Earliest Arrival Flows with Time-Dependent Data*

Horst W. Hamacher †, Stevanus A. Tjandra
Fachbereich Mathematik, Universität Kaiserslautern, Germany

Abstract

In this paper we discuss an earliest arrival flow problem of a network having arc travel times and capacities that vary with time over a finite time horizon T. We also consider the possibility to wait (or park) at a node before departing on outgoing arc. This waiting is bounded by the value of maximum waiting time and the node capacity which also vary with time. We give a pseudo-polynomial time $\mathcal{O}(nm^2T^3U)$ algorithm to find such a flow, where U is the largest capacities, n is the number of nodes, and m is the number of arcs. We also show that the worst-case complexity decreases by a factor $\mathcal{O}(T)$ when infinite waiting is allowed. Finally we report the computational results of our algorithm.

Keywords: Network flows; Earliest arrival augmenting path; Dynamic cut

1 Introduction

Classical (static) network flow models have been well known as valuable tools for many applications (see e.g. Ahuja, Magnanti, and Orlin [1]). However, they fail to capture the dynamic property of many routing problems, for instance, transportation, production plan, and evacuation problems, as described in a survey by Aronson [2]. To tackle this problem, Ford and Fulkerson [5, 6] introduced flows which takes time, called as travel time, to pass an arc of the network. These flows are called dynamic flows or flows over time and determine the number of flow units entering the arc at each point in time. The travel times and arc capacities are assumed to be constant. Ford and Fulkerson [5, 6] gave a polynomial time algorithm to solve the maximum dynamic flow problem. The objective of maximum dynamic flow problem is to send as much flow as possible to the sink for a given time horizon T.

Gale [7] introduced a variant of maximum dynamic flow problem that seeks

*This research has been supported by Deutsche Forschungsgemeinschaft (DFG) grant spp-1126 "Algorithmik Grosser und Komplexer Netzwerke"

†Tel.: +49 631 205 2267; Fax: +49 631 ; E-mail: hamacher@mathematik.uni-kl.de

a dynamic flow which is maximum not only for T, but also for every time $T' < T$. Naturally, this problem is harder than the maximum dynamic flow problem, though both problems share the same constraints. Concerning the application of dynamic flow to the evacuation problem, this earliest arrival flow gives a better evacuation plan than the maximum dynamic flow, since it pushes more people to reach the safety as early as possible. An overview on the interrelation of dynamic network optimization and evacuation modeling can be found in Hamacher and Tjandra [9].

Minieka [12] and Wilkinson [15] showed that the earliest arrival flow exists, and they both also provide pseudo-polynomial time algorithms to find this flow. Their algorithms work on the assumption that the network has constant travel times and capacities. There is no known polynomial time algorithm to solve the earliest arrival flow problem (Fleischer [3]). Under the same assumption of network data, Hoppe and Tardos [10] developed the first polynomial-time approximation algorithm, with time complexity $\mathcal{O}(\frac{m}{\epsilon}(m + n \log n)\log U)$, where U is the largest capacities, n is the number of nodes, and m is the number of arcs. It is proved to be within $(1+\epsilon)$ of optimality, i.e. if $\mathbf{x}$ is a dynamic flow for the time horizon T provided by the algorithm and $V_{\sum^{T'}}$ is the value of earliest arrival flow for any time horizon $T' \leq T$, then $V_{\sum^{T'}} \leq (1+\epsilon)V_{\sum^{T'}}(\mathbf{x})$. This algorithm is a *capacity scaling shortest augmenting path* algorithm applied to the static network. The capacity scaling is done in an *upward* direction, which is opposite of the usual capacity scaling algorithm (see e.g. Ahuja, Magnanti, and Orlin [1]). Instead of using path decomposition as in the *temporally repeated flow* technique (see e.g. Ford and Fulkerson [6]), Hoppe and Tardos used the chain decomposition which allowing to use some backward arcs.

In the continuous-time environment, the flow represents the rate at which some commodity entering the arc at each point in time. Several results dealing with the earliest arrival flow problem in this context can be mentioned here. Ogier [13] worked on the earliest arrival flow problem on the continuous-time dynamic network of having zero travel times. He assumed that the arc and node capacities are piecewise-constant function on interval $[0, T]$ with at most k *breakpoints* (i.e. the points of time at which the value of function changes). Under such assumption, Ogier proved that the earliest arrival flow has at most nk breakpoints. These breakpoints can be computed with nk series of static maximum flow computations on the static network with nk nodes and $(m+n)k$ arcs. The desired flow is then obtained by combining together the maximum flows with respect to each breakpoint. This process needs additional $\mathcal{O}(nk)$ series of static maximum flow computations, each on a network with n nodes. Thus, the overal complexity is determined by the time to solve nk series of static maximum flow problems on the static network with nk nodes and $(m + n)k$ arcs. Fleischer [3] improved Ogier's algorithm by using a generalization of parametric maximum flow algorithm of Gallo, Grigoriadis, and Tarjan [8]. The complexity is improved to $\mathcal{O}(k^2 mn \log(kn^2/m))$. Fleischer and Skutella [4] generalized the earliest arrival flow problem with constant data by considering multiple sources.

In this paper we generalize the earliest arrival flow problem by considering the time dependence of the network data. We will develop an algorithm solving the earliest arrival flow problem when the network has time-dependent travel times and arc capacities (TdEAFP). We also consider the possibility to wait (or park) at a node before departing on outgoing arc. This waiting is bounded by the value of maximum waiting time and the node capacity which also vary with time. In the next section, we formally introduce TdEAFP. In Section 3 we extend the ideas of the residual network in the case of static network (see e.g. Ahuja, Magnanti, and Orlin [1]) to the dynamic case. In Section 4 we work out the details of the algorithm. The finite waiting assumption is relaxed to infinite waiting in Section 5, where a faster algorithm is obtained. An illustrative example is given in Section 6. Computational results on several examples on randomly generated networks in Section 7 conclude this paper.

2 The Time-Dependent Earliest Arrival Flow Problem

A discrete-time dynamic network $G = (N, A, T)$ is a directed graph, where N is the set of nodes, A is the set of directed arcs, and T is a finite time horizon of interest discretized into the set $\{0, \ldots, T\}$. The dynamic network G is assumed to have only a single source s and a single sink d. For simplicity, we assume that no arc enters the source s and no arc leaves the sink d. Moreover, G is *antisymmetric*, i.e. $(i,j) \in A \Rightarrow (j,i) \notin A$. Each arc $(i,j) \in A$ has a time-dependent capacity $u_{ij}(t) \in \mathbb{R}_0^+$ and an associated time-dependent travel time $\lambda_{ij}(t) \in \mathbb{Z}_0^+$. The capacity $u_{ij}(t)$ defines the maximum number of flow units that can enter arc (i,j) at time t. The travel time $\lambda_{ij}(t)$ defines the time period needed to traverse the arc (i,j), departing from node i at time t. This travel time is defined upon entering an arc, and is assumed to be constant for the duration of travel along that arc. This model of travel time is known as a *frozen arc model*, see Orda and Rom [14]. Waiting in any node $i \in N - \{s,d\}$ at any time $t \in \{0, \ldots, T-1\}$ is described by $w_i(t)$, the maximum allowable waiting time, and $a_i(t) \in \mathbb{R}_0^+$, the corresponding waiting capacity. This capacity defines the maximum number of flow units that can be held over one time unit at node i. The value of $w_i(t)$ is decided by the current time t and not by the arrival time at the node. Therefore, it has a value of either zero or one.

