Chapter 2

TIME-VARYING MINIMUM SPANNING TREES

1. Introduction

Given a network N, the *minimum spanning tree* (MST) problem is to find a connected acyclic subnetwork T that spans all the vertices of N, such that the sum of costs (or lengths) of the constituent arcs of T is minimum. The problem is further called a rooted minimum spanning tree problem, when there is a pre-specified vertex s, and the spanning tree T must have its root at s. To find the minimum spanning tree of a given network is one of the well-known problems in the field of network optimization; see Ahuja et al (1993); Graham et al (1985); Gabow et al (1986); Recski (1988).

Although the MST problem and its variants have been extensively studied in the literature, most of the works published so far treat the problem as a static one, where it is assumed that zero time is needed to travel from one vertex to another vertex, and that all attributes of the network are time-invariant. Apparently, as we indicated earlier, these assumptions are only approximations of real-world problems. In most practical situations, the network under consideration may inevitably change over time.

We will study the time-varying MST problem in this chapter. Our model considers a network where positive transit time $b(x, y, t)$ is needed to traverse an arc (x, y), at a cost $c(x, y, t)$; Moreover, both the transit time $b(x, y, t)$ and the cost $c(x, y, t)$ are time-varying, which are functions of the departure time at the vertex x. Waiting at a vertex x may be allowed, in order to catch the best timing to depart from x. Given a deadline κ and a root s, the problem is to find a rooted spanning tree to cover all vertices in the network, so that the total cost of the constituent

arcs of the spanning tree is minimum, while any vertex z of the network can be reached, before the deadline κ, along a path in the spanning tree that connects s and z.

Although the static version of the MST problem is polynomially solvable (see, for example, Ahuja et al (1993)), we will show that the time-varying version is, in general, NP-complete. After laying down some basic concepts and terminologies in Section 2 below, we will focus on the MST problem over a type of arc series-parallel networks (Section 3). We will show that this problem is NP-complete in the ordinary sense (Section 3.1), and present an algorithm that can find an optimal solution in $O((m+n)m\kappa^2)$ time, where m and n are the numbers of arcs and vertices, respectively (Section 3.2). We will then consider a more general network, an undirected network containing no subgraph homomorphic to K_4 (Section 4). We will derive an algorithm that can find an exact optimal solution in $O((m+n)m\kappa^2)$ time. The general case will be studied in Section 5. Its complexity in terms of strong NP-completeness will be examined. Heuristic algorithms will be developed and their time complexity and errors will be analyzed. Finally, some additional references and remarks will be given in Section 6.

2. Concepts and problem formulation

Let $N = (V, A, b, c)$ be a time-varying network. A vertex in N is known as the *source* (*root*), denoted as s. Assume that $b(x, y, t)$ is a positive integer, $t = 0, 1, ..., \kappa$, where κ is a positive integer, representing a given time limit. By an approach similar to that of Section 5, Chapter 1, the results developed in this chapter may also be generalized to situations where the transit time b is a non-negative integer.

Recall that a dynamic path in the time-varying network N is a path P where all departure times, arrival times, and waiting times at each vertex on P are specified. We now introduce the concept of *path-induced subnetwork* as below.

Definition 2.1 *For $x_j \in V$ and $j = 1, 2, ..., J$, where $1 \leq J \leq n$, suppose $P_j(s, x_j)$ is a dynamic path from s to x_j of time at most t. Let $V(P_j)$ and $A(P_j)$ be the vertex set and the arc set of P_j, respectively, and let $V' = \bigcup_j V(P_j)$, $A' = \bigcup_j A(P_j)$. Further, let $\Gamma(P_j)$ be the set of the triples $(x, y, \tau(x))$ over P_j, where $\tau(x)$ is the departure time at vertex x on P_j, and $\Gamma = \bigcup_j \Gamma(P_j)$. Define $I_j(x) = \{[\alpha(x), \tau(x)]\}$ as the set of time intervals (the waiting times at the vertices on P_j), and $I(x) = \bigcup_j I_j(x)$. Then, $N' = (V', A', b, c)$, together with Γ and $I(x)$ ($x \in V'$), is said to be a path-induced subnetwork of N. The subnetwork N' is also said to be generalized by paths P_j, $1 \leq j \leq J$, denoted by $N' = [P_1, P_2, ..., P_J]$.*

Obviously, a single dynamic path $P(s,x)$ in N is also a path-induced subnetwork of N. On the other hand, given a path-induced subnetwork of N' of N, there must exist paths P_1, P_2, ..., P_k, such that $N' = [P_1, P_2, ..., P_k]$. The following definition gives the concept of *dynamic spanning tree.*

Definition 2.2 *Let $N' = [P_1, P_2, ..., P_J]$ be a path-induced subnetwork of N and $t \leq \kappa$. N' is said to be a dynamic spanning tree of time at most t, denoted by $T(t)$, if it satisfies the following conditions:*

(i) For each $x \in V$, there exists a dynamic path $P(s,x)$ of time at most t in N';

(ii) If x is the end vertex of path P_i in N', then x must be neither in P_i again as an intermediate vertex, nor in any other path P_j in N', where $i \neq j$ and $1 \leq i, j \leq J$.

Definition 2.3 *Let $T(t)$ be a dynamic spanning tree of time at most t, and let*

$$\zeta(T(t)) = \sum_{(x,y,\tau)\in\Gamma} c(x,y,\tau) + \sum_{x\in V} \sum_{t'\in I(x),t'=0,1,...,\kappa} c(x,t')$$

A dynamic spanning tree $T^*(t)$ is said to be a minimum spanning tree of time at most t on the time-varying network $N(V,A,b,c)$, if

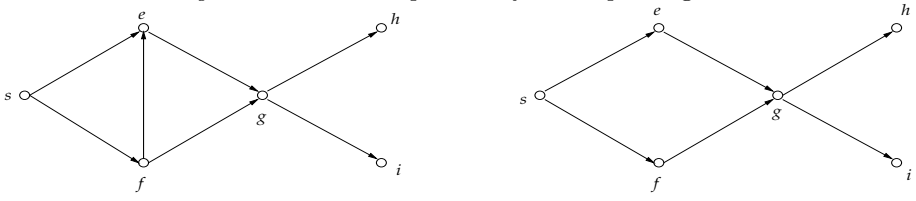
$$\zeta(T^*(t)) = \min_{T(t)\in\mathcal{T}(t)} \zeta(T(t))$$

where $\mathcal{T}(t)$ is the set of all dynamic spanning trees of time at most t.

The *time-varying minimum spanning tree* (TMST) problem is to determine the minimum spanning tree $T^*(\kappa)$ for the given time-varying network N.

Example 2.1

Figure 2.1. An example of a dynamic spanning tree



Consider a network N as shown in Figure 2.1(a), where

$$b(s,e,0)=1, b(s,f,0)=3, b(f,g,2)=1, b(e,g,1)=1,$$
$$b(f,g,3)=1, b(g,h,4)=1, b(g,i,2)=3,$$
$$c(s,e,0)=2, c(s,f,0)=1, c(f,g,2)=1, c(e,g,1)=1,$$
$$c(f,g,3)=2, c(g,h,4)=3, c(g,i,2)=2.$$

All other b and c are equal to ∞, while all $c(x,t)=0$. Waiting at any vertex is not allowed. A dynamic spanning tree with root at s is to be found for $\kappa=6$.

We can see that there are two dynamic paths: $P_1=(s,e,g,i)$ with $\tau(s)=0$, $\tau(e)=1$ and $\tau(g)=2$ (arrival at i at time 5); $P_2=(s,f,g,h)$ with $\tau(s)=0$, $\tau(f)=3$ and $\tau(g)=4$ (arrival at h at time 5). Let $N'=[P_1,P_2]$ be a path-induced subnetwork. From Definition 2.1, we have $V'=V$, $A'=A\backslash\{(f,e)\}$, $\Gamma=\{(s,e,0),(s,f,0),(e,g,1),(f,g,3),(g,h,4),(g,i,2)\}$, and $I(x)=\emptyset$ for all $x\in V$. By Definition 2.2, we can see that N' is also a dynamic spanning tree of N, since all vertices in V can be visited from the root s within time 6, and the end vertices of P_1 and P_2, i and h, only be visited once, respectively. We denote T as this tree (see Figure 2.1(b)). By Definition 2.3, the cost of T is $\zeta(T)=c(s,e,0)+c(s,f,0)+c(e,g,1)+c(f,g,3)+c(g,h,4)+c(g,i,2)=11$.

The example above also shows a difference between the static spanning tree and a dynamic spanning tree as we define here. We can note that the underlying graph of T (Figure 2.1(b)) contains a cycle $C=(s,e,g,f,s)$. On the other hand, if we remove any edge from this cycle, then neither h nor i could be visited within time κ.

Remarks are given below.

Remark 2.1

(1) The model described above covers the situation where an arc (x,y) is not available during an interval $[t_1,t_2]$ (For example, the arc is not usable due to repair/maintenance work during the interval). This can be achieved by setting the value of $b(x,y,t)$ or $c(x,y,t)$ to be infinity for the interval $[t_1,t_2]$.

(2) Solomon (1986) formulates a MST model with time window constraints, where a vertex x_i $(i=1,...,n)$ should only be visited during a given time window $[e_i,l_i]$. Early arriving at a vertex is allowed, but it must then wait at the vertex until the earliest visiting time (In other words, departure from the vertex before its earliest visiting time is prohibited).

A problem with time windows can be converted into the model we discuss here as follows: Suppose that the time window of a vertex y is $[e_y, l_y]$, and suppose that x_i (and z_i) are immediate preceding (and succeeding) vertices of y. Let $b(y, z_i, t) = +\infty$ for any $t < e_y$, and $b(y, z_i, t) = t_{y,z_i}$ for any $t \geq e_y$, where t_{y,z_i} is the travel time between y and z_i. Also, let $c(x_i, y, t') = d_{x_i,y}$ for any $t' + b(x_i, y, t') \leq l_y$, where $d_{x_i,y}$ is the cost of travelling from x_i to y, and $c(x_i, y, t') = +\infty$ for any $t' + b(x_i, y, t') > l_y$. Then, one can arrive at y earlier than e_y and wait at y, but he cannot depart from y before time e_y since the transit time from y to any of its succeeding vertices is infinite if the departure time is earlier than e_y. On the other hand, arriving at y later than l_y will lead to an unacceptable schedule, since the travel cost to arrive at y from any of its preceding vertices is infinite. Consequently, the optimal solution for this time-varying MST model will automatically satisfy the time window constraints.

3. Arc series-parallel networks

We will concentrate, in this section, on the TMST problem in which waiting at a vertex is arbitrary allowed, and the network has an arc series-parallel structure (see Duffin (1965); Weinberg (1971)). This type of networks have interest themselves in applications (see, e.g., application of Boolean algebra to switching circuits (Shannon 1938)). In addition, results derived on this type of networks may offer valuable insights for the study of more general problems.

A network that meets the following properties is an arc series-parallel network (ASP network):

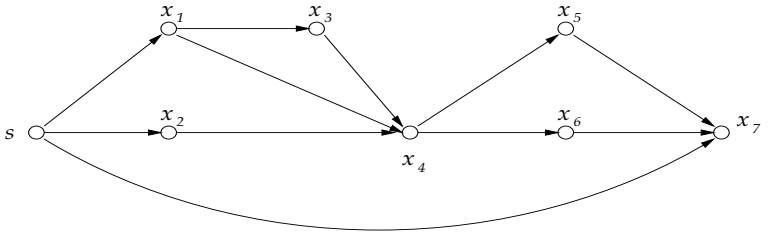
(i) An arc (s, ρ) is an ASP network;

(ii) If G_1 and G_2 are two ASP networks and s_1, s_2 and ρ_1, ρ_2 are sources and sinks of G_1 and G_2, respectively, then G_p generated from G_1 and G_2 by merging s_1 with s_2 and ρ_1 with ρ_2 is an ASP network. Also, G_s formed from G_1 and G_2 by merging ρ_1 with s_2 is an ASP network.

In fact, any ASP network can be induced by the two properties above. Computationally, we can tell whether a given network is arc series-parallel, by using the algorithm of Valdes et al (1982), which runs in $O(m + n)$ time. An example of an arc series-parallel network is given in Figure 2.2.

In what follows, we will first study the complexity (in terms of NP-completeness) of the TMST problem over time-varying ASP networks. We will then present an exact algorithm which can find a minimum spanning tree in pseudopolynomial time. Note that when we say that

Figure 2.2. An arc series-parallel network



the problem we consider has an arc series-parallel structure, we always imply that the source vertex (root) of the problem is the starting vertex in the ASP network (see, e.g., Figure 2.2).

3.1 Complexity

We now show that, the TMST problem on an ASP network is NP-complete. The decision version of the problem is defined below.

TMST-SPN Given a time-varying arc series-parallel network $N(V, A, b, c)$, a time limit κ, and a threshold value K, does there exist a dynamic spanning tree $T(\kappa)$ of time at most κ, such that $\zeta(T(\kappa)) \leq K$?

We will show that the Knapsack problem (see Section 3, Chapter 1) is reducible to TMST-SPN. The following theorem establishes the NP-completeness of the TMST problem with arc series-parallel networks.

Theorem 2.1 *The TMST problem on an arc series-parallel network is NP-complete in the ordinary sense.*

Proof: For any given instance of Knapsack, we can construct a time-varying network N, with structure same as Figure 1.3, and transit times, transit costs, B, K, and time limit κ as specified in the proof of Theorem 1.1. (Note that T is the time limit used in the proof of Theorem 1.1).

