extracted vertices can be feasibly reinserted into the partial tree. This is because vertex $i$ can be at least feasibly linked to the root vertex (i.e., the depot).

# 6 Computational Results

To evaluate our TTS algorithm, we conducted experiments on two sets of test instances. The first instance set, introduced by [Li *et al.*, 2005], was generated using the geometric data of Hong Kong and Singapore. The number of task vertices in these instances ranges from 79 to 106, and the number of worker types (i.e., $K$) is fixed to 5. The second instance set was generated by us from 56 well-known Solomon's VRPTW instances [Solomon, 1987]. All instances in this set have 100 vertices and 5 worker types. For each worker type, the number of workers required was generated using the similar way to that of the first instance set (refer to [Li *et al.*, 2005] for details). For all instances, we set $\alpha \gg \beta$ (to be precise, we set $\alpha = 10^6$, $\beta = 1$), implying that labor cost dominates the traveling cost.

Our TTS algorithm was coded in C++ and all experiments were conducted on a Dell server with Intel Xeon E5520 2.66 GHz CPU, 8 GB RAM and Linux operating system. For each instance, we executed our TTS algorithm 5 times using different random seeds. By some preliminary experiments, we set the tabu tenure to 100. If no improvements can be achieved after 30 consecutive iterations, we believe that the search is trapped in a local optimum and therefore activate the diversification phase. We also applied ILOG CPLEX 11.1 with default settings to solve the model (*IP*) for each instance. A time limit of 2 CPU hours was imposed on CPLEX for each execution. We find that CPLEX could only optimally solve some of the test instances. In addition, when we conducted experiments on several instances with more than 150 tasks, CPLEX encountered "Out of memory" exception for most of these instances.

The computational results of the first instance set is shown in Table 2, where the columns with headings "# W." and "Dist." report the numbers of workers employed and the total distance traveled, respectively. We collected the results of our TTS algorithm at 10 and 30 minutes since its start, and then present the best results of 5 runs in the blocks "TTS (10 min)" and "TTS (30 min)". The results of CPLEX are displayed in the block "CPLEX". The instances that were optimally solved by CPLEX are marked with an asterisk (*) in the first column. As a comparison, we put in Table 2 the results of the algorithms *Simple-Append with simulated annealing* (A + SA) and *Block-Insertion with simulated annealing* (I + SA) reported by [Li *et al.*, 2005]. As they did not report the total traveling distance in their article, we can only show the number of workers employed ("# W.") and the computation times ("CPU (s)") consumed by their algorithms. Note that the values in the columns "CPU (s)" were obtained on a Pentium IV 2.0GHz machine. As clearly shown in this tables, both TTS and CPLEX outperform two SAs in terms of solution quality. However, TTS algorithm did not show too much superiority over CPLEX on this set of instances.

To further compare the TTS algorithm with CPLEX, we conducted experiments on the second instance set, and report the results in Table 3. This set of instances is divided into six groups: C1 (9 instances), C2 (8 instances), R1 (12 instances), R2 (11 instances), RC1 (8 instances) and RC2 (8 instances). Due to space limitations, we only report the aver-

Table 2: Computational results of the first instance set.

| Instance | TTS (10 mins) | | TTS (30 mins) | | CPLEX | | A + SA | | I + SA | |
|---|---|---|---|---|---|---|---|---|---|---|
| | # W. | Dist. | # W. | Dist. | # W. | Dist. | # W. | CPU(s) | # W. | CPU(s) |
| HHK_1* | 111 | 6107 | 111 | 6107 | 111 | 6107 | 113 | 146 | 111 | 361 |
| HHK_2* | 119 | 5965 | 119 | 5965 | 119 | 5965 | 122 | 105 | 119 | 541 |
| HHK_3* | 116 | 5758 | 116 | 5758 | 116 | 5758 | 118 | 95 | 116 | 753 |
| HHK_4* | 121 | 6225 | 121 | 6225 | 121 | 6224 | 129 | 137 | 121 | 604 |
| HHK_5* | 104 | 5892 | 104 | 5892 | 104 | 5892 | 119 | 138 | 107 | 281 |
| HSG_1* | 116 | 5658 | 116 | 5658 | 116 | 5658 | 120 | 120 | 116 | 983 |
| HSG_2* | 123 | 5685 | 123 | 5685 | 123 | 5685 | 126 | 139 | 124 | 929 |
| HSG_3* | 139 | 6397 | 139 | 6397 | 139 | 6397 | 140 | 162 | 139 | 1804 |
| HSG_4 | 126 | 5491 | 126 | 5491 | 126 | 5491 | 129 | 137 | 127 | 1050 |
| HSG_5* | 114 | 5970 | 113 | 5972 | 113 | 5972 | 119 | 138 | 115 | 713 |
| LHK_1* | 74 | 4311 | 74 | 4311 | 72 | 4151 | 85 | 70 | 80 | 414 |
| LHK_2 | 67 | 4481 | 67 | 4414 | 64 | 4469 | 77 | 87 | 71 | 297 |
| LHK_3 | 70 | 4690 | 70 | 4689 | 68 | 4688 | 87 | 81 | 74 | 267 |
| LHK_4 | 89 | 4964 | 89 | 4964 | 87 | 4933 | 98 | 80 | 95 | 413 |
| LHK_5* | 79 | 4491 | 78 | 4599 | 76 | 4696 | 88 | 84 | 83 | 593 |
| LSG_1 | 58 | 4259 | 58 | 4259 | 58 | 4404 | 82 | 114 | 64 | 417 |
| LSG_2 | 45 | 4064 | 44 | 4064 | 45 | 3914 | 64 | 109 | 50 | 327 |
| LSG_3 | 55 | 4215 | 55 | 4063 | 55 | 4230 | 73 | 131 | 62 | 470 |
| LSG_4 | 62 | 4400 | 59 | 4464 | 60 | 4289 | 86 | 108 | 68 | 671 |
| LSG_5 | 58 | 4149 | 54 | 4319 | 58 | 4219 | 77 | 112 | 62 | 441 |