Definition 2.1 *A discrete-time dynamic network flow* $\mathbf{x}$ *(also known as* flow over time*) over a time horizon* $T \in \mathbb{Z}_0^+$ *is given by the function*

$$\mathbf{x} : (A \cup \{(i,i) \ : \ i \in N\}) \times \{0, \ldots, T\} \to \mathbb{R}_0^+$$

For any $t \in \{0, \ldots, T\}$, the value $x_{ij}(t)$ determines the number of flow units entering arc (i,j) at time t. Since we are interested to find the flow distribution only for the time horizon T, we bound the time t by $t + \lambda_{ij}(t) \leq T$. The flow $x_{ij}(t)$ is bounded above by the arc capacity $u_{ij}(t)$, i.e.

$$0 \leq x_{ij}(t) \quad \leq \quad u_{ij}(t), \ (i,j) \in A, \ t + \lambda_{ij}(t) \leq T \tag{1}$$

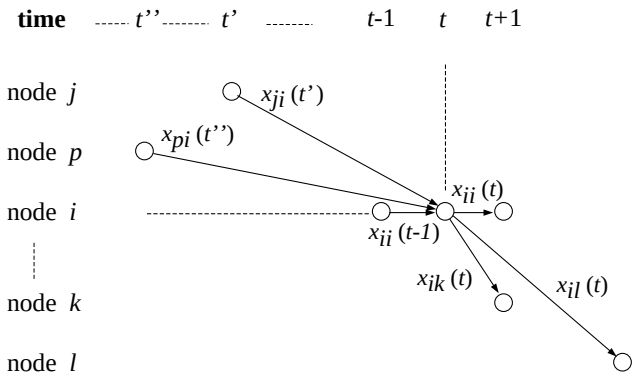


Figure 1: Conservation flows

Flow from node i at time t to the same node with travel time $\lambda_{ii}(t) = 1$ represents the amount of holdover flows. This flow is denoted by $x_{ii}(t)$ and bounded above by the node capacity $a_i(t)$.

$$0 \leq x_{ii}(t) \quad \leq \quad a_i(t) w i_i(t),\ i \in N - \{s, d\},\ t = 0, \ldots, T \tag{2}$$

The value of the flow arriving at a given node i at a given time t is obtained by summing flows from all predecessor nodes of i over all possible departure times t' in which the sum of departure time and travel time equals t, as shown in Figure 1. The dynamic network flow conservation constraint is thus formulized by

$$\sum_{(j,i)\in A} \sum_{\{t':t'+\lambda_{ji}(t')=t\}} x_{ji}(t') - \sum_{\{(i,j)\,:\,(i,j)\in A,\ t+\lambda_{ij}(t)\leq T\}} x_{ij}(t) = x_{ii}(t) - x_{ii}(t-1),\ i \in N - \{s, d\},\ t = 0, \ldots, T \tag{3}$$

To describe the earliest arrival flow, we need the notion of a maximum dynamic flow. Given a time horizon T, the time-dependent maximum dynamic network flow problem (TdMDNFP) maximizes the dynamic flows reaching the sink. It is formulized as

$$\text{(TdMDNFP)} \quad \max\ V_{\sum^T}(\mathbf{x}) \quad := \quad \sum_{t=0}^{T} \sum_{(i,d)\in A} \sum_{\{t':t'+\lambda_{id}(t')=t\}} x_{id}(t') \tag{4}$$

$$\text{Subject to} \qquad (1) - (3)$$

Here, $V_{\sum^T}(\mathbf{x})$ denotes *the value of a discrete-time dynamic flow* $\mathbf{x}$ *for the time horizon* T. We denote by $V_{\sum^T}$ *the value of maximum dynamic flow for a time horizon* T.

Now, we are ready to describe what the *discrete-time earliest arrival flow problem* (TdEAFP) problem is.

Definition 2.2 *A discrete-time earliest arrival flow for the time horizon T is a dynamic flow in which as many flow as possible arrive at the sink during any time horizon $T^{'}$, for every $T^{'} \leq T$.*

By Definition 2.2, the objective function of TdEAFP is to maximize $V_{\sum^{T'}}(\mathbf{x})$ for every $T^{'} = 0, \ldots, T$. We denote by $V_{\sum^{T' \leq T}}$ *the value of discrete-time earliest arrival flow for a time horizon* T. Furthermore, the set of feasible solutions of TdEAFP is defined exactly the same as that of TdMDNFP.

By Definition 2.2, every earliest arrival flow is a maximum dynamic flow, but the converse is not true. In Figure 2, the dynamic flow indicated by the solid lines is a maximum dynamic flow for $T = 7$. But it is not an earliest arrival flow, since its value for $T^{'} = 3$ is zero while $V_{\sum^{T'=3}} = 1$.

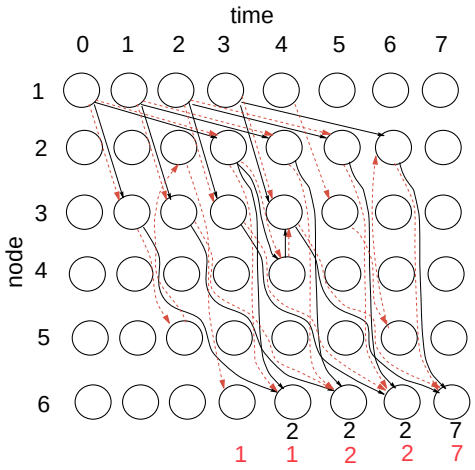


Figure 2: Discrete-time maximum dynamic flow vs discrete-time earliest arrival flow. The solid lines indicate the maximum dynamic flow without having the earliest arrival property, and the dashed lines indicate the earliest arrival flow

3 The Residual Dynamic Network

Here we extend the ideas of the residual network of the static network (see e.g. Ahuja, Magnanti, and Orlin [1]) to the dynamic one. Suppose that arc (i, j) at time t carries $x_{ij}(t)$ units of flow. Then we can send an additional $u_{ij}(t) - x_{ij}(t)$ units of flow departing from node i at time t to node j along arc (i, j). Also we can send up to $x_{ij}(t)$ units of flow from node j departing at time $t + \lambda_{ij}(t)$ and consequently arriving at node i at time t over the arc (i, j), which amounts to canceling the existing flow on the arc. Here we employ arc with negative travel time (i.e. departing at $t + \lambda_{ij}(t)$ and arriving at t) to permit the return of capacity to an arc.

The similar ideas can be applied for the waiting times and capacities. Suppose that $x_{ii}(t)$ units occupy node i at time t for at least one unit time. Then the additional $a_i(t) - x_{ii}(t)$ units can wait and at most $x_{ii}(t)$ units can cancel their waiting, at node i at time t. The waiting canceling at node i at time t is modeled by using a negative waiting time at node i at time $t+1$ and denoted by $w_i^{x-}(t+1)$. Since the waiting canceling at node i at time t may occur only when $w_i(t)$ is positive, $w_i^{x-}(t+1)$ has value -1 or zero. The capacity of waiting canceling is denoted by $a_i^{x-}(t+1)$. Moreover, we denote by $a_i^{x+}(t)$ the maximum free waiting space at node i at time t.

Using these ideas, the residual dynamic network with respect to the current dynamic flow $\mathbf{x}$ is defined as follows.

Definition 3.1 *The residual network with respect to a given feasible dynamic flow* $\mathbf{x}$ *is defined as* $G_x := (N, A_x, T)$ *with* $A_x := A_x^+ \cup A_x^-$ *where*

$$A_x^+ := \{(i,j) : (i,j) \in A,\ \exists t \leq T - \lambda_{ij}(t) \text{ with } u_{ij}(t) > x_{ij}(t)\} \quad (5)$$

and

$$A_x^- := \{(i,j) : (j,i) \in A,\ \exists t \leq T - \lambda_{ji}(t) \text{ with } x_{ji}(t) > 0\} \quad (6)$$

G_x *is provided with the set of parameters, namely:*

- *the residual travel times*

$$\lambda_{ij}^x(t) := \begin{cases} \lambda_{ij}(t) & ,(i,j) \in A,\ t + \lambda_{ij}(t) \leq T \\ -\lambda_{ji}(t') & ,(j,i) \in A,\ t' + \lambda_{ji}(t') = t \leq T,\ x_{ji}(t') > 0 \end{cases} \quad (7)$$

- *the residual arc capacities*

$$u_{ij}^x(t) := \begin{cases} u_{ij}(t) - x_{ij}(t) & ,(i,j) \in A,\ t + \lambda_{ij}(t) \leq T \\ x_{ji}(t') & ,(j,i) \in A,\ t' + \lambda_{ji}(t') = t \leq T \end{cases} \quad (8)$$

- *the residual positive and negative waiting times*

$$w_i^{x+}(t) := w_i(t),\ i \in N - \{s,d\},\ t \leq T, \quad (9)$$

$$w_i^{x-}(t+1) := -w_i(t),\ i \in N - \{s,d\},\ t < T \quad (10)$$

- *the residual waiting capacities*

$$a_i^{x+}(t) := a_i(t) - x_{ii}(t),\ i \in N - \{s,d\},\ t \leq T, \quad (11)$$

$$a_i^{x-}(t+1) := x_{ii}(t),\ i \in N - \{s,d\},\ t < T, \quad (12)$$

Since the travel times are time-dependent, there may exist an arrival time t at node j which corresponds to several departure times $t_1, \ldots, t_k$ from node i along the arc $(i,j) \in A$, i.e. $t' + \lambda_{ij}(t') = t,\ \forall t' \in \{t_1, \ldots, t_k\}$. Therefore, at such an arrival time t, the corresponding backward arc (j,i) of (i,j) has k different negative travel times $\lambda_{ji}^x(t)$ as shown in Figure 3.