We now prove that a "yes" answer to Knapsack is equivalent to a "yes" to the decision version of TMST-SPN.

If Knapsack has a set $S \subseteq Q$ such that $\sum_{i \in S} v_i \geq v^*$ and $\sum_{i \in S} w_i \leq w^*$, then we can obtain a path $P(x_0, x_{n+1})$ by the following way: starting from x_0 at time zero, for each i, if $i \in S$, then traverse arcs (x_{i-1}, x'_i) and (x'_i, x_i); if $i \notin S$, then traverse arc (x_{i-1}, x_i). At last, traverse arc (x_n, x_{n+1}). Let $w(x_i) = 0$ $(1 \leq i \leq n-1)$ and $w(x_n) = n + w^* - \alpha(x_n)$. Notice that, since $\sum_{i \in S} w_i \leq w^*$, we have $\alpha(x_n) = \sum_{i \in S}(w_i + 1) + \sum_{i \notin S} 1 = n + \sum_{i \in S} w_i \leq n + w^*$, and $\alpha(x_{n+1}) \leq n + w^* + 1 = \kappa$, where $\alpha(x_n)$ and $\alpha(x_{n+1})$ are the arrival times of P at vertices x_n and x_{n+1}, respectively. Combining P with arcs (x_{i-1}, x'_i) for each $i \notin S$, we obtain a tree, denoted as T^o. Clearly, T^o is a spanning tree of N within time

duration κ. Moreover, since $\sum_{i\in S} v_i \geq v^*$, i.e., $-\sum_{i\in S} v_i \leq -v^*$, we have $\zeta(T^o) = \zeta(P) = \sum_{i\notin S} B + \sum_{i\in S}(B - v_i) = nB - \sum_{i\in S} v_i \leq nB - v^* = K$.

If TMST-SPN has a spanning tree T with the total cost not exceeding K, then there exists a path $P(x_0, x_{n+1})$ such that $\zeta(P) \leq \zeta(T) \leq K$. Let $S = \{i|(x'_i, x_i) \in A(P), 1 \leq i \leq n\}$. We have $\zeta(P) = \sum_{i\in S} c(x'_i, x_i, \tau(x'_i)) + \sum_{i\notin S} B = \sum_{i\in S}(B - v_i) + \sum_{i\notin S} B = nB - \sum_{i\in S} v_i \leq K = nB - v^*$, which implies $\sum_{i\in S} v_i \geq v^*$. On the other hand, since T is a spanning tree of N within time κ, by the construction of the network N, we must have $\alpha(x_n) = \sum_{i\in S}(w_i + 1) + \sum_{i\notin S} 1 = n + \sum_{i\in S} w_i \leq n + w^*$, where $\alpha(x_n)$ is the arrival time of P at x_n. Therefore, $\sum_{i\in S} w_i \leq w^*$.

The analyses above show that TMST-SPN can be reduced from Knapsack and thus it is NP-complete. On the other hand, the optimal solution of TMST-SPN can be found by an algorithm in pseudopolynomial time (see Section 3.2 below) and hence it is NP-complete in the ordinary sense (cf. Garey and Johnson 1979). This completes the proof. □

3.2 A pseudo-polynomial algorithm

In an ASP network, a vertex x is called a *spreading vertex* if $d^+(x) > 1$; or a *converging vertex* if $d^-(x) > 1$, where $d^+(x)$ and $d^-(x)$ are outdegree and indegree of x, respectively. To develop our algorithm, we introduce the concept of *diamond* as follows.

Definition 2.4 *Let $f \in V$ be a spreading vertex, and $g \in V$ a converging vertex. If there are two paths from f to g containing no other spreading and converging vertices, then the subgraph induced by these two paths is called a "diamond".*

Usually, we denote $D(f, g)$ as a diamond constructed by a spreading vertex f and a converging vertex g. We also need the following definition.

Definition 2.5 *Suppose that $P(f, g)$ is a path in N which contains no other spreading and converging vertices other than f and g. For each vertex x in this path, define $d_{f,g}(x, t_s, t)$ as the cost of a shortest path from f to x so that this path can be traversed within the time duration $[t_s, t]$. If such a path does not exist, let $d_{f,g}(x, t_s, t) = \infty$.*

The following lemma gives a recursive relation to compute $d_{f,g}(x, t_s, t)$.

Lemma 2.1 *Let $P = (f, g)$ be a path which has no spreading and converging vertices other than f and g. Then, we have $d_{f,g}(f, t_s, t) = 0$ for all $0 \leq t_s \leq t \leq \kappa$, $d_{f,g}(y, t_s, t_s) = \infty$ for all $y \neq f$, and*

$$d_{f,g}(y, t_s, t) = \min\{d_{f,g}(y, t_s, t-1) + c(y, t-1), \min_{\{u|u+b(x,y,u)=t\}} \{d_{f,g}(x, t_s, u) + c(x, y, u)\}\},$$

for $t_s < t \le \kappa$ and $y \ne f$, where x is the predecessor of y in P.

Lemma 2.1 is a simple generalization of Lemma 1.1. For a diamond $D(f,g)$, the notation $d_{f,g}(g,t_s,t)$ may cause some confusion since there are two paths ending at g. We therefore use $d^{x_1}_{f,g}(g,t_s,t)$ and $d^{x_2}_{f,g}(g,t_s,t)$ to denote the minimum costs from f to g along paths P_1 and P_2, where x_1 and x_2 are the predecessors of g in P_1 and P_2, respectively.

Definition 2.6 *Let $D(f,g)$ be a diamond as defined in Definition 2.4 and $D' = D\backslash\{g\}$. Define $\delta_I(D,t_s,t)$ as the cost of the minimum spanning tree of D such that $\tau(f) \ge t_s$ and $\alpha(g) \le t$. Define $\delta_E(D',t_s)$ as the cost of the minimum spanning tree of D' such that $\tau(f) \ge t_s$.*

Lemma 2.2 *Suppose D is a diamond in N and $0 \le t_s \le \kappa$. Then, we have*

$$\begin{aligned}\delta_I(D,t_s,t) &= \min\{d^{x_1}_{f,g}(g,t_s,t) + d_{f,g}(x_2,t_s,\kappa),\\ &\qquad d^{x_2}_{f,g}(g,t_s,t) + d_{f,g}(x_1,t_s,\kappa)\},\ \textit{for any } t_s \le t \le \kappa,\\ \delta_E(D',t_s) &= d_{f,g}(x_1,t_s,\kappa) + d_{f,g}(x_2,t_s,\kappa),\end{aligned}$$

where x_1 and x_2 are the predecessors of g in P_1 and P_2, respectively.

Proof: Notice that $D = P_1(f,...,x_1,g) \cup P_2(f,...,x_2,g)$, and P_1 and P_2 have no common vertex except f and g. Thus, there are two ways to span D only, i.e., reach g by path P_1 and reach x_2 by P_2, or reach g by path P_2 and reach x_1 by P_1. Notice that if g is reached through P_1 with $\alpha(g) \le t$, then the arrival time at vertex x_2 could be less than or equal to κ. Therefore the cost of the minimum spanning tree of D should be the minimum between these two. □

The key ideas of our algorithm can be described below.

(a) If N is a path, then $d_{s,\rho}(\rho,0,\kappa)$ is the cost of the minimum spanning tree.

(b) If N is a diamond, then $\delta_I(N,0,\kappa)$ is the cost of the minimum spanning tree.

(c) Otherwise, choose a diamond D in N. Calculate δ_I and δ_E respectively, and make a *contracting operation* as follows:
(i) Delete P_1 and P_2 in N except vertices f and g.
(ii) Create an artificial vertex x' and two artificial arcs (f,x') and (x',g). Let $\delta(x',t_s,t) = \delta_E(D',t_s)$ and $\delta(g,t_s,t) = \delta_I(D,t_s,t) - \delta_E(D',t_s)$ for any $0 \le t_s \le t \le \kappa$ (if both $\delta_I(D,t_s,t)$ and $\delta_E(D',t_s)$ are infinite, then $\delta(g,t_s,t) = \infty$). Then, a diamond is converted equivalently to a path $P(f,x',g)$.

(iii) It is possible that f and g are no longer a spreading vertex and a converging vertex in the new network. Suppose that the path $P(f', g')$ contains $P(f, g)$ as its subpath. Calculate $d_{f',g'}(x, t_s, t)$ for each vertex x in $P(f', g')$ according to the formula given in Lemma 2.3 below.

(d) Repeat step (c) until N becomes a path or a diamond.

Note that after a contracting operation, the new network obtained, denoted as N', contains an artificial path $P(f, g)$. Moreover, a spanning tree T' in N' should contain either an arc (f, x') or arcs (f, x') and (x', g), since $d^-(x') = d^+(x') = 1$, where $d^-(x')$ and $d^+(x')$ are in-degree and out-degree of x' in N', respectively. Therefore, we can compute the cost of T' as follows:

$$\zeta(T') = \begin{cases} \sum_{(x,y)\in A(T')\setminus(f,x')} c(x, y, \tau(x)) + \delta(x', \tau(f), \kappa), \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } (x', g) \notin A(T') \\ \sum_{(x,y)\in A(T')\setminus\{(f,x'),(x',g)\}} c(x, y, \tau(x)) \\ \qquad + \delta(x', \tau(f), \alpha(x')) + \delta(g, \tau(f), \alpha(g)), \quad \text{otherwise} \end{cases}$$

For the case where there are multiple artificial paths, we can compute the cost of a spanning tree in a similar way. The formula given in Lemma 2.1 can be revised as follows:

Lemma 2.3 *Let $P(f, g)$ be a path, $0 \le t_s \le \kappa$, $d_{f,g}(f, t_s, t) = 0$ for all $t_s \le t \le \kappa$ and $d_{f,g}(y, t_s, t_s) = \infty$ for all $y \ne f$. For $t_s < t \le \kappa$ and $y \ne f$, we have:*

(i) If (x, y) is not an artificial arc, then

$$d_{f,g}(y, t_s, t) = \min\{d_{f,g}(y, t_s, t-1) + c(y, t-1), \min_{\{u|u+b(x,y,u)=t\}} \{d_{f,g}(x, t_s, u) + c(x, y, u)\}\}$$

(ii) If (x, y) is an artificial arc, then

$$d_{f,g}(y, t_s, t) = \min\{d_{f,g}(y, t_s, t-1) + c(y, t-1), \min_{t_s \le u < t} \{d_{f,g}(x, t_s, u) + \delta(y, u, t)\}\}$$

where x is the predecessor of y in P.

Let $P(f', x_1, x_2, ..., x_r, f, x', g, x_{r+1}, ..., g')$ be a path we are considering now, and $P(f, x', g)$ be a path which comes from a diamond D. For notational convenience, let us assume that $d_{s_{i+1},\rho_{j+1}}(x', t_s, t)$ denotes the cost of the minimum spanning tree of the subnetwork generated by D'

and the path $P(f', x_1, ..., x_r, f)$, with $\tau(f') \geq t_s$, while $d_{f',g'}(g, t_s, t)$ denotes the cost of the minimum spanning tree of the subnetwork generated by D and the path $P(f', x_1, ..., x_r, f)$, with $\tau(f') \geq t_s$ and $\alpha(g) \leq t$.

We are now ready to present the following algorithm.

Algorithm TMST-SP
Begin
While N is neither a path nor a diamond **do**
Select arbitrarily a diamond D in N;
Compute $d_{f,g}(x, t_s, t)$ for all $x \in V(D)$ and for all $0 \leq t_s \leq t \leq \kappa$;
Compute $\delta_I(D, t_s, t)$ and $\delta_E(D', t_s)$ for $0 \leq t_s \leq t \leq \kappa$;
Delete D, except the vertices f and g, from N;
Create a path $P(f, x', g)$ in N and let $\delta(x', t_s, t) = \delta_E(D', t_s)$ and $\delta(g, t_s, t) = \delta_I(D, t_s, t) - \delta_E(D', t_s)$ for $0 \leq t_s \leq t \leq \kappa$;
End *While*;
If N is a path **then** $\zeta(T) = d_{s,\rho}(\rho, 0, \kappa)$;
If N is a diamond **then** $\zeta(T) = \delta_I(N, 0, \kappa)$
End.

Theorem 2.2 *TMST-SP can find an optimal solution for the time-varying minimum spanning tree problem with an arc series-parallel network considered in this section.*

Proof: Use induction on $|A|$. Consider the case with $m = 1$. Since there is only one arc, N is a path and the claim holds obviously. Assume that when $m < k$, the claim is true. Now we consider the case with $m = k$.

We examine the following cases:

(i) N is a path. By Lemma 2.1, the claim holds.

(ii) N is a diamond. By Lemma 2.2, the claim is also true.

(iii) N is neither a path nor a diamond. Then we select a diamond D in N, and compute $\delta_I(D, t_s, t)$ and $\delta_E(D', t_s)$ and change D to a path $P(f, x', g)$ to obtain a new network N'. Now, we prove $\zeta(T') = \zeta(T)$, where T' and T are the minimum spanning trees in N' and N, respectively.