age results of each instance group. The columns "Avg. # W. " and "Avg. dist." give the average numbers of workers employed and the average total traveling distance, respectively. The results generated by the TTS algorithm with 10 minutes of computation time are already very competitive, compared to the results generated by CPLEX with 2 hours of computation time. When the computation time increases to 30 minutes, the results produced by the TTS algorithm are superior to those found by CPLEX solutions; TTS is able to reduce the numbers of workers generated by CPLEX by 4.1% on average. After 2 hours of computation time, the TTS algorithm further improved the solutions and the percentage reduction of "Avg. # W." increases to 6.6%.

Table 3: Computational results of the second instance set.

| Instance group | TTS (30min) | | TTS (2hr) | | CPLEX | |
|---|---|---|---|---|---|---|
| | Avg. # W. | Avg. dist. | Avg. # W. | Avg. dist. | Avg. # W. | Avg. dist. |
| C1 | 46.80 | 6420.4 | 46.11 | 6341.0 | 47.11 | 6284.1 |
| C2 | 22.00 | 5245.1 | 21.50 | 5280.1 | 21.63 | 5399.0 |
| R1 | 64.38 | 5944.7 | 63.50 | 5949.8 | 68.42 | 6215.3 |
| R2 | 22.58 | 5456.9 | 21.64 | 5440.1 | 24.18 | 5559.0 |
| RC1 | 63.50 | 7063.7 | 62.88 | 7001.8 | 69.00 | 7449.4 |
| RC2 | 25.98 | 6940.2 | 24.63 | 6744.6 | 27.38 | 7037.8 |

## 7 Conclusion

The main contribution of this study is the tree representation of solutions for the MAPTWTC. Its optimal solution can be obtained from a tree by solving a minimum cost flow problem for each worker type. As a result, we can get the optimal MAPTWTC solution by identifying the best tree. This observation enables us to restrict the search space to the set of spanning trees on the graph, significantly reducing the scale of the solution space of the problem. To highlight the importance and effectiveness of the new solution representation, we proposed a simple and standard tabu search algorithm to solve the problem. The computational results show that our tree-based tabu search algorithm not only outperforms the simulated algorithms proposed in [Li *et al.*, 2005], but also is superior to CPLEX.

Our work presents a new thinking on solving the VRP models with synchronization constraints. In addition, future research can consider developing more sophisticated meta-heuristics by making use of this tree representation.

## References


[Ahuja *et al.*, 1993] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows: theory, algorithms, and applications. 1993.

[Archetti *et al.*, 2006] C. Archetti, M. G. Speranza, and A. Hertz. A tabu search algorithm for the split delivery vehicle routing problem. *Transportation Science*, 40(1):64 – 73, 2006.

[Bredström and Rönnqvist, 2008] D. Bredström and M. Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European journal of operational research*, 191(1):19 – 31, 2008.

[Brucker, 2007] P. Brucker. *Scheduling algorithms*. Springer, 2007.

[Dohn *et al.*, 2009] A. Dohn, E. Kolind, and J. Clausen. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, 36(4):1145 – 1157, 2009.

[Drexl, 2012] M. Drexl. Synchronization in vehicle routing–a survey of VRPs with multiple synchronization constraints. *Transportation Science*, 46(3):297 – 316, 2012.

[Glover and Laguna, 1998] F. Glover and M. Laguna, editors. *Tabu Search*. Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061, USA, 1998.

[Li *et al.*, 2005] Y. Li, A. Lim, and B. Rodrigues. Manpower allocation with time windows and job-teaming constraints. *Naval Research Logistics (NRL)*, 52(4):302 – 311, 2005.

[Lim *et al.*, 2004] A. Lim, B. Rodrigues, and L. Song. Manpower allocation with time windows. *Journal of the Operational Research Society*, 55(11):1178 – 1186, 2004.

[Potvin *et al.*, 1996] J.-Y. Potvin, T. Kervahut, B.-L. Garcia, and J.-M. Rousseau. The vehicle routing problem with time windows part I: Tabu search. *INFORMS Journal on Computing*, 8(2):158 – 164, 1996.

[Solomon, 1987] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254 – 265, 1987.

[Taillard *et al.*, 1997] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170 – 186, 1997.

[Tu *et al.*, 2010] D. Tu, S. Guo, H. Qin, W.-C. Oon, and A. Lim. The tree representation of feasible solutions for the TSP with pickup and delivery and LIFO loading. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010), Atlanta, Georgia, USA*, pages 191 – 196, 2010.

[Zhu *et al.*, 2012] W. Zhu, H. Qin, A. Lim, and L. Wang. A two-stage tabu search algorithm with enhanced packing heuristics for the 3L-CVRP and M3L-CVRP. *Computers & Operations Research*, 39(9):2178 – 2195, 2012.