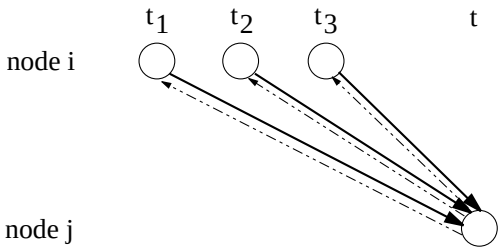


Figure 3: A single arrival time associates with several departure times. The dashed lines correspond to the backward arcs

4 Solution Algorithm for Earliest Arrival Flow Problem with Time-Dependent Data

To solve TdEAFP with time-dependent data, we adapt a well known successive shortest augmenting path technique for solving the static maximum flow problem (see e.g. Ahuja, Magnanti, and Orlin [1]). Instead of looking for a shortest $s-d$ augmenting path in the residual network, we look for an $s-d$ augmenting path with the earliest arrival time at node d. Therefore, we call this modified technique as the *successive earliest arrival augmenting path algorithm.*

We define a dynamic augmenting path as follows.

Definition 4.1 (Dynamic augmenting path)

- *A dynamic augmenting path is a dynamic $s-d$ path $P_{sd}(t_1)$ in G_x composed by a sequence of node-time pairs (NTPs) from node s to node d that is ready at node s at time $t_1 \in \{0,\dots,T\}$, as given by*

$$\begin{aligned} P_{sd}(t_1) \quad = \quad & \{s = j_1(t_1, t_1^{'}), j_2(t_2, t_2^{'}), \dots, d = j_l(t_l, t_l^{'})\}, \\ & t_k, t_k^{'} \in \{0,\dots,T\},\ k = 1,\dots,l \end{aligned} \tag{13}$$

  where $t_{k+1} = t_k^{'} + \lambda^x_{j_k j_{k+1}}(t_k^{'}),\ k = 1,\dots,l-1$ and define $t_l = t_l^{'}$.
  For any NTP $j_k(t_k, t_k^{'})$, the first time parameter t_k denotes the ready time at node j_k, and the second one $t_k^{'}$ denotes the departure time from node j_k. The ready time t_k also defines the arrival time at node j_k from the previous node j_{k-1}, for $k = 2,\dots,l$.

- *The residual capacity $\epsilon(P)$ of $P_{sd}(t_1)$ is the minimum value between the minimum residual capacity of arcs in $P_{sd}(t_1)$ and the minimum residual waiting capacity of any waiting node in $P_{sd}(t_1)$ denoted by $\epsilon_u(P)$ and $\epsilon_a(P)$, respectively, i.e.*

$$\epsilon_u(P) \quad := \quad \min_{1\le k\le l-1}\{u^x_{j_k j_{k+1}}(t_k^{'})\} \tag{14}$$

$$\epsilon_a(P) \quad := \quad \min_{1\le k\le l-1}\{\min_{t_k\le t^{''}\le t_k^{'}-1}\{a^{x+}_{j_k}(t^{''})\}, \min_{t_k^{'}+1\le t^{''}\le t_k}\{a^{x-}_{j_k}(t^{''})\}\} \tag{15}$$

$$\epsilon(P) \quad := \quad \min\{\epsilon_u(P), \epsilon_a(P)\} \tag{16}$$

If $\{t'' \ : \ t_k \le t'' \le t'_k - 1\} = \emptyset$, *then we define*

$$\min_{t_k \le t'' \le t'_k - 1} \{a^{x+}_{j_k}(t'')\} := \infty$$

Also if $\{t'' \ : \ t'_k + 1 \le t'' \le t_k\} = \emptyset$, *then*

$$\min_{t'_k + 1 \le t'' \le t_k} \{a^{x-}_{j_k}(t'')\} := \infty$$

Figure 4 illustrates Definition 4.1. The new flow distribution $\mathbf{x}$ is computed as

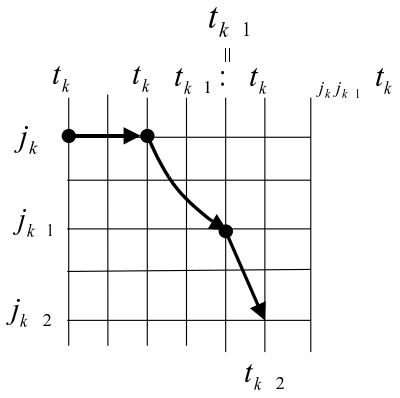


Figure 4: Dynamic augmenting path

follows.

$$x_{ij}(\bar{t}) \ := \ \begin{cases} x_{ij}(\bar{t}) + \epsilon(P) & , \ i \neq j, \ i(t, \bar{t}), \ j(\bar{t} + \lambda^x_{ij}(\bar{t}), t'') \in P \\ x_{ij}(\bar{t}) - \epsilon(P) & , \ i \neq j, \ j(t, \bar{t} + \lambda^x_{ij}(\bar{t})), \ i(\bar{t}, t'') \in P \\ x_{ii}(\bar{t}) + \epsilon(P) & , \ i = j, \ t \le \bar{t} < t', \ i(t, t') \in P \\ x_{ii}(\bar{t}) - \epsilon(P) & , \ i = j, \ t' < \bar{t} \le t, \ i(t, t') \in P \\ x_{ij}(\bar{t}) & , \text{ otherwise} \end{cases} ,$$

$$\forall \ (i,j) \in A \cup \{(i,i) \ : \ i \in N - \{s,d\}\}, \ \bar{t} \in \{0, 1, \ldots, T\} \tag{17}$$

We denote $\mathbf{x} = \mathbf{x} \pm \epsilon(P)$ the dynamic flow given by (17).

The successive earliest arrival augmenting path algorithm always augment flow along an $s-d$ path having the earliest arrival time at node d in the residual network. Since the residual network may have backward arcs whose associated arc travel times are negative valued, a time-dependent label correcting algorithm, called the EAAP (Earliest Arrival Augmenting Path) algorithm, is used to find an $s - d$ earliest arrival augmenting path. This algorithm uses the so-called *scan eligible* (SE) list that stores the nodes which have potential of improving the arrival time of at least one other node. During the initialization step of the algorithm, only the source node s is in the SE list. At each iteration of the algorithm, a node, called the *current node*, is removed from the SE list. We denote by

- π_i, the earliest arrival time at node $i \in N$,
- $pred_i(t),\ t \in \{0, \ldots, T\}$, the predecessor node of node i along an $s-i$ augmenting path that arrives at node i at time t, and
- $dep_i(t),\ t \in \{0, \ldots, T\}$, the departure time from node $pred_i(t)$ corresponding to an arrival time t at node i along an $s-i$ augmenting path.

Suppose that node i is the current node. For every successor nodes j of node i, a temporary label is computed through the corresponding $s-i$ augmenting path and arc (i,j). If it is possible to build an $s-j$ augmenting path departing from i at time t and arriving at j at time $t+\lambda_{ij}(t)$ which is not possible previously via another path (i.e. previously recorded $pred_j(t+\lambda_{ij}(t)) = \infty$), then the value of $pred_j(t+\lambda_{ij}(t))$ is updated from ∞ to i. Because any $s-d$ augmenting path that uses this $s-j$ path as a subpath may lead to a lower earliest arrival time at the sink node d, this node j will enter the SE list. Moreover, $t+\lambda_{ij}(t)$ will update the value of π_j if it is earlier than π_j. The labeling process of node j is then continued by considering both positive and negative waiting allowance at node j. The labeling at node j is done for $t' = t+\lambda_{ij}(t)+1, t+\lambda_{ij}(t)+2, \ldots, T$ as long as waiting is allowed (i.e. $w_j^{x+}(t'-1) > 0$ and $a_j^{x+}(t'-1) > 0$) and $pred_j(t') = \infty$. Otherwise, this labeling process at node j is stopped. By considering that the waiting canceling at node j may lead to a better decision, the labeling of node j is also continued for $t' = t+\lambda_{ij}(t)-1, t+\lambda_{ij}(t)-2, \ldots, 0$. The labeling process on this stage is stopped when it meets t' that does not satisfy the waiting conditions $w_j^{x-}(t'+1) < 0$ and $a_j^{x-}(t'+1) > 0$ or $pred_j(t') \neq \infty$. Once all the successor nodes of the current node have been considered, another current node is selected from the SE list, triggering the next iteration of EAAP algorithm. The algorithm stops once an iteration has completed and the SE list is empty. If $\pi_d \leq T$, then the corresponding $s-d$ earliest augmenting path P and maximum flow augmentation $\epsilon(P)$ can be obtained by backtracking procedure. Otherwise, the current dynamic flow $\mathbf{x}$ is optimal (see Proposition 4.4 later on). The pseudocode of EAAP algorithm is given in Algorithm 4.1.