First, we show $\zeta(T') \geq \zeta(T)$. Recall that N' differs from N, since a diamond D in N is replaced by a path $P(f, x', g)$. Because the indegree of x' equals one in N, the arc (f, x') must be in T'. Consider the arc (x', g). If $(x', g) \notin A(T')$, where $A(T')$ is the arc set of T', then we can restore the arc (f, x') to a spanning tree $T'_{D'}$ of D' with cost $\delta_E(D', t_s)$, where t_s is the departure time at f in T'. Combining $T'_{D'}$ and all other arcs in T' except (f, x'), we can obtain a spanning tree of N, denoted as T''. From Lemma 2.3, we know that the cost to reach x' from f with

$\tau(f) = t_s$ is $\delta(x', t_s, t)$. Then we have

$$\zeta(T') = \sum_{(x,y)\in A(T'\setminus(f,x'))} c(x,y,\tau(x)) + \sum_{x\in V(T'\setminus\{x'\})} \sum_{t=\alpha(x)}^{\tau(x)-1} c(x,t) + \delta(x', t_s, t)$$

$$= \sum_{(x,y)\in A(T'\setminus(f,x'))} c(x,y,\tau(x)) + \sum_{x\in V(T'\setminus\{x'\})} \sum_{t=\alpha(x)}^{\tau(x)-1} c(x,t) + \delta_E(D', t_s)$$

$$= \zeta(T'') \geq \zeta(T)$$

The last inequality holds since T is the minimum spanning tree of N. A similar analysis can be applied to the case with the arc $(x', g) \in A(T')$.

We now show $\zeta(T) \geq \zeta(T')$. For any spanning tree T^o of N, we can construct T^*, a spanning tree of N', such that $\zeta(T^o) \geq \zeta(T^*)$. Suppose that $T^o_{D'}$ is the spanning tree of D' in T^o with $t_s \leq \tau(f)$. We create T^* by replacing $T^o_{D'}$ by arc (f, x'). Let $\delta(x', t_s, t)$ denote the cost of reaching x' from f with $\tau(f) = t_s$. By the definition, we have $\delta(x', t_s, t) = \delta_E(D', t_s)$, where $\delta_E(D', t_s)$ is the cost of minimum spanning tree of D'. Therefore, we have

$$\zeta(T^o) = \sum_{(x,y)\in A(T^o\setminus T^0_{D'})} c(x,y,\tau(x)) + \sum_{x\in V(T^o\setminus T^0_{D'})} \sum_{t=\alpha(x)}^{\tau(x)-1} c(x,t) + \zeta(T^o_{D'})$$

$$\geq \sum_{(x,y)\in A(T^*\setminus(f,x'))} c(x,y,\tau(x)) + \sum_{x\in V(T^o\setminus T^0_{D'})} \sum_{t=\alpha(x)}^{\tau(x)-1} c(x,t) + \delta(x', t_s, t)$$

$$= \zeta(T^*) \geq \zeta(T').$$

Since for any T^o the inequality is true and T' is the minimum spanning tree of N', we have $\zeta(T) \geq \zeta(T')$. The proof for the case that T^o contains a spanning tree of D can be established similarly.

In summary, we have $\zeta(T') = \zeta(T)$. In other words, we can obtain the minimum spanning tree of N by finding the minimum spanning tree in N'. Notice that N' is still a time-varying arc series-parallel network with $|A(N')| \leq k - 1$ since we replace a diamond by a path and the number of arcs decreases by at least 1. By the induction, we complete the proof. □

Theorem 2.3 *TMST-SP can be implemented in* $O((m+n)m\kappa^2)$ *time.*

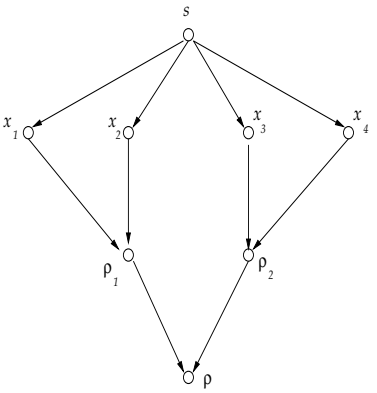
Proof: The time needed for selecting a diamond in N is $O(m+n)$ (see Valdes et al (1982)). To calculate $d_{f,g}(x, t_s, t)$ for all $x \in D$ and for all $0 \leq t_s \leq t \leq \kappa$, we need $O((m+n)\kappa^2)$ time. Both the deleting and

the creating operations require $O(m+n)$ time. Therefore, one iteration (within the *While* loop) needs $O((m+n)\kappa^2)$ time. Since each iteration decreases by at least one arc, we need at most m iterations. Thus, the total running time is bounded above by $O((m+n)m\kappa^2)$. □

Let us now examine an example to illustrate Algorithm TMST-SP.

Example 2.2

Figure 2.3. An example to illustrate Algorithm TMST-SP



(a)

arc	t = 0	t = 1	t = 2	t = 3	t = 4
	b, c	b, c	b, c	b, c	b, c
(s, x_1)	2, 5	1, 3	2, 6		
(s, x_2)	1, 1	2, 1	2, 2		
(s, x_3)	1, 4	1, 3	2, 5		
(s, x_4)	2, 5	2, 3	1, 4		
(x_1, ρ_1)			1, 2	1, 1	1, 3
(x_2, ρ_1)		1, 6	2, 3	1, 4	
(x_3, ρ_2)		1, 2	2, 1	1, 4	
(x_4, ρ_2)			1, 1	2, 1	1, 2
(ρ_1, ρ)			1, 1	1, 1	2, 1
(ρ_2, ρ)			1, 2	1, 2	2, 1

(b)

Consider an ASP network N as shown in Figure 2.3(a). Associated with each arc, there are two numbers, transit time $b(x, y, t)$ and cost $c(x, y, t)$, as listed in Figure 2.3(b) (a blank in the table stands for $b = c = \infty$). All waiting costs $c(x, t) = 0$. Given $\kappa = 4$, the problem is to find the time-varying minimum spanning tree T of N.

First, pick up a diamond $D_1 = (s, x_1, x_2, \rho_1)$. For each $x \in V(D_1)$ and $0 \le t_1 < t_2 \le \kappa$, calculate $d_{s,\rho_1}(x, t_1, t_2)$. For example, $d_{s,\rho_1}(s, 0, 0) = 0$, $d_{s,\rho_1}(x_2, 0, 0) = \infty$, and

$$d_{s,\rho_1}(x_2, 0, 1) = \min\{d_{s,\rho_1}(x_2, 0, 0), d_{s,\rho_1}(s, 0, 0) + c(s, x_2, 0)\}$$

$$= \min\{\infty, 0 + 1\} = 1$$

since there exists $u = 0$ which satisfies $u + b(s, x_2, u) = 1$. The values of $d_{s,\rho_1}(x, t_1, t_2)$ are shown as in Table 2.1, while $\delta_I(D_1, t_1, t_2)$ and $\delta_E(D_1', t_1)$ are given in Table 2.2.

The network N is now converted into a network as shown in Figure 2.4(a). Then, we pick up $D_2 = (s, x_3, x_4, \rho_2)$. For each $x \in V(D_1)$ and

Table 2.1. The values of d_{s,ρ_1} for diamond D_1

$d_{s,\rho_1}(x_1,t_1,t_2)$	$t_2=1$	2	3	4	$d_{s,\rho_1}(x_2,t_1,t_2)$	$t_2=1$	2	3	4
$t_1=0$	∞	3	3	3	$t_1=0$	1	1	1	1
1		3	3	3	1		∞	1	1
2			∞	6	2			∞	2
3				∞	3				∞

$d^{x_1}_{s,\rho_1}(\rho_1,t_1,t_2)$	$t_2=1$	2	3	4	$d^{x_2}_{s,\rho_1}(\rho_1,t_1,t_2)$	$t_2=1$	2	3	4
$t_1=0$	∞	∞	5	4	$t_1=0$	∞	7	7	4
1		∞	5	4	1		∞	∞	5
2			∞	∞	2			∞	∞
3				∞	3				∞

Table 2.2. The values of δ_I and δ_E for diamond D_1

$\delta_I(D_1,t_1,t_2)$	$t_2=1$	2	3	4	t_1	$\delta_E(D_1',t_1)$
$t_1=0$	∞	10	6	5	0	4
1		∞	6	5	1	4
2			∞	∞	2	8
3				∞	3	∞

$0 \le t_1 < t_2 \le \kappa$, calculate $d_{s,\rho_2}(x,t_1,t_2)$, see Table 2.3. The values of $\delta_I(D_2,t_1,t_2)$ and $\delta_E(D_2',t_1)$ are listed in Table 2.4.

A new network is obtained as shown in Figure 2.4(b).

Finally, as the network is a diamond now, we can calculate $\delta_I(D,t_1,t_2)$ for $0 \le t_1 < t_2 \le \kappa$. The results are listed in Table 2.5.

Since $\delta_I(D,0,4) = 14$, we have $\zeta(T) = 14$. The minimum spanning trees T is shown as in Figure 2.5(a). The number in a box associated with the arc is the departure time and the number without a box is the transit cost. Figure 2.5(b) shows a minimum spanning tree of N with $\kappa = 3$.

Table 2.3. The values of d_{s,ρ_2} for diamond D_2

$d_{s,\rho_2}(x_3,t_1,t_2)$	$t_2=1$	2	3	4	$d_{s,\rho_2}(x_4,t_1,t_2)$	$t_2=1$	2	3	4
$t_1=0$	4	3	3	3	$t_1=0$	∞	5	3	3
1		3	3	3	1		∞	3	3
2			∞	5	2			4	4
3				∞	3				∞

$d^{x_3}_{s,\rho_2}(\rho_2,t_1,t_2)$	$t_2=1$	2	3	4	$d^{x_4}_{s,\rho_2}(\rho_2,t_1,t_2)$	$t_2=1$	2	3	4
$t_1=0$	∞	6	6	4	$t_1=0$	∞	∞	6	6
1		∞	∞	4	1		∞	∞	∞
2			∞	∞	2			∞	∞
3				∞	3				∞

Table 2.4. The values of δ_I and δ_E for diamond D_2

$\delta_I(D_2,t_1,t_2)$	$t_2=1$	2	3	4	t_1	$\delta_E(D_2',t_1)$
$t_1=0$	∞	9	9	7	0	6
1		∞	∞	7	1	6
2			∞	∞	2	9
3				∞	3	10

Table 2.5. The values of δ_I for diamond D

$\delta_I(D,t_1,t_2)$	$t_2=1$	2	3	4
$t_1=0$	∞	∞	16	14
1		∞	∞	14
2			∞	∞
3				∞

Figure 2.4. An example to illustrate Algorithm TMST-SP (continued)

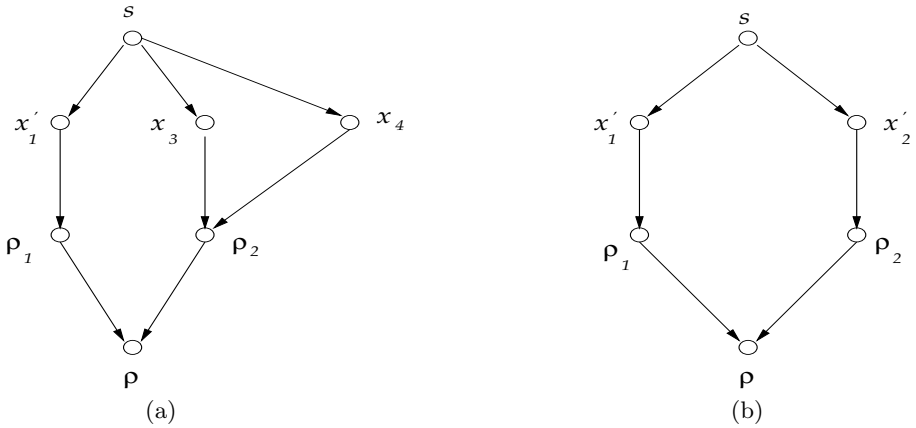
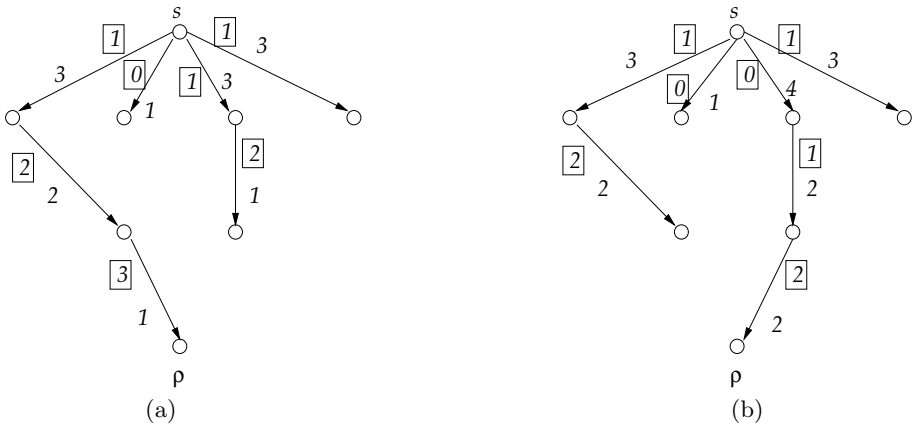


Figure 2.5. The optimal solutions



4. Networks containing no subgraph homomorphic to K_4

In this section, we will generalize the model of Section 3.3 to such networks whose underlying graphs contain no subgraphs homomorphic to K_4 (a complete graph with four vertices).

4.1 Properties and complexity

Liu and Geldmacher (1976) have shown that, any graph with no subgraph homomorphic to K_4 can be recursively transformed, by applying four transformation rules (see Definition 2.7 below), to a single vertex. They have further devised a linear time algorithm that can decide

whether a graph has a subgraph homomorphic to K_4 (Liu and Geldmacher 1980).

Definition 2.7 *Let G' be the resultant graph after applying four transformation rules T_1, T_2, T_3 and T_4 to a graph G until none of the rules can be further applied, where*

T_1: Replace a loop vv with a vertex v.