Algorithm 4.1 works well when G_x does not contain any negative cycle. However, since the original network G has no negative travel times, adding some arcs with negative travel times to create G_x, will not create any negative cycle. The following proposition describes this property.

Proposition 4.1 *Given a dynamic network G that does not contain any negative cycle, its associated G_x will also not contain any negative cycle*

Proof :
Suppose that there is a negative cycle in G_x with respect to the travel times. Since the original network G does not have negative travel times, the cycle must use some backward arcs. Supppose that the cycle begins at node j at time t, uses a backward arc (j,i) departing at time t, and reaching node i at time $t+\lambda_{ji}(t)$. By construction of backward arc, there must exists, in previous iteration, an augmenting path that uses arc (i,j) departing from node i at time t' such that $t'+\lambda_{ij}(t') = t$ with $\lambda_{ij}(t') > 0$. Since $\lambda_{ji}(t) = -\lambda_{ij}(t')$, we obtain $t+\lambda_{ji}(t) = t'$.

Algorithm 4.1 (Earliest Arrival Augmenting Path Algorithm) : finding an $s-d$ earliest arrival augmenting path

INPUT The residual network $G_x := (N, A_x^+ \cup A_x^-, T)$ as given by Definition 3.1
OUTPUT The $s-d$ earliest arrival augmenting path P and $\epsilon(P)$

0 Set $SE := \{s\}$ and define the initial labels for each node $i \in N$:

$$\pi_i := \begin{cases} 0 & , \ i = s \\ \infty & , \ \text{otherwise} \end{cases}$$

and for all $t \in \{0, \ldots, T\}$

$$pred_i(t) := \begin{cases} -1 & , \ i = s \\ \infty & , \ \text{otherwise} \end{cases} \quad ; \ dep_i(t) := \begin{cases} t & , \ i = s \\ \infty & , \ \text{otherwise} \end{cases}$$

1 Select the current node. If $SE = \emptyset$ then go to step 3.
Otherwise, select $i \in SE$ and set $SE := SE - \{i\}$.

2 Scan current node and update the labels
For all $(i,j) \in A_x$ do {
For all $t \in \{t : \pi_i \le t \le T, \ u_{ij}^x(t) > 0, \ t + \lambda_{ij}(t) \le T, \ pred_i(t) \ne \infty\}$
do { If $(pred_j(t + \lambda_{ij}(t)) = \infty)$ then
{ If $(\pi_j > t + \lambda_{ij}(t))$ then $\pi_j := t + \lambda_{ij}(t)$
$pred_j(t + \lambda_{ij}(t)) := i \quad ; \quad dep_j(t + \lambda_{ij}(t)) := t$
$SE := SE + \{j\}$
Define $t' := t + \lambda_{ij}(t) + 1$
While $(t' \le T, \ w_j^{x+}(t'-1) > 0, \ a_j^{x+}(t'-1) > 0,$
and $pred_j(t') = \infty)$
do $\{pred_j(t') = j \quad ; \quad dep_j(t') := t' - 1 \ ; \ t'++\}$
Define $t' := t + \lambda_{ij}(t) - 1$
While $(t' \ge 0, \ w_j^{x-}(t'+1) < 0, \ a_j^{x-}(t'+1) > 0,$
and $pred_j(t') = \infty)$
do { if $(\pi_j > t'$ then $\pi_j := t'$
$pred_j(t') = j \quad ; \quad dep_j(t') := t' + 1 \ ; \ t'--\}$
} } } Return to step 1

3 Constructing the $s-d$ earliest arrival augmenting path.
If $\pi_d > T$ then $P := \emptyset$ and $\epsilon(P) = 0$
Else { $j := d \ ; \ t := \pi_j \ ; \ i := pred_j(t)$
Define $P := \{d(t,t)\}$ and $\epsilon(P) := \infty$
While $(j \ne -1)$ { $t' := dep_j(t)$
If $(i \ne j)$ then { $cap := u_{ij}^x(t') \ ; \ t'' := t'$ }
Else if $(t > t')$, then $cap := a_i^{x+}(t')$
Else $cap := a_i^{x-}(t')$
If $(cap < \epsilon(P))$ then $\epsilon(P) := cap$
$i' := i \ ; \ j := i \ ; \ i := pred_j(t') \ ; \ t := t'$
If $(i' \ne i)$ then $P := P + \{i'(t', t'')\}$ }
}

Suppose without loss of generality, the cycle continues to reach node k at time $t' + \lambda_{ik}(t')$, and back to node j from k at time $t' + \lambda_{ik}(t') + \lambda_{kj}(t' + \lambda_{ik}(t'))$. The total travel time to complete one cycle is $-\lambda_{ij}(t') + \lambda_{ik}(t') + \lambda_{kj}(t' + \lambda_{ik}(t'))$. Since this is a negative cycle, $-\lambda_{ij}(t') + \lambda_{ik}(t') + \lambda_{kj}(t' + \lambda_{ik}(t')) < 0$, i.e.

$$\lambda_{ik}(t') + \lambda_{kj}(t' + \lambda_{ik}(t')) < \lambda_{ij}(t') \tag{18}$$

Since in the previous iteration the augmenting path uses (i,j) at time t', direct connection from i departing at time t' to j must be not longer than uses arcs (i,k) and (k,j), i.e.

$$\lambda_{ik}(t') + \lambda_{kj}(t' + \lambda_{ik}(t')) \geq \lambda_{ij}(t')$$

contradicting (18). ■

To analyze the complexity of Algorithm 4.1, we divide its execution into *passes*. We define a pass as follows.

Definition 4.2

- *Pass* 0 *ends after node* s *is scanned for the first time.*
- *Pass* k *ends after all nodes in the SE-list at the end of pass* $k-1$ *have been scanned.*

From this definition, if a node j is removed from the SE list before the end of pass k, then there must be a node i with $(i,j) \in A_x$ removed from the list before the end of pass $(k-1)$ and some t, $\pi_i \leq t \leq T$ such that $pred_j(t + \lambda_{ij}(t))$ is improved from ∞ to i. In this condition, π_j may also be improved. Proposition 4.1 implies that G_x does not contain any negative cycle. Since G_x has at most $n(T+1)$ node-time pairs, there exists at most $n(T+1)-1$ passes.

Proposition 4.2 *Algorithm 4.1 terminates with the earliest arrival augmenting paths from the source node* s *to all other nodes in* N*, where waiting is limited. It is a pseudopolynomial algorithm with running time* $\mathcal{O}(nmT^2)$.

Proof :
Let us denote by $PRED_i$ the set $\{t \ : \ pred_i(t) \neq \infty, \ t \leq T\}$. When Algorithm 4.1 terminates, SE is empty and $\pi_j \leq t + \lambda_{ij}(t)$ for all $j \in N - \{i\}$ and $t \in PRED_i$. By definition, π_i must be in $PRED_i$ and $\pi_i \leq t$, $\forall t \in PRED_i$. Suppose $\exists j \ : \ \pi_j > t + \lambda_{ij}(t)$ for some i and $t \in PRED_i$. There are two possible cases that must be considered with respect to the travel times $\lambda_{ij}(t)$. Suppose that $\lambda_{ij}(t) \geq 0$. Since $t \geq \pi_i$ and $t \leq T$, we obtain $\pi_i \leq T$. The algorithm is initiated by defining $\pi_i = \infty$ for $i \neq s$. Therefore, if $i \neq s$, then π_i has been updated and node i was placed in the SE list. However, if $i = s$, then i was also in the SE list. The assumption $\pi_j > t + \lambda_{ij}(t)$ for some $t \in PRED_i$ implies that node i was not completely scanned and must still be in the SE list. This contradicts the assumption of termination. Now, consider the case when $\lambda_{ij}(t) < 0$, i.e. (i,j) is a backward arc in G_x. By construction of a backward arc, (i,j) can only have positive capacity for $t \leq T$. Therefore $\pi_i \leq T$ and node i was placed in the SE list. By the same reason as in the case of nonnegative travel times, the assumption $\pi_j > t + \lambda_{ij}(t)$ for some $t \in PRED_i$ contradicts the assumption of termination.

Concerning the computational complexity, there are at most $n(T+1)-1$ passes and any current node i scans all arcs $(i,j) \in A_x$ in $\mathcal{O}(m)$ time. Furthermore, each time an arc (i,j) is considered, at most $T+1$ computations are required in

order to determine the departure time from node i that will lead to the earliest arrival time at node j. Therefore, the overal complexity is $\mathcal{O}(nmT^2)$. ■

Remark 4.1 *By applying the classical (static) label correcting algorithm on the time-expanded network in which the travel times are considered as costs, the earliest arrival augmenting path is obtained in $\mathcal{O}(n(n+m)T^2)$. Therefore Algorithm 4.1 is faster with respect to the additional factor in the worst case computational complexity.*

The successive earliest arrival augmenting path algorithm repeats the process of finding an $s-d$ earliest arrival augmenting path until the dynamic flow is maximum. The detail description of the algorithm is given in Algorithm 4.2.