T_2: Replace a dangling edge uv with a vertex u.

T_3: Replace a pair of series edges uv and vw with an edge uw.

T_4: Replace a pair of parallel edges uv and uv with an edge uv.

If G' consists of only one single vertex, then we say G is reducible. Otherwise, we say G is nonreducible.

Note that in the definition above we follow Liu and Geldmacher (1976) to use the terminology "reducible", which is different from the terminology "reducible" used in the NP-completeness analysis (see Section 3.2).

The following two properties are established in Liu and Geldmacher (1976).

Property 2.1 *If T_1, T_2, T_3 and T_4 are applied to a graph until no longer possible, then a unique graph results, independent of the sequence of application of T_1, T_2, T_3 and T_4.*

Property 2.2 *A graph G is nonreducible if and only if it contains a subgraph homomorphic to K_4.*

Corresponding to the concept of reducible graph, we define the terminology of "reducible network" as follows.

Definition 2.8 *A is a* **reducible network** *if its underlying graph contains no subgraph homomorphic to K_4.*

Note that an edge series-parallel graph (the underlying graph of an arc series-parallel network) is reducible since it contains no subgraph homomorphic to K_4. On the other hand, we do have other networks, whose underlying graph are not edge series-parallel, but which are reducible. Figure 2.6 below is an example.

Example 2.3

An ASP network is a special case of the reducible network. Thus, from Theorem 2.1 and Theorems 2.5 and 2.6 (See below), we have the following Theorem.

Theorem 2.4 *The time-varying spanning tree problem on reducible networks is NP-complete in the ordinary sense.*

Figure 2.6. A reducible graph which is not edge series-parallel

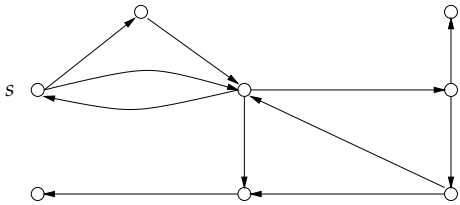
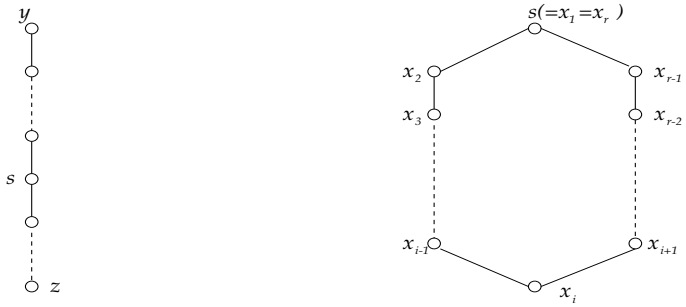


Figure 2.7. Two special cases of reducible networks



4.2 An exact algorithm

Recall that we consider time-varying networks with no parallel arcs, that is, there do not exist two arcs of the same direction between two vertices. It is possible to have, however, two arcs of opposite directions between two vertices. To simplify the presentation in figures, in this section we will use a link to indicate a single arc or a pair of opposite arcs between two vertices.

Let us first examine the following two special cases.

Case I. The network N under consideration is shown in Figure 2.7(a), where s is the source vertex. By Definition 2.5, $d_{s,x}(x, t_s, t)$ is the cost of a shortest path from s to x within the time duration $[t_s, t]$ (Note that if such a path does not exist, then $d_{s,x}(x, t_s, t) = \infty$). Then, we can easily see that

$$\zeta(T(\kappa)) = d_{s,y}(y, 0, \kappa) + d_{s,z}(z, 0, \kappa) \tag{2.1}$$

where $\zeta(T(\kappa))$ is the minimum cost of the spanning tree of N within the time limit κ.

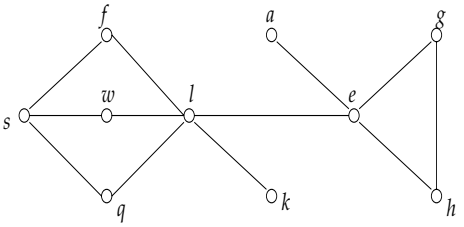
Case II. The network N under consideration is shown in Figure 2.7(b). In this case we have

$$\zeta(T(\kappa)) = \min_{1\le i\le r-1}\{d_{s,x_i}(x_i,0,\kappa) + d_{s,x_{i+1}}(x_{i+1},0,\kappa)\} \qquad (2.2)$$

where $d_{s,x_i}(x_i,0,\kappa)$ is calculated along the path $P(x_1,x_2,...,x_i)$ while $d_{s,x_{i+1}}(x_{i+1},0,\kappa)$ is calculated along the path $P(x_r,x_{r-1},...,x_{i+1})$. Again, note that by default we define $d_{s,x}(x,t_s,t)=\infty$ if no path exists from s to x.

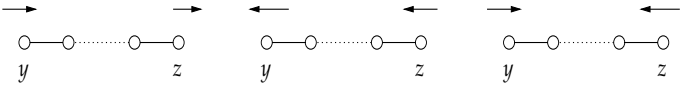
A vertex $v\in V$ is said to be a *conjunction vertex*, if it has more than two adjacent vertices in N. A path $P(x,y)$ (or a cycle $C(x=x_1,...,x_r=x)$) with one conjunction vertex x and $s\notin V(P)\backslash\{x\}$ (or $s\notin V(C)\backslash\{x\}$) is called a *dangling path (or a dangling cycle)* (see Figure 2.8). A cycle with two conjunction vertices is called a *diamond*. Note that this definition is a generalization of Definition 2.4.

Figure 2.8. l and e are conjunction vertices, while $P(a,e)$ is a dangling path, cycle $C(e,g,h)$ is a dangling cycle, and $D(P(s,w,l),P(s,f,l))$ is a diamond.



Both two structures, path and diamond, play the key roles in the algorithm we are to present below. Notice that, if a path contains two conjunction vertices as its two ends and s is not in it, the flow can flow in or out at each conjunction vertex of the path. Figure 2.9 shows these situations.

Figure 2.9. Directions of flow into or out from a path, where y and z are two conjunction vertices



Since for two vertices x and y, $c(x, y, t)$ may not equal $c(y, x, t)$, we need to calculate the cost of a path in terms of three different spanning ways above. This leads to the following definition.

Definition 2.9 *Suppose that $P(y = x_1, x_2, ..., x_r = z)$ is a path in N, where y and z are two conjunction vertices, and $s \notin V(P)\backslash\{y, z\}$. Define*

(1) $d_{y^I,z^O}(t_y, t_z)$ as the cost of the minimum spanning tree of P, such that y is the flow-in vertex, z is the flow-out vertex, $\tau(y) \geq t_y$ and $\alpha(z) \leq t_z$;

(2) $d_{y^O,z^I}(t_y, t_z)$ as the cost of the minimum spanning tree of P, where y is the flow-out vertex, z is the flow-in vertex, $\alpha(y) \leq t_y$ and $\tau(z) \geq t_z$;

(3) $d_{y^I,z^I}(t_y, t_z)$ as the minimum forest of P, where both y and z are flow-in vertices, $\tau(y) \geq t_y$ and $\tau(z) \geq t_z$.

If such trees do not exist, let the relevant cost be ∞.

Recall Definition 2.5 on $d_{f,g}(x, t_s, t)$. The following lemma gives a method to calculate $d_{y^I,z^O}(t_y, t_z)$, $d_{y^O,z^I}(t_y, t_z)$, and $d_{y^I,z^I}(t_y, t_z)$.

Lemma 2.4 *Suppose that $P(y = x_1, x_2, ..., x_r = z)$ is a path in N, where y and z are two conjunction vertices, and $s \notin V(P)\backslash\{y, z\}$. Then, we have*

$$d_{y^I,z^O}(t_y, t_z) = d_{y,z}(z, t_y, t_z), \quad 0 \leq t_y \leq t_z \leq \kappa$$

$$d_{y^O,z^I}(t_y, t_z) = d_{z,y}(y, t_z, t_y), \quad 0 \leq t_z \leq t_y \leq \kappa$$

$$d_{y^I,z^I}(t_y, t_z) = \min_{1 \leq i \leq r-1} \{d_{y,z}(x_i, t_y, \kappa) + d_{z,y}(x_{i+1}, t_z, \kappa)\}, \quad 0 \leq t_y, t_z \leq \kappa.$$

Proof: Straightforward. □

By Lemma 2.4, formulae (2.1) and (2.2) can be rewritten as (2.3) and (2.4) respectively.

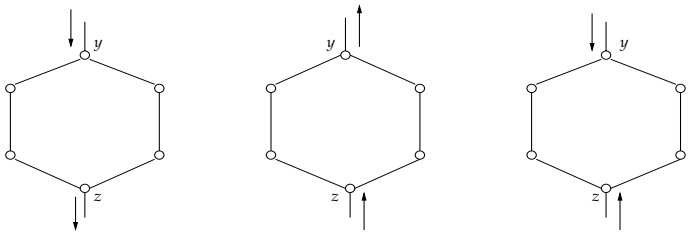
$$\zeta(T^*) = d_{s^I,y^O}(0, \kappa) + d_{s^I,z^O}(0, \kappa), \tag{2.3}$$

$$\zeta(T^*) = \min_{1 \leq i \leq r-1} \{d_{s^I,x_i^O}(0, \kappa) + d_{s^I,x_{i+1}^O}(0, \kappa)\} \tag{2.4}$$

Similarly, if a diamond does not contain the source vertex, the flow can also flow in or out at each conjunction vertex of a diamond. Figure 2.10 shows these situations.

Definition 2.10 *Suppose that D is a diamond in N, where y and z are two conjunction vertices, and $s \notin V(D)\backslash\{y, z\}$. Define $\delta_{y^I,z^O}(t_y, t_z)$ as the cost of the minimum spanning tree of D such that y is the flow-in vertex, z is the flow-out vertex, $\tau(y) \geq t_y$ and $\alpha(z) \leq t_z$. Define $\delta_{y^O,z^I}(t_y, t_z)$ and $\delta_{y^I,z^I}(t_y, t_z)$ in a similar way. If such spanning trees do not exist, let the cost be ∞.*

Figure 2.10. Directions of flow into or out from a diamond, where y and z are two conjunction vertices



Lemma 2.5 *Suppose that $D = (P_1(y,z), P_2(y,z))$ is a diamond which consists of two paths P_1 and P_2, with y and z being the two conjunction vertices, and $s \notin V(D)\backslash\{y,z\}$. Then, we have*

$$\begin{aligned}
\delta_{y^I,z^O}(t_y,t_z) &= \min\{d^1_{y^I,z^O}(t_y,t_z) + d^2_{y^I,z^I}(t_y,t_z), d^2_{y^I,z^O}(t_y,t_z) \\
&\quad + d^1_{y^I,z^I}(t_y,t_z)\}, \qquad 0 \le t_y \le t_z \le \kappa \\
\delta_{y^O,z^I}(t_y,t_z) &= \min\{d^1_{y^O,z^I}(t_y,t_z) + d^2_{y^I,z^I}(t_y,t_z), d^2_{y^O,z^I}(t_y,t_z) \\
&\quad + d^1_{y^I,z^I}(t_y,t_z)\}, \qquad 0 \le t_z \le t_y \le \kappa \\
\delta_{y^I,z^I}(t_y,t_z) &= d^1_{y^I,z^I}(t_y,t_z) + d^2_{y^I,z^I}(t_y,t_z), \qquad 0 \le t_y, t_z \le \kappa
\end{aligned}$$

where d^k denotes the cost for P_k ($k = 1, 2$).

Proof: Straightforward. □

The basic idea of our algorithm is to carry out two operations, one transferring a dangling path or a dangling cycle to a vertex, while the other transferring a diamond to a path.

Operation I Consider a dangling path (or a dangling cycle) of N. Before give the operation, we introduce the following definition first.

Definition 2.11 *Suppose $P(x,...,y)$ be a dangling path in N and x is a conjunction vertex. Define*

$$\eta(x,t) = d_{x^I,y^O}(t,\kappa)$$

where $0 \le t \le \kappa$. In case $y = x$, P becomes a dangling cycle $C(x,...,x)$. Then, define

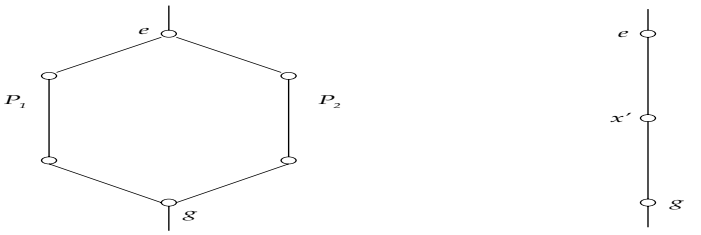
$$\eta(x,t) = d_{x^I,x^I}(t,t).$$

Actually, $\eta(x,t)$ is the cost of the shortest path of P or the cost of the minimum spanning tree of C with departure time t at vertex x. The

operation is to remove the path P from N (except vertex x), and attach $\eta(x,t)$ $(0 \le t \le \kappa)$ to vertex x in N'. For completeness, we let $\eta(z,t) = 0$ $(0 \le t \le \kappa)$ for any $z \in V(N')$ if there does not exist a dangling path or a dangling cycle at z. Furthermore, If there are more than one dangling paths or dangling cycles at vertex x, let $\eta(x,t)$ be the summation of the costs.

Operation II Consider a diamond $D(P_1(e,...,g), P_2(e,...,g))$, where e and g are two conjunction vertices. Now, we remove D in N except vertices e and g, and add an artificial path $P(e,x',g)$ into N to obtain a new network N', where x' is an artificial vertex, ex' and $x'g$ are two artificial edges (see Figure 2.11). Moreover, attach $\delta_{e^I,g^O}(t_e,t_g)$, $\delta_{e^I,g^O}(t_e,t_g)$, and $\delta_{e^I,g^O}(t_e,t_g)$ to vertex x'.

Figure 2.11. A diamond is changed into a path



Notice that, the network N is reducible, therefore after performing these two operations until no longer possible, the resultant network is either a path or a cycle as described in case I and II.

After completing operations, N' contains artificial vertices, artificial edges and extra variables $\eta(x,t)$. Therefore, we need to define the cost of a spanning tree of N' as below:

Definition 2.12 *Suppose N' is a network obtained after the operations, and T' be a spanning tree of N'. Define*

$$\zeta(T') = \sum_{xy \in E(T'), x,y \notin \mathcal{A}(N')} c(x,y,\tau(x)) + \sum_{x \in V(T')} \eta(x,\alpha(x)) + \sum_{x \in \mathcal{A}(N')} \Delta(x)$$

where

$$\Delta(x) = \begin{cases} \delta_{e^I,g^I}(\alpha(e),\alpha(g)), & \text{if } (e,x') \in E(T'), (x',g) \notin E(T'), \\ & \text{or } (x',g) \in E(T'), (e,x') \notin E(T') \\ \delta_{e^I,g^O}(\alpha(e),\tau(g)), & \text{if } (e,x'),(x',g) \in E(T'), \alpha(e) \le \tau(g) \\ \delta_{e^O,g^I}(\tau(e),\alpha(g)), & \text{if } (e,x'),(x',g) \in E(T'), \tau(e) > \alpha(g) \end{cases}$$

e and g are conjunction vertices of D, and $\mathcal{A}(N')$ is the artificial vertex set of N'.

For the case where there are multiple artificial vertices, we can define the cost of a spanning tree in a similar way. Definition 2.12 is the generalization of Definition 2.3. One can see that if N' has no artificial paths and all $\eta = 0$, then Definition 2.12 becomes Definition 2.3.

Furthermore, the formula for calculating $d_{e,g}(x, t_e, t_g)$ and $d_{g,e}(x, t_g, t_e)$ should be revised as follows:

Lemma 2.6 *Let $P(e, ..., x, y, z, ..., g)$ be a path of N where e and g are two conjunction vertices. Then, $d_{e,g}(e, t_e, t) = 0$ for all $0 \le t_e \le t \le \kappa$, $d_{e,g}(y, t_e, t_e) = \infty$ for all $y \ne e$, and, for $0 \le t_e < t \le \kappa$ and $y \ne e$, we have*

(i) if xy is not an artificial edge, then

$$d_{e,g}(y, t_e, t) = \min\{d_{e,g}(y, t_e, t-1), \eta(y,t) + \min_{\{u|u+b(x,y,u)=t\}}\{d_{e,g}(x, t_e, u) + c(x, y, u)\}\},$$

(ii) if xy is an artificial edge, and

(a) if y is an artificial vertex, then

$$d_{e,g}(y, t_e, t) = \min\{d_{e,g}(y, t_e, t-1), \min_{t_e<u<t}\{d_{e,g}(x, t_e, u) + \delta_{x^I,z^O}(u, t)\}\},$$

(b) otherwise, if y is not an artificial vertex, then

$$d_{e,g}(y, t_e, t) = \eta(y,t) + \min\{d_{e,g}(y, t_e, t-1), d_{e,g}(x, t_e, t)\},$$

Similarly, we can compute $d_{g,e}(x, t_g, t_e)$.

Also, Lemma 2.4 should be revised as follows:

Lemma 2.7 *Suppose that $P(y = x_1, x_2, ..., x_r = z)$ is a path in N, where y and z are two conjunction vertices, and $s \notin V(P)\backslash\{y, z\}$. Then, we have*

$$d_{y^I,z^O}(t_y, t_z) = d_{y,z}(z, t_y, t_z), \qquad 0 \le t_y \le t_z \le \kappa$$
$$d_{y^O,z^I}(t_y, t_z) = d_{z,y}(y, t_z, t_y), \qquad 0 \le t_z \le t_y \le \kappa$$
$$d_{y^I,z^I}(t_y, t_z) = \min_{1\le i\le r-1} \xi_i, \qquad 0 \le t_y, t_z \le \kappa$$

where

$$\xi_i = \begin{cases} d_{y,z}(x_i, t_y, \kappa) + d_{z,y}(x_{i+1}, t_z, \kappa), \\ \qquad \text{both } x_i \text{ and } x_{i+1} \text{ are not artificial vertices} \\ \min\limits_{t_y\le u_1\le\kappa, t_z\le u_2\le\kappa}\{d_{y,z}(x_{i-1}, t_y, u_1) + \delta_{x^I_{i-1},x^I_{i+1}}(u_1, \kappa) \\ \qquad + d_{z,y}(x_{i+1}, t_z, \kappa)\}, \quad x_i \text{ is an artificial vertex} \\ \min\limits_{t_y\le u_1\le\kappa, t_z\le u_2\le\kappa}\{d_{y,z}(x_i, t_y, \kappa) + \delta_{x^I_i,x^I_{i+2}}(\kappa, u_2) \\ \qquad + d_{z,y}(x_{i+2}, t_z, u_2)\} \quad x_{i+1} \text{ is an artificial vertex} \end{cases}$$

By Lemma 2.6 and Lemma 2.7, formulae (2.3) and (2.4) now should be rewritten as (2.5) and (2.6) respectively.

$$\zeta(T^*) = d_{s^I,y^O}(0,\kappa) + d_{s^I,z^O}(0,\kappa) + \eta(s,0), \tag{2.5}$$

$$\zeta(T^*) = d_{s^I,s^I}(0,0) + \eta(s,0). \tag{2.6}$$

The basic steps of our algorithm are as follows:

(a) If N is a path $P(y,...,s,...,z)$, then $\zeta(T(\kappa))$ is calculated by formula (2.5) (Assume the initial value of $\eta(x,t)$ is set to zero for each $x \in V$ and $t = 0,1,...,\kappa$).

(b) If N is a cycle $C(s = x_1, x_2, ..., x_r = s)$, then $\zeta(T(\kappa))$ is calculated by formula (2.6).

(c) If N contains a dangling path $P(x,y)$, where x is a conjunction vertex, then let $\eta(x,t) = d_{x^I,y^O}(t,\kappa) + \eta(x,t)$ for $t = 0,1,...,\kappa$, and do Operation I.

(d) If N contains a dangling cycle $C(x = x_1, x_2, ..., x_r = x)$, where x is a conjunction vertex, then for $t = 0,1,...,\kappa$, let $\eta(x,t) = d_{s^I,s^I}(t,t) + \eta(s,t) + \eta(x,t)$, and do Operation I.

(e) Otherwise, choose a diamond $D(P_1(y,z), P_2(y,z))$ in N (if there are more than one diamond, choose one arbitrarily), where y and z are two conjunction vertices of D. According to Lemma 2.5, calculate $\delta_{y^I,z^O}(t_y,t_z)$ for $0 \le t_y \le t_z \le \kappa$, $\delta_{y^O,z^I}(t_y,t_z)$ for $0 \le t_z \le t_y \le \kappa$, and $\delta_{y^I,z^I}(t_y,t_z)$ for $0 \le t_y, t_z \le \kappa$. Perform Operation II.

(f) Still denote the new network as N. Repeat step (c) to (e) until N becomes a path or a cycle.

Now, we are ready to present our algorithm.

Algorithm TMST-RN
Begin
Set $\eta(x,t) = 0$ for each $x \in V(N)$ and for each $0 \le t \le \kappa$;
While N is neither a path nor a cycle with s **do**
Repeat doing Operation I to delete the dangling paths and dangling cycles;
Select arbitrarily a diamond $D(y,z)$ in N;
Calculate $\delta_{y^I,z^O}(t_y,t_z)$, $\delta_{y^O,z^I}(t_y,t_z)$, and $\delta_{y^I,z^I}(t_y,t_z)$ for D and for all $0 \le t_y, t_z \le \kappa$;
Do Operation II;

End *While*;
If N is a path $P(x, ..., s, ..., y)$ **then** $\zeta(T) = d_{s^I,y^O}(0,\kappa) + d_{s^I,z^O}(0,\kappa) + \eta(s,0)$;
If N is a cycle $C(s = x_1, x_2, ..., x_r = s)$ **then** $\zeta(T) = d_{s^I,s^I}(0,0) + \eta(s,0)$;
End.

Theorem 2.5 *TMST-RN can optimally solve the time-varying minimum spanning tree problem on a reducible network.*

Proof: We prove that, when the algorithm is terminated, $\zeta(T)$ is the cost of the minimum spanning tree of N. Use induction on $m = |E(N)|$. Consider the case with $m = 1$. Since there is only one arc, N is a path and the claim holds obviously. Assume that when $m < k$, the claim is true. Now we consider the case with $m = k$.

We examine the following cases:

(1) N is a path or a cycle. By formulae (2.5) and (2.6), we know that $\zeta(T)$ is the cost of the minimum spanning tree of N.

(2) N is neither a path nor a cycle. Then consider the following cases:

(i) N has a dangling path $P(x, y)$, where x is a conjunction vertex. By the algorithm, we delete P (except vertex x) from N and obtain a new network N'. In what follows, we first prove that, for each spanning tree T' of N', it can be extended to a spanning tree T^o of N such that $\zeta(T') = \zeta(T^o)$. Next, we show that for each spanning tree T of N, there is a spanning tree T' of N', such that $\zeta(T') \leq \zeta(T)$. That is to say, finding the minimum spanning tree of N can be equivalently converted to finding the minimum spanning tree of N'. Since $m' = |E(N') < k$, by the induction, $\zeta(T')$ obtained by the algorithm is the cost of minimum spanning tree of N', therefore, it is also the cost of the minimum spanning tree of N.

Suppose T' is a spanning tree of N'. By the definition, we have $\zeta(T') < \infty$. By the definition of $\zeta(T')$, we know that $\sum_{x' \in V(T')} \eta(x', \alpha(x')) < \infty$, or, $\eta(x, \alpha(x)) < \infty$. By the definition of η, we know that there is a spanning tree T'' of $P(x, y)$ with the cost $\eta(x, \alpha(x))$. Thus, we can extend T' by adding T'' to obtain a dynamic spanning tree of N.

Now, we show that $\zeta(T') \leq \zeta(T)$. Notice that P is a dangling path in N, that is to say, all vertices y in P can only be visited from x. Suppose that t^o be the departure time at x in T. We create T' by cutting path P in T except vertex x, and let $\eta(x, t^o)$ be the minimum spanning tree of path P with departure time t^o at vertex x. Then, we

have $\eta(x, t^o) \leq \sum_{uv \in E(P)} c(u, v, \tau(u))$. By the definition, we have

$$\zeta(T') = \sum_{x'y' \in E(T')} c(x', y', \tau(x')) + \sum_{x \in V(T')} \eta(x', \alpha(x'))$$

$$= \sum_{x'y' \in E(T')} c(x', y', \tau(x')) + \sum_{x \in V(T'), x' \neq x} \eta(x', \alpha(x')) + \eta(x, t^o)$$

$$\leq \sum_{x'y' \in E(T')} c(x', y', \tau(x')) + \sum_{x \in V(T'), x' \neq x} \eta(x', \alpha(x')) + \sum_{uv \in E(P)} c(u, v, \tau(u))$$

$$= \zeta(T)$$

Therefore, the claim is proved.

(ii) N has a dangling cycle. The analysis is similar to (i).

(iii) N has neither dangling paths nor dangling cycles. Since N is reducible, there must exist a diamond with two conjunction vertices only (Otherwise, N will contain a subgraph homomorphic to K_4 since the transformation T_4 can not be applied to N).

Select a diamond D in N with two conjunction vertices, say e and g. We compute $\delta_{e^I, g^O}(t_1, t_2)$, $\delta_{e^O, g^I}(t_1, t_2)$, and $\delta_{e^I, g^I}(t_1, t_2)$. Change D to a path $P(e, x', g)$ to obtain a new network N'. Similarly, we first prove that for any spanning tree T' of N", it can be extended to a spanning tree T of N.

Notice that in N', the degree of x' is 2, therefore edge ex' (or $x'g$) must be in T'. Suppose ex' in T' and $x'g \notin T'$. We restore edge ex' and vertex g to a spanning forest F of the diamond D, and add F to T' to obtain a spanning tree T of N. A similar analysis can be applied to the case with $ex', x'g \in E(T')$ or $x'g \in E(T')$ only.

We now show that for any spanning tree T of N, there is a spanning tree T' of N', such that $\zeta(T') \leq \zeta(T)$. Suppose that F is the spanning forest of D in T with $t_e = \tau(e)$, and $t_g = \tau(g)$. We create T' by replacing F by edge ex'. Let $\delta_{e^I, g^I}(t_e, t_g)$ denote the cost of minimum spanning forest of D. Therefore, we have

$$\zeta(T) = \sum_{xy \in E(T) \setminus E(F)} c(x, y, \tau(x)) + \zeta(F)$$

$$\geq \sum_{xy \in E(T') \setminus ex'} c(x, y, \tau(x)) + \delta_{e^I, g^I}(t_e, t_g) = \zeta(T').$$

The proof for the case that T contains a spanning tree of D can be established similarly.