Algorithm 4.2 : Solving $(s,d)/(\lambda(t),u(t),a(t),w(t))/V_{\sum^T}$

INPUT	Network $G=(N,A,T)$, time-dependent travel time $\lambda(t)$, capacity $u(t)$, holdover capacity $a(t)$, and maximum allowable waiting time $w(t)$.
OUTPUT	Earliest arrival flow $x_{ij}(t)$.
0	Set the dynamic flow $x_{ij}(t)=0,\ \forall(i,j)\in A;\ t=0,\ldots,T$.
1	Call Algorithm 4.1 to find an $s-d$ earliest arrival augmenting path P in G_x and its $\epsilon(P)$. If $P\neq\emptyset$, then go to step 2. Otherwise, go to step 3.
2	Find the maximum dynamic augmentation of $\mathbf{x}$ along P and update the current flow and G_x. Repeat the process by going back to step 1.
3	Stop the process and $\mathbf{x}$ is an earliest arrival flow.

To prove that Algorithm 4.2 produces a maximum dynamic flow, we need the notion of dynamic cut.

Definition 4.3 (Dynamic Cut)

- *Consider two set-valued functions*

$$C_T:\{0,\ldots,T\} \rightarrow 2^N \tag{19}$$

and

$$\overline{C}_T(t) = N - C_T(t) \tag{20}$$

that satisfy $s\in C_T(t)$ and $d\in\overline{C_T}(t)$ for all $t\leq T$.
We denote by $\mathbf{C_T}$ the collection of all C_T.

The $s-d$ dynamic cut is a set of arcs $(C_T, \overline{C}_T)$ defined by

$$\begin{aligned}(C_T, \overline{C}_T) \quad := \quad & \{\ (i(t), j(t+\lambda_{ij}(t))) : i \in C_T(t), j \in \overline{C}_T(t+\lambda_{ij}(t)), \\ & \quad t+\lambda_{ij}(t) \le T,\ (i,j) \in A\} \cup \\ & \{(i(t), i(t+1))\ :\ i \in C_T(t) \cap \overline{C}_T(t+1),\ w_i(t) > 0, \\ & \quad i \in N - \{s,d\},\ t < T\} \end{aligned} \tag{21}$$

- *The set of times when the movement arc $(i,j) \in A$ crosses the dynamic cut is determined by*

$$\Gamma_{ij}^T := \{t\ :\ i \in C_T(t),\ j \in \overline{C_T}(t+\lambda_{ij}(t)),\ t+\lambda_{ij}(t) \le T\} \tag{22}$$

and the set of times when the holdover arc $(i,i),\ i \in N - \{s,d\}$ crosses the dynamic cut is determined by

$$\Gamma_{ii}^T := \{t\ :\ i \in C_T(t) \cap \overline{C_T}(t+1),\ w_i(t) > 0\},\ i \in N - \{s,d\} \tag{23}$$

- *The value (also called capacity) $W_{\sum^T}(C_T)$ of a dynamic cut $(C_T, \overline{C}_T)$ is defined as*

$$W_{\sum^T}(C_T) := \sum_{(i,j)\in A} \sum_{t \in \Gamma_{ij}^T} u_{ij}(t) + \sum_{i \in N-\{s,d\}} \sum_{t \in \Gamma_{ii}^T} a_i(t) \tag{24}$$

- *The minimum $s-d$ dynamic cut $(C_T, \overline{C}_T)$ is an $s-d$ dynamic cut with*

$$W_{\sum^T}(C_T) \le W_{\sum^T}(C_T^{'}),\ \forall C_T^{'} \in \mathbf{C_T}$$

The value of a dynamic cut as defined by (24) is determined by the capacities of the arcs crossing the cut as well as the storage capacities at points in time where some nodes in N passes from the source side to the sink side of the cut.

Theorem 4.1 *The value $V_{\sum^T}(\mathbf{x})$ of any dynamic flow $\mathbf{x}$ is bounded above by the capacity of any dynamic cut $(C_T, \overline{C}_T)$.*

Theorem 4.2 (Maximum dynamic flow - minimum dynamic cut) *The value of the maximum dynamic flow from a source node s to the sink node d equals the value of the minimum $s-d$ dynamic cut. Moreover, a dynamic flow $\mathbf{x}^*$ and a dynamic cut $(C_T^*, \overline{C_T^*})$ are jointly optimal and having equal valu if and only if*

$$\begin{aligned} x_{ij}^*(t) &= 0 && ,\ i \in \overline{C_T^*}(t) \text{ and } j \in C_T^*(t+\lambda_{ij}(t)) \\ x_{ij}^*(t) &= u_{ij}(t) && ,\ i \in C_T^*(t) \text{ and } j \in \overline{C_T^*}(t+\lambda_{ij}(t)) \\ x_{ii}^*(t) &= w_i(t)a_i(t) && ,\ i \in C_T^*(t) \text{ and } i \in \overline{C_T^*}(t+1) \\ x_{ii}^*(t) &= 0 && ,\ i \in \overline{C_T^*}(t) \text{ and } i \in C_T^*(t+1) \end{aligned}$$

The termination of Algorithm 4.2 occurs when the sink node d in the residual network is not s-reachable, i.e. $\pi_d > T$ or $pred_d(t) = \infty,\ \forall t \leq T$. Using the value of label *pred*, we can give a specific definition to the function C_T in (19) as follows.

$$C_T(t) \ := \ \{i : pred_i(t) \neq \infty,\ i \in N\} \tag{25}$$

and

$$\overline{C_T}(t) \ := \ \{i : pred_i(t) = \infty,\ i \in N\} \tag{26}$$

By this definition, when Algorithm 4.2 terminates, node s and d is in $C_T(t)$ and $\overline{C_T}(t)$, respectively, for any time $t \leq T$. Furthermore, the set of times $\Gamma_{ij}^T,\ \forall (i,j) \in A$ and $\Gamma_{ii}^T, \forall i \in N - \{s,d\}$ is given by

$$\Gamma_{ij}^T \ = \ \{\ t\ :\ pred_i(t) \neq \infty,\ pred_j(t + \lambda_{ij}(t)) = \infty,\ t + \lambda_{ij}(t) \leq T\} \tag{27}$$

and

$$\Gamma_{ii}^T \ = \ \{\ t\ :\ w_i(t) > 0,\ pred_i(t) \neq \infty,\ pred_i(t+1) = \infty,\ t+1 \leq T\}, \tag{28}$$

respectively.

Proposition 4.3 *When Algorithm 4.2 terminates, the set of time-expanded arcs* $(C_T, \overline{C_T})$ *as defined by* (25)-(26) *with respect to the label at the termination stage, defines a minimum* $s - d$ *dynamic cut for the time horizon* T *and the current dynamic flow* $\mathbf{x}$ *is a maximum dynamic flow.*

Proof:
Clearly, $s \in C_T(t)$ and $d \in \overline{C_T}(t)$ for any $t \leq T$. Since the algorithm cannot label any node $j \in \overline{C_T}(t + \lambda_{ij}(t))$ from any node $i \in C_T(t)$ with $t + \lambda_{ij}(t) \leq T$ and $(i,j) \in A$, the residual movement capacity $u_{ij}^x(t) = 0$ for each $(i,j) \in (C_T(t), \overline{C_T}(t + \lambda_{ij}(t)))$. Furthermore, since $u_{ij}^x(t) = u_{ij}(t) - x_{ij}(t) + x_{ji}(t + \lambda_{ij}(t)), x_{ij}(t) \leq u_{ij}(t)$, and $x_{ji}(t + \lambda_{ij}(t)) \geq 0$, the condition $u_{ij}^x(t) = 0$ implies that $x_{ij}(t) = u_{ij}(t)$ for every arc $(i,j) \in (C_T(t), \overline{C_T}(t + \lambda_{ij}(t)))$ and $x_{ji}(t + \lambda_{ij}(t)) = 0$ for every arc $(j,i) \in (\overline{C_T}(t + \lambda_{ij}(t)), C_T(t))$. We must also show that $x_{ij}(t) = 0$ for every arc $(i,j) \in (\overline{C_T}(t), C(t + \lambda_{ij}(t)))$. Suppose that this is not true, i.e. $x_{ij}(t) > 0$ for every arc $(i,j) \in (\overline{C_T}(t), C(t + \lambda_{ij}(t)))$. By (8), $u_{ji}^x(t + \lambda_{ij}(t)) > 0$. Consequently, we can label i at time t from j at time $t + \lambda_{ij}(t)$, contradicting the assumption that $i \in \overline{C_T}(t)$. Therefore, we can conclude that $x_{ij}(t) = 0$ for every arc $(i,j) \in (\overline{C_T}(t), C(t + \lambda_{ij}(t)))$.

Similar ideas are used to prove that the holdover flows $x_{ii}(t) = a_i(t)$ for every holdover arc (i,i) crossing from $C_T(t)$ to $\overline{C_T}(t+1)$ and $x_{ii}(t) = 0$ for every holdover arc (i,i) crossing from $\overline{C_T}(t)$ to $C_T(t+1)$. Since the algorithm can not label any node i in $\overline{C_T}(t+1)$ from node i-itself at time t with $t+1 \leq T$, it must be that the residual positive waiting capacity $a_i^{x+}(t) = 0$. Let us denote by $x_{ii}^-(t+1)$ the number of waiting canceling units at node i at time t. Since

$a_i^{x+}(t) = a_i(t) - x_{ii}(t) + x_{ii}^-(t+1)$ with $x_{ii}(t) \leq a_i(t)$, and $x_{ii}^-(t+1) \geq 0$, the condition $a_i^{x+}(t) = 0$ implies that $x_{ii}(t) = a_i(t)$ and $x_{ii}^-(t+1) = 0$ for every node $i \in C_T(t) \cap \overline{C_T}(t+1)$. To show that $x_{ii}(t) = 0$ for every holdover arc $(i,i) \in (\overline{C_T}(t), C_T(t+1))$, we must show that $a_i^{x-}(t+1) = 0$ for such an arc. If $a_i^{x-}(t+1) > 0$, then node i at time t can be labeled from i-itself at time $(t+1)$, contradicting the assumption that $i \in \overline{C_T}(t)$. Therefore, it must be $a_i^{x-}(t+1) = 0$, implying $x_{ii}(t) = 0$ for every holdover arc (i,i) crossing from $\overline{C_T}(t)$ to $C_T(t+1)$.

Furthermore, Theorem 4.2 implies that $\mathbf{x}$ is a maximum dynamic flow and the corresponding dynamic cut $(C_T, \overline{C_T})$ is a minimum dynamic cut. ■

Finally, the correctness that Algorithm 4.2 produces an earliest arrival flow is stated by the following proposition.

Proposition 4.4 *Let us denote by U the biggest capacity over all $(i,j) \in A$ and over time $t \in \{0, \ldots, T\}$, i.e.*

$$U := \max_{(i,j)\in A} \max_{t\in\{0,\ldots,T\}} \{u_{ij}(t)\} \tag{29}$$

Algorithm 4.2 solves TdEAFP with the worst case complexity $\mathcal{O}(nm^2T^3U)$.

Proof :
By Proposition 4.3 and the fact that the augmentation always done in a path with the earliest arrival time at the sink, Algorithm 4.2 produces a maximum dynamic flow with earliest arrival property for the time horizon T. Furthermore, since the capacity of the cut is at most mUT and each augmentation carries at least one unit of flow, there is at most mUT augmentations. By Proposition 4.2, the overall complexity is $\mathcal{O}(nm^2T^3U)$. ■

5 Infinite waiting

Here we allow infinite waiting at every node $i \in N - \{s,d\}$, i.e.

$$w_i(t) = 1 \quad \text{and} \quad a_i(t) := \infty, \ i \in N - \{s,d\}, \ t \in \{0, \ldots, T\}$$

Proposition 5.1 *Suppose that the waiting times and capacities are infinite. If there is an $s-j$ augmenting path with arrival time $\pi_j < T$ (i.e. $pred_j(\pi_j) \neq \infty$), then there must exist an $s-j$ augmenting path for any arrival time $t > \pi_j$, i.e. $pred_j(t) \neq \infty$, $\forall t \in \{\pi_i + 1, \pi_i + 2, \ldots, T\}$.*

Proof :
Let P_{sj} be an $s-j$ augmenting path with arrival time π_j. Since the waiting at any node in $N - \{s,d\}$ and at any time in $\{0, \ldots, T\}$ is infinite, we can extend this path by considering the waiting at node j for $t - \pi_j$ time units to obtain an $s-j$ augmenting path arriving at node j at time $t > \pi_j$. ■

Proposition 5.1 has the following direct consequence.

Corollary 5.1 *Consider the case when infinite waiting in any node $i \in N - \{s, d\}$ is allowed. Assume that during an iteration in Algorithm 4.1, node i is selected as the current node. If node j at time t' is reachable from node i at time t, i.e. $u_{ij}^x(t) > 0,\ t + \lambda_{ij}(t) = t' \leq T$, but it is not previously reachable from any other node, i.e. $pred_j(t') = \infty$, then the previous value of π_j must be strictly greater than t' and the current value is greater than or equal to t'.*

Using this corollary, step 2 of Algorithm 4.1 can be simplified as given in Table 1.

Scan current node and update the labels
For all $(i, j) \in A_x$ do
{ For all $t \in \{t : \pi_i \leq t \leq T,\ u_{ij}^x(t) > 0,\ t + \lambda_{ij}(t) \leq T\}$
do {
If $(pred_j(t + \lambda_{ij}(t)) = \infty)$ then
{
$\pi_j = t + \lambda_{ij}(t)$
$pred_j(t + \lambda_{ij}(t)) := i \quad ; \quad dep_j(t + \lambda_{ij}(t)) := t$
$SE := SE + \{j\}$
Define $t' := t + \lambda_{ij}(t) + 1$
While $(t' \leq T)$ and $(pred_j(t') = \infty))$
do $\{pred_j(t') = j \quad ; \quad dep_j(t') := t' - 1\ ;\ t'++\}$
Define $t' := t + \lambda_{ij}(t) - 1$
While $(t' \geq 0,\ w_j^{x-}(t' + 1) < 0,\ a_j^{x-}(t' + 1) > 0,$
and $pred_j(t') = \infty)$
do { if $(\pi_j > t')$ then $\pi_j := t'$
$pred_j(t') = j \quad ; \quad dep_j(t') := t' + 1\ ;\ t'--\}$
}
}
} Return to step 1

Table 1: Modified step 2 of Algorithm 4.1 when infinite waiting is allowed

Another important consequence of Proposition 5.1 is stated by the following proposition.

Proposition 5.2 *If the waiting is infinite, by applying the scaning and updating processes given in Table 1 to step 2 of Algorithm 4.1, an earliest arrival augmenting path can be found in $\mathcal{O}(nmT)$.*

Proof :
By Proposition 4.2, the modified algorithm finds an earliest arrival augmenting path. Since the waiting is infinite, by Proposition 5.1, there are at most $n - 1$ passes (instead of $n(T + 1) - 1$ passes in the case of finite waiting). Conse-

quently, an earliest arrival augmenting path can be found in $\mathcal{O}(nmT)$. ■

Corollary 5.2 *Algorithm 4.2 solves TdEAFP in $\mathcal{O}(nm^2T^2U)$ when infinite waiting is allowed for every node $i \in N - \{s, d\}$.*

By considering Remark 4.1, we obtain the following corollary.

Corollary 5.3 *Under the assumption of infinite waiting, Algorithm 4.2 is more efficient by factor T than implementing the successive static shortest augmenting path algorithm on the time-expanded network.*

This assumption of infinite waiting also influences the characteristic of the dynamic cut as stated by the following lemma.

Proposition 5.3 *If infinite waiting is allowed for every node $i \in N - \{s, d\}$, once a node i is in the source side of the cut at time t, it will stay there forever, i.e. if $i \in C_T(t)$, then $i \in C_T(t')$ for any time $t' > t$.*

Proof :
Follow directly from the definition of C_T given by (25) and Proposition 5.1. ■

6 Illustrative Example

The following Example 6.1 illustrates the implementation of Algorithms 4.1 and 4.2.

Example 6.1
Figure 5 shows a network structure with 6 nodes and 8 arcs where node 0 is the source and node 5 is the sink. The time-dependent travel times and arc capacities are given in Table 2. The time-dependent maximum waiting times and capacities are given in Table 3. We define T equals 7 time units.