Notice that N' is still a time-varying reducible network with $|E(N')| < k$, since we replace a diamond by a path or replace a path (a cycle) to a

vertex and the number of edges decreases by at least 1. By the induction, we complete the proof.

□

Theorem 2.6 *TMST-RN can be implemented in* $O((m+n)m\kappa^2)$ *time.*

Proof: Setting $\eta(x,t)$ for each vertex $x \in V$ and for each time $\leq t \leq \kappa$ needs $O(n\kappa)$ time. In the while-loop, to find a dangling path or a dangling cycle, we can use depth-first traversal (see Gilberg et al (2001)). This step can be done in $O(m)$ time. The time needed for selecting a diamond in N is $O(m+n)$ (see Valdes et al (1982)). To calculate $\delta_{y^I,z^O}(t_1,t_2)$, $\delta_{y^O,z^I}(t_1,t_2)$, and $\delta_{y^I,z^I}(t_1,t_2)$ for all $0 \leq t_1 \leq t_2 \leq \kappa$, we need $O((m+n)\kappa^2)$ time. Replacing a diamond by an artificial path requires $O(m)$ time. Therefore, one iteration (within the *While* loop) needs $O((m+n)\kappa^2)$ time. Since each iteration decreases at least one edge, we need at most m iterations. Thus, the total running time is bounded above by $O((m+n)m\kappa^2)$. □

5. General networks

We now study the time-varying minimum spanning tree (TMST) problem on a general network. We will first examine its complexity in terms of strong NP-completeness, and then develop algorithms which can find, in pseudopolynomial time, approximate solutions.

5.1 Strong NP-hardness

It is well known that the classical minimum spanning tree problem is polynomially solvable (see, for example, Graham et al (1985)). We have shown in Sections 3 and 4 above that, the time-varying minimum spanning tree (TMST) problem on an arc series-parallel network or a reducible network is, however, NP-complete in the ordinary sense. In this section we will further show that the general TMST problem is NP-complete in the strong sense even if the underlying graph of N is a tree with $b(x,y,t) = b(x,y)$, or $c(x,y,t) = c(x,y)$ for any arc $(x,y) \in A$.

We will show that the *Minimum Set Cover (MSC) problem* is reducible to TMST.

Definition 2.13 MSC: *Given a set* $C = \{C_1, C_2, ..., C_m\}$ *of finite sets and a number* K_s*, does there exist a set cover* C' *such that* $|C'| \leq K_s$*?*

To study its complexity in terms of NP-completeness, we examine the decision version of the TMST problem as stated below.

Definition 2.14 TMST: *Given a time-varying network N and two integers k and κ, does there exist a spanning tree within the time limit κ such that its total cost is not greater than k ?*

We first examine the problem with the constraint that waiting at any vertex is not allowed. Our results are given in Theorem 2.7 and Theorem 2.8 below.

Theorem 2.7 *If waiting at any vertex is not allowed, then TMST is NP-complete in the strong sense, even if the underlying graph of N is a tree, and*

(i) $c(x, y, t)$ are time-varying and $b(x, y, t) = b(x, y)$, $\forall (x, y) \in A$; or

(ii) $b(x, y, t)$ are time-varying and $c(x, y, t) = c(x, y)$, $\forall (x, y) \in A$.

Proof: Clearly, TMST is in NP. Now, we will prove that MSC reduces to TMST with no waiting allowed at any vertices. For each $u \in \bigcup_{i=1}^{m} C_i$, we create a vertex x to represent it in N. Moreover, add a source vertex s and a linking vertex v_0 in N. Use arcs (s, v_0) and (v_0, u) to connect these vertices and let

$$b(s, v_0, \kappa) = \infty, b(s, v_0, t) = 1, t = 0, 1, 2, ..., \kappa - 1$$
$$c(v_0, u, 0) = \infty, c(s, v_0, t) = 1, t = 0, 1, 2, ..., \kappa$$
$$b(v_0, u, t) = \begin{cases} 1 & \text{if } u \in C_t \\ \infty & \text{otherwise} \end{cases}$$
$$t = 1, 2, ..., \kappa$$
$$c(v_0, u, t) = 0, t = 1, 2, ..., \kappa$$

Let $k = K_s$ and $\kappa = m$. Finally, assume that at any $x \in V$, no waiting is allowed.

The network N defined above possesses the property that b are time-varying and c only depend on arc (x, y). Clearly, this reduction can be implemented in polynomial time (see Figure 2.12).

We now prove that a "yes" answer to MSC is equivalent to a "yes" answer to TMST.

If MSC has a set cover C' with $|C'| = l \leq K_s$, then a spanning tree with zero waiting times at any intermediate vertices can be constructed starting with the source vertex s. Without loss of generality, suppose $C_i \in C', 1 \leq i \leq l$ (note that $l \leq K_s \leq m$). Let $C'_1 = C_1$ and $C'_i = C_i \backslash \bigcup_{j=1}^{i-1} C_j$, $2 \leq i \leq l$. Clearly, $C' = \bigcup^i C_i = \bigcup^i C'_i$. For each C'_i, if $|C'_i| \neq 0$, we choose the path $P_i(s, v_0)$ starting at s at time $i - 1$ and arriving at v_0 at time i. If u is the element that occurs in C'_i, we add

Figure 2.12. A network constructed from MSC

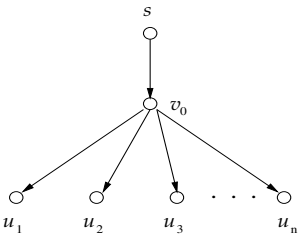
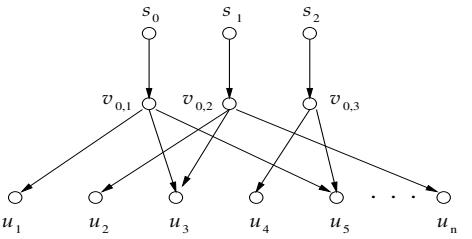


Figure 2.13. The spanning tree is splitted into subtrees



arc (v_0, u) to $P_i(s, v_0)$ with the starting time i and arriving time $i+1$ to form a subtree, denoted by T_i. According to our reduction, each vertex $u \in \bigcup_{i=1}^{m} C_i$ can be reached within time κ. It is obvious that the tree constructed by combining all T_i above is a spanning tree T within time $\kappa = m$ and the total cost is less than $k = K_s$ (see Figure 2.13).

If TMST has a spanning tree T of total cost not exceeding $k = K_s$, then the cost restriction guarantees that there exist less than k subtrees T_i which contain paths from s to each element $u \in \bigcup_{i=1}^{m} C_i$. Choose all elements u in T_i to form the set C_i'. Obviously, $C' = \bigcup_i C_i'$ contains all elements $u \in \bigcup_{i=1}^{m} C_i$, and C' is a set cover with $|C'| \leq k = K_s$.

For the case $b(x, y, t) = b(x, y)$ and c are time-varying, we can modify the reduction as follows:

$$
\begin{aligned}
& b(x, y, t) = 1, \forall (x, y) \in A, 0 \leq t \leq \kappa \\
& c(s, v_0, \kappa) = \infty \\
& c(v_0, u, t) = \begin{cases} 0 & \text{if } u \in C_t \\ \infty & \text{otherwise} \end{cases} \\
& \qquad\qquad\qquad\qquad\qquad\qquad t = 0, 1, ..., \kappa
\end{aligned}
$$

All other analyses remain unchanged. This completes the proof. □

A problem is said to be in the class of APX, if it has a constant-error approximation algorithm, i.e., an algorithm that can find, in polynomial time, an approximate solution with an error bound β, where $\beta > 1$ is a fixed constant (Note that the error bound is defined as follows: Let ζ^* and ζ^0 be the optimal and approximate solutions, respectively. Then, ζ^0 is said to have an error bound β if $\zeta^0/\zeta^* \leq \beta$). Lund and Yannakakis (1993) indicate that MSC is not in the class of APX; i.e., to find a constant-factor approximation algorithm for MSC is at least as hard as to prove P=NP. The reduction we constructed above implies that the TMST problem is also not in the class of APX. This gives us the following result.

Theorem 2.8 *Consider the TMST problem with no waiting allowed at any vertex. There is no constant-error polynomial-time approximation algorithm for the TMST problem unless P=NP, even if the underlying graph of N is a tree, waiting time at any vertex must be zero, and one of the two parameters, b or c, is time independent.*

We now consider the situation where waiting at any vertex is arbitrarily allowed (Namely, waiting at any vertex is subject to no constraints). We will establish its NP-completeness using another reduction from MSC. Note that in the case where waiting at any vertex is prohibited, we can show its NP-completeness even when the underlying graph of N is a tree. This is however not extendible to the case with waiting at any vertex being arbitrarily allowed. The reduction analysis is now built on a network that is not a rooted tree; see below.

Theorem 2.9 *The TMST problem where waiting at any vertex is arbitrarily allowed is NP-complete in the strong sense, even if $c(x, y, t) = c(x, y)$ and $b(x, y, t) = b(x, y)$, $\forall (x, y) \in A$.*

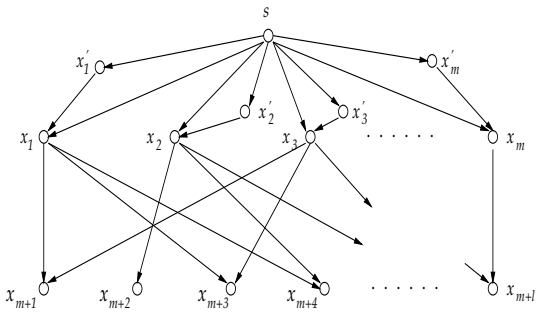
Proof: For any given instance of MSC, we construct an instance of TMST as follows: For each $C_i \in C$, create a pair of vertices x_i and x'_i $(1 \le i \le m)$. For each element $u_j \in \bigcup C_i$, create a vertex x_{m+j}. These vertices together with an extra vertex, the source vertex s, compose the vertex set V. Create arcs (s, x_i), (s, x'_i), and (x'_i, x_i), $1 \le i \le m$, as shown in Figure 2.14. For each vertex x_{m+j}, create an arc (x_i, x_{m+j}) if $u_j \in C_i$ (see Figure 2.14). These arcs compose the arc set A. Let

$$b(s, x_i, t) = b(s, x'_i, t) = b(x'_i, x_i, t) = b(x_i, x_{m+j}, t) = 1,$$
$$1 \le i \le m, 1 \le j \le n, 0 \le t \le \kappa$$
$$c(s, x_i, t) = 1, c(s, x'_i, t) = c(x'_i, x_i, t) = 0, 1 \le i \le m, 0 \le t \le \kappa$$
$$c(x_i, x_{m+j}, t) = 0, 1 \le i \le m, 1 \le j \le n, 0 \le t \le \kappa$$

Finally, let $k = K_s$, $\kappa = 2$. The network $N(V, A, b, c)$ as defined above has the property that both c and b are time independent (Note that we do not have any restriction on the waiting time at any vertex). In what follows, we prove that a "yes" answer to MSC is equivalent to a "yes" answer to TMST.

If MSC has a set cover C' with $|C'| = l \le K_s$, then a spanning tree with waiting times at its vertices can be constructed as follows: Without loss of generality, assume $C_i \in C'$, $1 \le i \le l$ (note that $l \le K_s \le m$), and let $C'_1 = C_1$ and $C'_i = C_i \backslash \bigcup_{j=1}^{i-1} C_j$, $2 \le i \le l$. For each C'_i, if $|C'_i| \ne 0$, we choose the path $P_i(s, x_i)$ with $\tau(s) = 0$ and $\alpha(x_i) = 1$. If u_j is the element in C'_i, we add arc (x_i, x_{m+j}) to $P_i(s, x_i)$ with $\tau(x_i) = 1$

Figure 2.14. A time-varying network created from MSC



and $\alpha(x_i) = 2$. When there are more than one arc added to $P_i(s, x_i)$, we have a subtree, denoted by T_i. Since $\cup_{i=1}^{l} C_i = \cup_{i=1}^{l} C_i' = \cup_{i=1}^{m} C_i$, each vertex x_{m+j} can be reached within time 2. For those $C_i \notin C'$, we choose the path $P(s, x_i) = (s, x_i', x_i)$ with $\tau(s) = 0$, $\alpha(x_i') = \tau(x_i') = 1$ and $\alpha(x_i) = 2$. It is obvious that all vertices in V can be reached within time 2. Combining all T_i and P_i' we obtain a spanning tree T within time $\kappa = 2$. Since the cost of T_i is equal to 1 and the cost of P_i' is 0, we have $\zeta(T) \leq l \leq k = K_s$.

Now, if TMST has a spanning tree $T(2)$ with $\zeta(T(2)) \leq k \leq K_s$, then the cost and transit time in the problem constructed above guarantee that there exist $l \leq k$ subtrees, which contain path(s) from s to each vertices x_{m+j} with $\alpha(x_{m+j}) = 2$. Choose those sets C_i if x_{m+j} appears in T_i. Let $C' = \bigcup_{i=1}^{l} C_i$. Since $T(2)$ contains all vertices x_{m+j}, we have all elements $u_j \in \bigcup_{i=1}^{l} C_i$. Thus C' is a set cover with $|C'| \leq k = K_s$. In summary, we complete the proof. □

Again, from the definition of the class of APX, we have:

Theorem 2.10 *Consider the TMST problem where waiting at any vertex is arbitrarily allowed. There is no constant-error polynomial-time approximation algorithm for the problem unless P=NP, even if for any arc $(x, y) \in A$, $b(x, y, t) = b(x, y)$ and $c(x, y, t) = c(x, y)$.*

Finally, we consider the general TMST problem where waiting time at any vertex may be constrained by an arbitrary function. Since this contains the situation where waiting time at a vertex must be zero as its special case, Theorem 2.11 and Theorem 2.12 below follow immediately from Theorem 2.7 and Theorem 2.8, respectively.