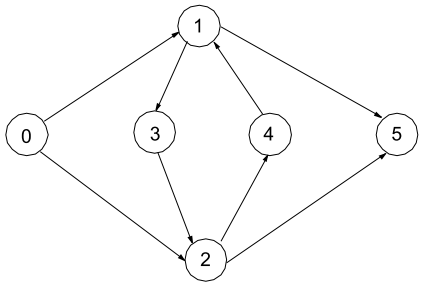


Figure 5: A network for Example 6.1

Initially, we define the flow variables $x_{ij}(t) = 0,\ \forall (i,j) \in A;\ \forall t \leq T$. Therefore, we have $G_x = G$. The results of Step 1 of Algorithm 4.2 are given in Table 4. Since $\pi_5 = 4 < T = 7$, the augmenting path P_1 and the maximum flow augmentation $\epsilon(P_1)$ can be obtained by a backtracking procedure as described in

(i,j)	$(0,1)$	$(0,2)$	$(1,3)$	$(1,5)$	$(2,4)$	$(2,5)$
$u_{ij}(t)$	$6,\ t \le 1$ $2,\ t \ge 2$	$2,\ t \le 1$ $5,\ t \ge 2$	$5,\ t \ge 0$	$3,\ t \le 3$ $1,\ t \ge 4$	$5,\ t \ge 0$	$6,\ t \ge 0$
$\lambda_{ij}(t)$	$4,\ t \le 1$ $5,\ t \ge 2$	$2,\ t \le 1$ $4,\ t \ge 2$	$1,\ t \ge 0$	$3,\ t = 4$ $1,\ t \ne 4$	$1,\ t \le 4$ $2,\ t \ge 5$	$5,\ t \le 1$ $3,\ t \ge 2$

(i,j)	$(3,2)$	$(4,1)$
$u_{ij}(t)$	$5,\ t \ge 0$	$5,\ t \ge 0$
$\lambda_{ij}(t)$	$0,\ t \ge 0$	$0,\ t \ge 0$

Table 2: Time-dependent travel times and capacities for the network in Example 6.1

$i \in N - \{0,5\}$	1	2	3	4
$w_i(t)$	$1,\ t \le 4$ $0,\ t \ge 5$	$1,\ t \ge 0$	$0,\ t \ge 0$	$0,\ t \ge 0$
$a_i(t)$	$4,\ t \le 4$ $0,\ t \ge 5$	$5,\ t \ge 0$	$0,\ t \ge 0$	$0,\ t \ge 0$

Table 3: Time-dependent maximum waiting times and capacities for the network in Example 6.1

Labels		time t							
		0	1	2	3	4	5	6	7
$\pi_0 = 0$	$pred_0(t)$	-1	-1	-1	-1	-1	-1	-1	-1
	$dep_0(t)$	0	1	2	3	4	5	6	7
$\pi_1 = 3$	$pred_1(t)$	∞	∞	∞	4	0	1	∞	0
	$dep_1(t)$	∞	∞	∞	3	0	4	∞	2
$\pi_2 = 2$	$pred_2(t)$	∞	∞	0	2	2	2	2	2
	$dep_2(t)$	∞	∞	0	2	3	4	5	6
$\pi_3 = 4$	$pred_3(t)$	∞	∞	∞	∞	1	1	1	∞
	$dep_3(t)$	∞	∞	∞	∞	3	4	5	∞
$\pi_4 = 3$	$pred_4(t)$	∞	∞	∞	2	2	2	∞	2
	$dep_4(t)$	∞	∞	∞	2	3	4	∞	5
$\pi_5 = 4$	$pred_5(t)$	∞	∞	∞	∞	1	2	2	2
	$dep_5(t)$	∞	∞	∞	∞	3	2	3	4

Table 4: Labels on nodes after completing Algorithm 4.1

step 3 of Algorithm 4.1. We obtain $P_1 = \{0(0,0), 2(2,2), 4(3,3), 1(3,3), 5(4,4)\}$ with $\epsilon(P_1) = 2$.

The time-dependent travel times and capacities of the new residual network are given in Table 5. Four backward arcs $(2,0)$, $(4,2)$, $(1,4)$, and $(5,1)$ are added to the residual network. The dynamic flow $\mathbf{x}$ is updated by using

(i,j)	$(0,1)$	$(0,2)$	$(1,3)$	$(1,5)$	$(2,4)$	$(2,5)$
$u^x_{ij}(t)$	$6,\ t \le 1$ $2,\ t \ge 2$	$0,\ t = 0$ $2,\ t = 1$ $5,\ t \ge 2$	$5,\ t \ge 0$	$1,\ t \ge 3$ $3,\ t \le 2$	$3,\ t = 2$ $5,\ t \ne 2$	$6,\ t \ge 0$
$\lambda^x_{ij}(t)$	$4,\ t \le 1$ $5,\ t \ge 2$	$2,\ t \le 1$ $4,\ t \ge 2$	$1,\ t \ge 0$	$3,\ t = 4$ $1,\ t \ne 4$	$1,\ t \le 4$ $2,\ t \ge 5$	$5,\ t \le 1$ $3,\ t \ge 2$

(i,j)	$(3,2)$	$(4,1)$	$(2,0)$	$(4,2)$	$(1,4)$	$(5,1)$
$u^x_{ij}(t)$	$5,\ t \ge 0$	$3,\ t = 3$ $5,\ t \ne 3$	$2,\ t = 2$ $0,\ t \ne 2$	$2,\ t = 3$ $0,\ t \ne 3$	$2,\ t = 3$ $0,\ t \ne 3$	$2,\ t = 4$ $0,\ t \ne 4$
$\lambda^x_{ij}(t)$	$0,\ t \ge 0$	$0,\ t \ge 0$	$-2,\ t = 2$ $0,\ t \ne 2$	$-1,\ t = 3$	$0,\ t \ge 0$ $0,\ t \ne 3$	$-1,\ t = 4$ $0,\ t \ne 4$

Table 5: Time-dependent travel times and capacities for the residual network after the first pass of Algorithm 4.1

Labels		time t							
		0	1	2	3	4	5	6	7
$\pi_0 = 0$	$pred_0(t)$	-1	-1	-1	-1	-1	-1	-1	-1
	$dep_0(t)$	0	1	2	3	4	5	6	7
$\pi_1 = 4$	$pred_1(t)$	∞	∞	∞	∞	0	1	∞	0
	$dep_1(t)$	∞	∞	∞	∞	0	4	∞	2
$\pi_2 = 5$	$pred_2(t)$	∞	∞	∞	∞	∞	3	0	2
	$dep_2(t)$	∞	∞	∞	∞	∞	5	2	6
$\pi_3 = 5$	$pred_3(t)$	∞	∞	∞	∞	∞	1	1	∞
	$dep_3(t)$	∞	∞	∞	∞	∞	4	5	∞
$\pi_4 = 7$	$pred_4(t)$	∞	∞	∞	∞	∞	∞	∞	2
	$dep_4(t)$	∞	∞	∞	∞	∞	∞	∞	5
$\pi_5 = \infty$	$pred_5(t)$	∞	∞	∞	∞	∞	∞	∞	∞
	$dep_5(t)$	∞	∞	∞	∞	∞	∞	∞	∞

Table 6: Labels on nodes after completing Algorithm 4.1 for the fifth times

(17). The process is then continued to look for another earliest augmenting paths. Three additional earliest augmenting paths have been found before the stopping criterion of Algorithm 4.2 is fulfilled (i.e. $\pi_5 > T$), namely: $P_2 = \{0(1,1), 2(3,3), 5(6,6)\}$ with $\epsilon(P_2) = 2$, $P_3 = \{0(0,0), 1(4,5), 5(6,6)\}$ with $\epsilon(P_3) = 1$, and $P_4 = \{0(0,0), 1(4,4), 5(7,7)\}$ with $\epsilon(P_4) = 1$.

The final labels (i.e. after the fifth pass of Algorithm 4.1) which are not able to generate any augmenting path, are shown in Table 6. The earliest arrival flow $\mathbf{x}$ having value $V_{\sum^{T=7}}(\mathbf{x}) = 6$ is given in Table 7.