Theorem 2.11 *The general TMST problem is NP-complete in the strong sense, even if the underlying graph of N is a tree and*

(i) $c(x, y, t)$ are time-varying and $b(x, y, t) = b(x, y)$, $\forall (x, y) \in A$; or

(ii) $b(x, y, t)$ *are time-varying and* $c(x, y, t) = c(x, y)$, $\forall (x, y) \in A$.

Theorem 2.12 *There is no constant-error polynomial-time approximation algorithm for the general TMST problem unless P=NP, even if the underlying graph of* N *is a tree and one of the two parameters,* b *or* c*, is time independent.*

5.2 Heuristic algorithms

We now describe our algorithms that can find approximate solutions in pseudopolynomial time. Basically, our algorithms consist of two main steps: Firstly, for each vertex $x \in V$, we identify the shortest path from s to x of time at most κ. Putting all these paths together we obtain a path-induced subnetwork of N. Then, we remove redundant arcs from this subnetwork so as to obtain a spanning tree of N. We will show that the resulted tree, denoted as $T_A(\kappa)$, is an approximate solution for the general TMST problem, while the total time required to construct $T_A(\kappa)$ is pseudopolynomial.

In order to apply the algorithms in Chapter 1 to find the shortest paths, in the following we will limit the problem to be solved to the one where waiting time at vertex x, for $x \in V$, is bounded above by u_x. Note that when $u_x = 0$ for all $x \in V$, we have the case with no waiting at any vertex being allowed, while if $u_x = \infty$ for all $x \in V$, we have the case where waiting at x is arbitrarily allowed.

5.2.1 Finding a shortest path

The definition below defines the return function of the dynamic program to be introduced below.

Definition 2.17 *Let* $d_b(y, t)$ *be the cost of a shortest path from* s *to* y *of time exactly* t*. If such a path does not exist, let* $d_b(y, t) = \infty$.

We rewrite Algorithm TSP-BW as a procedure below:

Procedure DP
Begin
Initialize: $d_b(s, 0) := 0$, and $d_b(x, 0) := \infty$, $\forall x \neq s$; $d_b(x, t) := \infty$, $\forall x$ and $\forall t > 0$; $Heap_x := \{d_b(x, 0)\}$ and $d_b^m(x, 0) := d_b(x, 0)$, $\forall x$;
Sort all values $u + b(x, y, u)$ for all $u = 0, 1, ..., \kappa$ and for all arcs $(x, y) \in A$;
For $t = 1, ..., \kappa$ **do**
  For every arc $(x, y) \in A$ **do** $\mathcal{R}_b(x, y, t) := \infty$;
  For all arcs $(x, y) \in A$ and all u_D such that $u_D + b(x, y, u_D) = t$ **do**

$\mathcal{R}_b(x,y,t) := \min\{\mathcal{R}_b(x,y,t), d_b^m(x,u_D) + c(x,y,u_D)\}$;
For every vertex y **do** $d_b(y,t) := \min_{\{x|(x,y)\in A\}} \mathcal{R}_b(x,y,t)$;
For every vertex y **update** the heap as follows
Insert-heap$_{(y)}$ $d_b(y,t)$;
If $t > u_y$ **then delete-heap**$_{(y)}$ $d_b(y, t-u_y-1)$;
For every vertex y **do**
$u_A :=$ **Minimum-heap**$_{(y)}$;
$d_b^m(y,t) := d_b(y,u_A)$;
For every y **do** $d_b^*(y) := \min_{0\le t\le\kappa} d_b(y,t)$;
End.

5.2.2 Removing redundant arcs

After applying the algorithm above to find the shortest paths between the root s and each $x \in V$, we can obtain a path-induced subnetwork of N by combining all these paths together. The next main step of our approach is to remove those redundant arcs from the path-induced subnetwork. We will perform the following vertex/arc deleting operations:

(i) *Deleting shared intermediate vertices.* If a vertex x appears in two paths P_i and P_j, and if $[\alpha_j(x), \tau_j(x)] \subseteq [\alpha_i(x), \alpha_i(x) + u_x]$, then arc (y,x), where y is the predecessor of x in P_j, is redundant, since all successors of x in P_j can be reached from s through the section $P_i(s,x)$ in path P_i and the waiting time constraint is not violated. Therefore we can delete arc (y,x). Then, the original path P_j ends at the vertex y, and a new path is created which consists of two sections: the section $P_i(s,x)$ in path P_i, and the section starting from x in the original path P_j with the departure time $\tau_j(x)$. Repeat this operation until there are no shared intermediate vertices. Then go to Operation (ii) next.

(ii) *Deleting redundant end vertices.* Let x_j^o denote the end vertex of path P_j. If x_j^o appears in another path, or in P_j as an intermediate vertex, then delete the end vertex x_j^o in P_j as well as its adjacent arc, and the predecessor of x_j^o in P_j becomes the new end vertex. Repeat this operation until the whole path is eliminated or the end vertex of P_j does not appear in any other paths or in P_j as an intermediate vertex.

By Definition 2.2, the path-induced subnetwork N' generated after performing Operations (i) and (ii) above is a spanning tree. In what follows, we will use two procedures, DSIV (Deleting Shared Intermediate Vertices) and DREV (Deleting Redundant End Vertices) to realize these two operations respectively.

(1) Procedure DSIV

Procedure DSIV is used to delete the shared intermediate vertices among paths. To reduce the time requirement, the procedure uses a 3-dimensional array $e(x,t,i)$. If vertex x appears in path P_j with arrival time $\alpha_j(x)$ and departure time $\tau_j(x)$, then $e(x,t,1)=j$ and $e(x,t,2)=\tau_j(x)$, where $t=\alpha_j(x)$. Initially, they are set to zero. The procedure contains the following two basic steps:

(i) If there are more than one path which include x with the same arrival time $\alpha(x)$, then keep the path P_{j^0} that has the latest departure time $\tau_{j^0}(x)$. Delete all arcs (y_i,x) in path P_i, where y_i is the predecessor of x in path P_i and $i\neq j^0$.

(ii) Check array e. For each vertex x, if there exist u and t such that $t<u$ and $e(x,u,2)\leq t+u_x$ (this means that there are two paths P_i and P_j with $e(x,t,1)=i$, $e(x,u,1)=j$, $t=\alpha_i(x)$, $u=\alpha_j(x)$ and $e(x,u,2)=\tau_j(x)$, which satisfies $[\alpha_j(x),\tau_j(x)]\subseteq[\alpha_i(x),\alpha_i(x)+u_x]$), then arc (y_j,x) can be deleted in P_j.
After completing these two steps, all shared intermediate vertices will be deleted.

Procedure DSIV
Begin
For $x\in V\backslash\{s\}$ and $t=0,1,...,\kappa$ **do** $e(x,t,1):=e(x,t,2):=0$;
For each path P_j $(j=1,...,n-1)$ and each $x\in V(P_j)$ **do**
If $e(x,\alpha(x),1)=0$ **then** $e(x,\alpha(x),1):=j$, $e(x,\alpha(x),2):=\tau(x)$;
Else If $\tau(x)\leq e(x,\alpha(x),2)$ **then** delete arc (y_j,x) in P_j;
Else let $i:=e(x,\alpha(x),1)$, delete arc (y_i,x) in P_i, $e(x,\alpha(x),1):=j$, $e(x,\alpha(x),2):=\tau(x)$;
For each $x\in V\backslash\{s\}$ **do**
Let $\alpha:=0$;
For $t=0,1,...,\kappa$ **do**
If $e(x,t,1)=0$ **then** $\alpha:=t$;
Else If $e(x,t,2)\leq\alpha+u_x$ **then** let $i:=e(x,\alpha(x),1)$, delete arc (y_i,x) in P_i, $\alpha_i(x):=\alpha$, $e(x,t,1):=e(x,t,2):=0$;
End.

The time complexity of the procedure can be analyzed as follows. The first and the third For-do loops need $O(n\kappa)$ time. The second loop also takes $O(n\kappa)$ time since each path P_j contains at most κ vertices. Therefore, the total running time of the procedure is bounded by $O(n\kappa)$.

(2) Procedure DREV

This procedure is used to delete the redundant end vertices of the paths. The basic idea is to set up a counter $num(x)$ for each vertex x to

record the number of its occurrences in all paths. Then, when we check whether an end vertex x of a path appears in another place, we only need to check $num(x)$. If $num(x) > 1$, it is clear that x must appear in another place and so we can delete it from the path and then decrease $num(x)$ by 1.

Procedure DREV
Begin
$num(0) := 0$;
For each $x \in V\backslash\{s\}$ **do** $num(x) := 0$;
For each $x \in V\backslash\{s\}$ and $t = 0, 1, ..., \kappa$ **do**
If $e(x, t, 1) > 0$ **then** $num(x) := num(x) + 1$;
For each path P_j **do**
Identify the end vertex x_j^o of P_j;
While $num(x_j^o) > 1$ **do**
Let $num(x_j^o) := num(x_j^o) - 1$;
Delete x_j^o in P_j;
End *while*;
End.

The procedure DREV checks each vertex in all paths. Since there are at most $n-1$ paths (because there are at most $n-1$ end vertices) in $T_A(\kappa)$ and each path contains at most κ vertices (because the transit time b is a positive integer and the arrival time at a vertex must be greater than that at its predecessor), the procedure DREV needs at most $O(n\kappa)$ time.

5.2.3 The algorithm A-TMST

Our algorithm can be described as follows.

Algorithm A-TMST
Begin
Call Procedure DP to obtain the shortest path $P_j(s, x)$, from s to each vertex $x \in V\backslash\{s\}$, for $1 \leq j \leq n-1$;
If there exists a path $P(s, x)$ with $d^*(x) = \infty$ **then** let $\zeta(T_A(\kappa)) := \infty$ and stop;
Call procedure DSIV to delete the shared intermediate vertices;
Call procedure DREV to delete the redundant end vertices;
Combine all paths that remain to generate an path-induced subnetwork $T_A(\kappa)$;
End.

An approximate solution that has an error bound f is called *an f-approximate solution*. Let L denote all leaves in $T_A(\kappa)$ obtained by A-TMST and $l = |L|$. Then, we have

Theorem 2.13 *Algorithm A-TMST can find, in at most* $O(\kappa(m + n\log\kappa))$ *time, an* l*-approximate solution for TMST. Moreover,* l *is the best possible bound for the algorithm A-TMST.*

Proof: Clearly, $T_A(\kappa)$ obtained by the algorithm is a spanning tree of N within time κ, according to Definition 2.2.

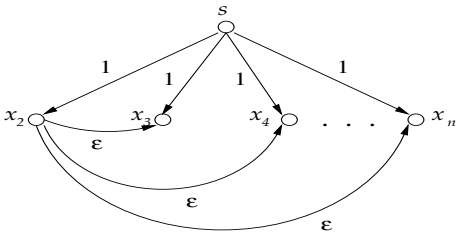
Let us first consider the time requirement of A-TMST. The running time required by Procedure DP is $O(\kappa(m + n\log\kappa))$. Since the procedures DSIV and DREV each need only $O(n\kappa)$ time, the total running time of A-TMST is thus bounded above by $O(\kappa(m + n\log\kappa))$.

We now analyze the error bound of the approximate solution $T_A(\kappa)$. Recall that l is the number of leaves in $T_A(\kappa)$. It is not hard to see that

$$\zeta(T_A(\kappa)) \leq \sum_{x\in L} \zeta(P^*(s,x)) \leq \sum_{x\in L} \zeta(P(s,x)) \leq l\zeta(T(\kappa)),$$

where $T(\kappa)$ is the optimal spanning tree and $P(s,x)$ is the path in $T(\kappa)$ from s to x, while $P^*(s,x)$ is the shortest path from s to x obtained by Procedure DP.

Figure 2.15. A network for error bound analysis



To show that the bound l is the best achievable by Algorithm A-TMST, we consider the following special class of the time-varying networks N, and illustrate that the approximate solution obtained by applying Algorithm A-TMST on such networks will achieve the bound l. Let $N_\varepsilon = (V_\varepsilon, A_\varepsilon, b_\varepsilon, c_\varepsilon)$ (see Figure 2.15), where $V_\epsilon = \{x_1(=s), x_2, ..., x_n\}$, $A_\varepsilon = \{(s,x_i), 2 \leq i \leq n, (x_2,x_j), 3 \leq j \leq n\}$, and

$$b_\varepsilon(s,x_i,t) = 1, c_\varepsilon(s,x_i,t) = 1, 2 \leq i \leq n, 0 \leq t \leq \kappa$$
$$b_\varepsilon(x_2,x_j,t) = \varepsilon, c_\varepsilon(x_2,x_j,t) = 1, 3 \leq j \leq n, 0 \leq t \leq \kappa.$$

Note that when $\kappa \geq 2$, the optimal spanning tree $T(\kappa)$ is Figure 2.16 with $\zeta(T(\kappa)) = 1 + (n-2)\varepsilon$. The solution of A-TMST, $T_A(\kappa)$, is Figure

Figure 2.16. The optimal solution

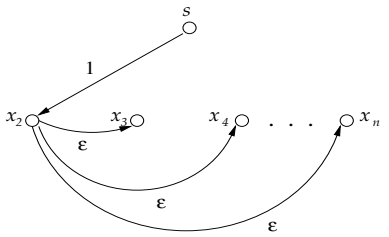
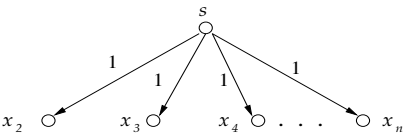


Figure 2.17. The solution obtained by A-TMST



2.17 with $\zeta(T_A(\kappa)) = n - 1$. Letting ε tend to zero, we have

$$\frac{\zeta(T_A(\kappa))}{\zeta(T(\kappa))} = \frac{n-1}{1+(n-2)\varepsilon} \stackrel{\varepsilon \to 0}{\longrightarrow} n - 1 = l$$

where l is the number of leaves in $T_A(\kappa)$. Therefore, we complete the proof. □

Remark. Note that the time complexity of the algorithm A-TMST is dominated by that of the procedure to find shortest paths.

As for the error bound of the algorithm A-TMST, can we further improve it? Theorem 2.13 indicates that the bound l is the best possible, in general. Nevertheless, in some special situations where certain structure/conditions are satisfied, we may get a better error bound. In the following we discuss such a case.

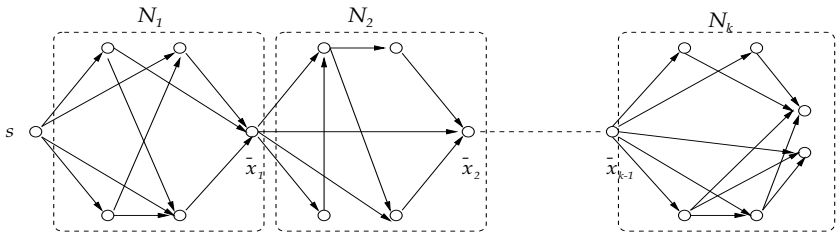
5.3 The error bound of the heuristic algorithms in a special case

We will show here that the error bound of the algorithm A-TMST can be improved for a type of multi-period networks. Such a network possesses a multi-period structure, with each period starting from a common source vertex; see Figure 2.18 (Note that the parameters, such as the transit times and the transit costs on the arcs, can be time varying).

Let N_i denote the sub-network of a period in N, $1 \leq i \leq k$; see Figure 2.18. Furthermore, let $L(N_i)$ denote the leaves of $T_A(N_i)$, where $T_A(N_i)$ is the sub-tree of $T_A(\kappa)$ covering N_i, and let $l_i = |L(N_i)|$ $(1 \leq i \leq k)$. Then, we have

Corollary 2.1 *The approximate solution obtained by the algorithm A-TMST has an error bound $f = max\{l_1, l_2, ..., l_k\}$, if N is a job scheduling network and*

Figure 2.18. A network with multi-period structure



(i) waiting at any vertex is allowed without any restriction, and
(ii) both $b(x, y, t)$ and $c(x, y, t)$ are nonincreasing over time t.

Proof: We use the same notation as in the proof of Theorem 2.13. First, we will show the following inequalities:

$$\sum_{x \in L(N_i)} \zeta(P^*(\bar{x}_{i-1}, x)) \leq \sum_{x \in L(N_i)} \zeta(P(\bar{x}_{i-1}, x)), \quad \text{for } 2 \leq i \leq k. \quad (2.1)$$

Note that, due to the structure of the network, each path $P^*(s, x)$ and $P(s, x)$, $x \in V(N_i)$, must pass through the vertex $\bar{x}_{i-1}$ $(2 \leq i \leq k)$. Moreover, since waiting at a vertex is allowed, we claim that each path $P^*(s, x)$ and $P(s, x)$ $(x \in L(N_i))$ must contain the same subpath $P^*(s, \bar{x}_{i-1})$ and $P(s, \bar{x}_{i-1})$ $(2 \leq i \leq k)$. Otherwise, suppose that there are two paths, say $P(s, x_1)$ and $P(s, x_2)$, which have different subpaths $P_1(s, \bar{x}_{i-1})$ and $P_2(s, \bar{x}_{i-1})$, where x_1 and $x_2 \in L(N_i)$. Then, let $\alpha_1(\bar{x}_{i-1})$ and $\alpha_2(\bar{x}_{i-1})$ be the arrival times at vertex $\bar{x}_{i-1}$ on $P(s, x_1)$ and $P(s, x_2)$, respectively. Without loss of generality, suppose $\alpha_1(\bar{x}_{i-1}) < \alpha_2(\bar{x}_{i-1})$. Then we can delete the arc $(y, \bar{x}_{i-1})$ in $P_2(s, \bar{x}_{i-1})$, where y is the predecessor of $\bar{x}_{i-1}$ in P_2. Denote the new tree as $T^0(\kappa)$. Clearly, we have $\zeta(T^0(\kappa)) < \zeta(T(\kappa))$. This contradicts the fact that $\zeta(T(\kappa))$ is optimal.

Now, let $\alpha^*(\bar{x}_{i-1})$ and $\alpha(\bar{x}_{i-1})$ be the arrival times at vertex $\bar{x}_{i-1}$ in $P^*(s, x)$ and $P(s, x)$, respectively. Consider two cases:

(i) $\alpha^*(\bar{x}_{i-1}) \leq \alpha(\bar{x}_{i-1})$. The inequalities (1) hold clearly since any shortest path $P^*(\bar{x}_{i-1}, x)$ starting from time $\alpha^*(\bar{x}_{i-1})$ is shorter than that starting from time $\alpha(\bar{x}_{i-1})$, and also shorter than the path $P(\bar{x}_{i-1}, x)$.

(ii) $\alpha^*(\bar{x}_{i-1}) > \alpha(\bar{x}_{i-1})$. Because both $b(x, y, t)$ and $c(x, y, t)$ are nonincreasing functions of time t, any shortest path $P^*(\bar{x}_{i-1}), x)$ starting from time $\alpha^*(\bar{x}_{i-1})$ is shorter than that starting from time $\alpha(\bar{x}_{i-1})$, and also shorter than path $P(\bar{x}_{i-1}), x)$.

In summary, we prove the inequalities (1). Now, it follows from (1) that

$$\zeta(T_A(\kappa))$$

$$\leq \sum_{x\in L(N_1)} \zeta(P^*(s,x)) + \sum_{x\in L(N_2)} \zeta(P^*(\bar{x}_1,x)) + ... + \sum_{x\in L(N_k)} \zeta(P^*(\bar{x}_{k-1},x))$$

$$\leq \sum_{x\in L(N_1)} \zeta(P(s,x)) + \sum_{x\in L(N_2)} \zeta(P(\bar{x}_1,x)) + ... + \sum_{x\in L(N_k)} \zeta(P(\bar{x}_{k-1},x))$$

$$\leq l_1\zeta(T_{N_1}(\kappa)) + l_2\zeta(T_{N_2}(\kappa)) + ... + l_k\zeta(T_{N_k}(\kappa))$$

$$\leq \max\{l_1, ..., l_k\}\zeta(T(\kappa))$$

where $T_{N_i}(\kappa)$ is the sub-spanning tree of $T(\kappa)$ on sub-network N_i, $1 \leq i \leq k$. That is,

$$\frac{\zeta(T_A(\kappa))}{\zeta(T(\kappa))} \leq \max\{l_1, ..., l_k\}$$

Therefore, we complete the proof. □

5.4 An approximation scheme for the problem with arbitrary waiting constraints

For the time-varying minimum spanning tree problem on the general network but under arbitrary waiting time constraints, we have another approximate scheme, which can solve the problem more efficiency. The scheme consists of two main steps: Firstly, for a given general time-varying network, create a spanning graph of N, denoted by N', which contains no subgraph homomorphic to K_4. Then, we apply Algorithm TMST-RN on N'. Let T' be the optimal solution obtained by the algorithm. Clearly, T' is an approximate solution of the original network N. Both of these two steps can be implemented in pseudopolynomial time.

5.4.1 Creating a spanning reducible network

Remind that $A(N)$ is the arc set of the original network N and s is the root of N. The basic idea of creating the spanning reducible network is:

(i) Let Q be a vertex set. Initially, set $Q = \{s\}$. Denote $dt(x)$ as the earliest possible departure time at x. Let $dt(s) = 0$.

(ii) Pick up a vertex, say x, from Q. Let $adj(x) = \{(x,y)|(x,y) \in A(N), (x,y)$ is unchecked$\}$, which denotes the all unchecked adjacent arcs of x, and sort it in nonincreasing order in terms of the value of $c(x,y,dt(x))$. Do the following repeatedly till $adj(x)$ becomes empty:

(a) Pick up the first arc (x,y) in $adj(x)$ and add it in N'. Check whether N' contains a subgraph homomorphic to K_4 or not. If the answer is "yes", remove (x,y) from N'; Otherwise, leave (x,y) in N' (still called the new network obtained as N'). If $y \notin Q$ then let $Q = Q + y$.

(b) Delete (x,y) from $adj(x)$ and (x,y) is said to be checked.

(iii) Do (ii) repeatedly till Q becomes empty.

Clearly, N' is a spanning reducible network of N with edges as many as possible.

Now we give the algorithm as below.

Algorithm TMST-A
Begin
Set $E(N') = \emptyset$ and $V(N') = \emptyset$. Let $Q = \{s\}$, $dt(s) = 0$;
While $Q \neq \emptyset$ **do**;
Select a vertex x from Q;
Sort $adj(x)$ in nonincreasing order on the value of $c(x, y, dt(x))$;
While $adj(x) \neq \emptyset$ **do**;
Pick up the first edge $(x, y) \in adj(x)$;
If N' contains a subgraph homomorphic to K_4, then discard (x, y);
Else let $A(N') = A(N') + (x, y)$ and $V(N') = V(N') + y$ if $y \notin V(N')$;
Let $adj(x) = adj(x) \backslash (x, y)$;
End *while*;
Let $Q = Q \backslash x$;
End *while*;
End

Clearly, N' obtained by the algorithm is a reducible network of N.

Theorem 2.14 *Algorithm TMST-A can be implemented in* $O(m \cdot \max\{m, n\})$.

Proof: The initialization can be done in constant time. Since Q will contain at most n vertices, the first "while" loop will be performed in at most n times. Sorting $adj(x)$ needs $O(d(x) \log d(x)) = O(d(x) \log m)$, where $d(x)$ is the degree of vertex x. Since we need do this step for all vertices in N, the total number of performances is $\sum_{x \in V(N)} d(x) \log m \leq m \log m$. During the second "while" loop, checking whether N' contains a subgraph homomorphic to K_4 needs $O(\max\{m, n\})$ (see Liu and Geldmacher 1980). As we need do this for all edges in $adj(x)$ and for all vertices x, it needs $O(m \cdot \max\{m, n\})$ time. In summary, the total running time of Algorithm TMST-A is bounded above by $O(m \cdot \max\{m, n\})$. □

5.4.2 Numerical experiments

We test the approximate algorithm on PC. Table 2.6 illustrate our numerical results. The size of problem is listed on the first column with the number of vertices n, the number of edges m, and the time duration T we considered, respectively. All the transit time b, cost c

are generated randomly. In the right column of Table 2.6, m' is the number of spanning reducible network created on the original network by Algorithm TMST-A. For example, the problem 1 has 30 vertices, 80 edges with time duration $T = 30$. The edges of the spanning reducible network N' is 38. The cost of the minimum spanning tree of N' is 31, and use 2 seconds CPU time. As a comparison, we apply Monte Carlo Method to the same problem. Generate 1000 minimum spanning trees for the original network N and choose the best one, which has the cost 70 and costs 170 seconds CPU time. All numerical experimental results show that our approximate algorithm is much batter than Monte Carlo Method.

Table 2.6. The numerical experimental results

Problem size			Monte Carlo method			Approximate algorithm		
n	m	T	repeating times	cost	CPU (sec.)	m'	cost	CPU (sec.)
30	80	30	1000	70	170	38	31	2
50	120	50	5000	113	454	62	49	5
80	260	60	5000	202	515	98	80	36
100	300	80	5000	265	631	127	99	69
150	400	100	5000	394	948	176	149	204
200	500	100	5000	527	1282	228	202	366

6. Additional references and comments

Applications of MST models have been extensively studied in the literature. These include physical systems design (Prim (1957); Loberman et al (1957); Dijkstra (1959)), network design (Magnanti et al (1984)), optimal message passing (Abdel-Wahab et al (1997); Prim (1957)), pattern classification (Dude et al (1973)), image processing (Osteen et al (1974); Xu et al (1997)), and network reliability analysis (Van Slyke et al (1972)). More references on applications of the MST problem can be found in Graham et al (1985). Efficient algorithms for the MST problem include those proposed by Kruskal (1956); Prim (1957), and Dijkstra (1959). The directed MST problem can be solved by an algorithm proposed by Edmonds (1965).

Solomom (1986) considers the situation where there is a time window associated with each vertex. A transit time is needed to traverse an arc, and any vertex must be visited within its time window. The problem is to find a minimum spanning tree to cover all the vertices under the

time-window constraints. Solomom proves that this problem is NP-hard, and presents a greedy algorithm and an insertion algorithm, which may generate approximate solutions for the problem. By appropriating setting the parameters in our time-varying MST model, we can show that Solomom's problem is a special case under our framework.

Springer

http://www.springer.com/978-0-387-71214-7

Time-Varying Network Optimization
Sha, D.; Wong, C.K.
2007, XVI, 248 p. 63 illus., Hardcover
ISBN: 978-0-387-71214-7