From the final labels given in Table 6, we can construct the sets Γ^T_{ij} for every arc $(i,j) \in A$ and Γ^T_{ii} for every node $i \in N - \{0,5\}$. The results are shown in Table 8. Figure 6 (a) depicts the minimum dynamic cut $(C_7, \overline{C_7})$ and the

$x_{ij}(t)$	time t							
	0	1	2	3	4	5	6	7
$x_{01}(t)$	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
$x_{02}(t)$	2	2	0	0	0	0	0	0
	2	2	0	0	0	0	0	0
$x_{13}(t)$	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
$x_{15}(t)$	0	0	0	2	1	1	0	0
	0	0	0	2	1	1	1	0
$x_{24}(t)$	0	0	2	0	0	0	0	0
	0	0	2	0	0	0	0	0
$x_{25}(t)$	0	0	0	2	0	0	0	0
	0	0	0	2	0	0	0	0
$x_{32}(t)$	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
$x_{41}(t)$	0	0	0	2	0	0	0	0
	0	0	0	2	0	0	0	0

Table 7: Earliest arrival flow of Example 6.1. The upper position of each row $x_{ij}(t)$ contains the optimal flow when the waiting at a node is limited, while the lower one deals with the case when infinite waiting is allowed at any node

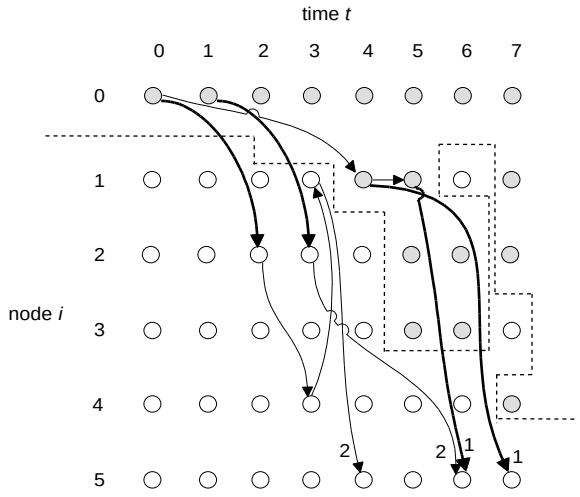
Γ^7_{01}	$=$	$\emptyset$
Γ^7_{02}	$=$	$\{0,1\}$
Γ^7_{13}	$=$	$\emptyset$
Γ^7_{15}	$=$	$\{4,5\}$
Γ^7_{24}	$=$	$\emptyset$
Γ^7_{25}	$=$	$\emptyset$
Γ^7_{32}	$=$	$\emptyset$
Γ^7_{41}	$=$	$\emptyset$
Γ^7_{11}	$=$	$\{5\}$
Γ^7_{22}	$=$	$\emptyset$
Γ^7_{33}	$=$	$\{6\}$
Γ^7_{44}	$=$	$\emptyset$

Table 8: The sets Γ^T_{ij} for every arc $(i,j) \in A$ and Γ^T_{ii} for every node $i \in N - \{0,5\}$ correspond to the earliest arrival flow of Example 6.1

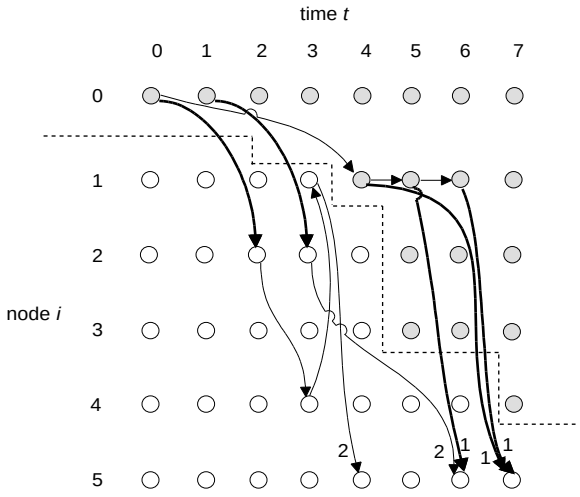
corresponding optimal dynamic flow.

The value of $(C_7, \overline{C_7})$ is determined by

$$\begin{aligned} W_{\sum^7}(C_7) &= \sum_{(i,j)\in A} \sum_{t\in\Gamma^7_{ij}} u_{ij}(t) + \sum_{i\in N-\{s,d\}} \sum_{t\in\Gamma^7_{ii}} a_i(t) \\ &= 6 \end{aligned}$$



(a)



(b)

Figure 6: The minimum dynamic cut of Example 6.1. The upper side of the dashed line is the source side of the cut, while the lower side is the sink side of the cut. The shaded circles are in the source side of the cut. The bold curves are the curves of the optimal flows of the arcs in the cut. Numbers in the curves define the values of the flows. (a) The minimum dynamic cut when only limited waiting is allowed and (b) when infinite waiting at any node but the sink node, is allowed

which is equal to the value of the earliest arrival flow for $T = 7$.

When infinite waiting at any node is allowed, including the waiting at the source node 0, the value of earliest arrival flow changes to $V_{\sum^{T=7}} = 7$. The optimal flow distribution of this case is given in the lower position of each row $x_{ij}(t)$ in Table 7. Figure 6 (b) depicts the optimal flow distribution and dynamic cut when infinite waiting at any node is allowed. $\square$

7 Computational Results

Experiments are run to know the computation performance of the proposed Algorithm 4.2. Toward this goal, a series of experiments is run on the basis of randomly generated dynamic networks. To do the experiments, Algorithm 4.2 is implemented in C++ and run on a PC Pentium III, 500 MHz, and RAM of 256 MB.

To generate the random sample of dynamic networks, a dynamic random network generator is developed by extending the idea of NETGEN, proposed by Klingman, Napier, and Stutz [11], to include the time-dependent data. The generated networks are connected. The experiments are conducted on random networks with 50, 100, 500, and 1000 nodes and time horizon $T = 100$. For each choice of n nodes, we create networks with indegree and outdegree of each node 2, 4, 6, and 8. It is assumed that the source node and sink node has zero indegree and zero outdegree, respectively. This degree setting implies the generated networks have $2n$, $4n$, $6n$, and $8n$ arcs. The minimum and maximum travel time is defined as 1 and 10, respectively, and the minimum and maximum capacitiy is defined as 25 and 50, respectively. The maximum waiting capacity is defined as 10. For each specific setting of n and m, we test five random dynamic networks. Therefore, the total number of observations is 80.

The results of the experiments are given in Table 9. To find out the inter-

n	m/n	Average CPU-time (in seconds)	Average no. of EAAP
50	2	24.31	1,310.40
	4	49.19	1,729.60
	6	74.66	2,232.60
	8	114.17	2,855.80
100	2	55.41	1,331.20
	4	101.02	1,649.20
	6	269.45	2,412.40
	8	381.74	3,203.00
500	2	367.22	1,474.20
	4	1,790.08	2,972.60
	6	2,342.31	3,312.80
	8	2,717.88	3,954.20
1000	2	2,214.14	2,679.00
	4	5,412.12	3,755.00
	6	6,723.90	4,095.40
	8	9,733.41	4,393.00

Table 9: Computational test results of Algorithm 4.2

relation among the CPU-time, number of nodes n, and network density m/n, we do regression of the log (base 10) of the CPU-time against the log of n and

log of m/n. We obtain the following equation

$$\widehat{CPU} = 0.0267 n^{1.4661} (m/n)^{1.2588}$$

where $\widehat{CPU}$ denotes the estimate CPU-time in seconds. The adjusted R^2 is 0.9631 and the standard error is 0.5343. The value of adjusted R^2 which close to 1 and a fairly small standard error indicate that the fit is indeed very good.

References

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows : Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] J. E. Aronson. A survey of dynamic network flows. *Annals of Operation Research*, 20:1–66, 1989.

[3] L. Fleischer. Universally maximum flows with piecewise-constant capacities. *Networks*, 38(3):115–125, 2001.

[4] L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In W. Cook and A. Schulz, editors, *Integer Programming and Combinatorial Optimization,* volume 2337 of *Lecture Notes in Computer Science*, pages 36–53, Springer, Berlin, 2002.

[5] L. Ford and D. Fulkerson. Constructing maximal dynamic flows from static flows. *Operation Research*, 6:419–433, 1958.

[6] L. Ford and D. Fulkerson. *Flows in Network.* Princeton University Press, Princeton, New Jersey, 1962.

[7] D. Gale. Transient flows in networks. *The Michigan Mathematical Journal*, 6:59–63, 1959.

[8] G. Gallo, M. Grigoriadis, and R. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. COMPUT*, 18(1):30–55, 1989.

[9] H. Hamacher and S. Tjandra. Mathematical modeling of evacuation problems: a state of the art. In M. Schreckenberg and S. Sharma, editors, *Pedestrian and Evacuation Dynamics*, pages 227–266, Springer-Verlag, Berlin, 2002.

[10] B. Hoppe and E. Tardos. Polinomial time algorithms for some evacuation problems. *Proc. of 5th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 433–441, 1994.

[11] D. Klingman, A. Napier, and J. Stutz. NETGEN : A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20, 1974.

[12] E. Minieka. Maximal, Lexicographic, and dynamic network flows. *Operations Research*, 21:517–527, 1973.

[13] R. Ogier. Minimum delay routing in continuous-time dynamic networks with piecewise constant capacities. *Networks*, 18:303–318, 1988.

[14] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the A.C.M.*, 37(3):607–625, 1990.

[15] W. Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operation Research*, 19:1602–1612, 1971.