Stevanus Adrianto Tjandra

Dynamic Network Optimization with Application to the Evacuation Problem

Vom Fachbereich Mathematik
der Universität Kaiserslautern
genehmigte

Dissertation

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)

Gutachter: Prof. Dr. Horst W. Hamacher
Prof. Dr. Margaret M. Wiecek

Datum der Disputation: 27. Mai 2003

UNIVERSITÄT KAISERSLAUTERN

Fachbereich Mathematik

For my wife

Fransiska Yulianty

Acknowledgement

I would like to take the opportunity to praise Jesus Christ for His blessing and mercy.

Also, I would like to thank all people who have supported my work in various ways and enabled me to write this thesis. Most gratefully I would like to thank Prof. Dr. Horst W. Hamacher. He gave me a chance to come to Germany and work at the *Fraunhofer Institut Techno- und Wirtschaftsmathematik* (ITWM) in Kaiserslautern, accepted me as his Ph. D student, and finally gave me a chance to work in his working group for a research project from *Deutsche Forschungsgemeinschaft* (DFG). His guidance and help through the years has been most essential reason that enabled me to do this work. Moreover I have to thank Prof. Dr. Margaret M. Wiecek for her effort to read and evaluate my thesis.

I am particularly indebted to Prof. Dr. Helmut Neunzert and Dr. Falk Triebsch who have given their effort to help me and my wife in many cases during our study at the University of Kaiserslautern.

Many thanks go to all my colleagues in the optimization group of ITWM and in the working group of Prof. Hamacher, for many interesting discussions and the very good atmosphere. I am also grateful to all my Indonesian friends in Kaiserslautern for making life nice and worth living.

Finally, in her gentle but firm criticism of many my ideas, and in her unfailing moral support and love, my lovely wife Fransiska Yulianty has contributed in many ways toward bringing the success of my study. For this, and for much more besides, I am very thankful.

Contents

Acknowledgement i

Table of Contents iii

List of Symbols vii

1 Introduction 1

2 Dynamic Network Flow 7
- 2.1 Discrete-Time Dynamic Network Flow . . . 7
- 2.2 Time-Expanded Network . . . 12
- 2.3 First-In First-Out . . . 17
- 2.4 Residual Dynamic Network . . . 18

3 Maximum Dynamic Network Flows 21
- 3.1 Problem Formulation . . . 22
- 3.2 Dynamic Cut . . . 23
- 3.3 Solution Algorithm for a Maximum Dynamic Network Flow Problem with Constant Attributes . . . 30
- 3.4 Solution Algorithm for a Maximum Dynamic Network Flow Problem with Time-Dependent Attributes . . . 41
- 3.5 Computational Results . . . 52

4 Earliest Arrival Flows 57
- 4.1 Problem Formulation . . . 58
- 4.2 Solution Algorithm for an Earliest Arrival Flow Problem with Constant Attributes . . . 60
- 4.3 Solution Algorithm for an Earliest Arrival Flow Problem with Time-Dependent Attributes . . . 63
- 4.4 Infinite Waiting . . . 69
- 4.5 Computational Results . . . 74

5 Quickest Flow Problems **79**
5.1 Problem Formulation 80
5.2 Quickest Flow Problem with Constant Attributes 80
5.3 Quickest Flow Problem with Time-Dependent Attributes 83
5.4 Quickest Flow Problem with Time-Dependent Supplies 87

6 Time-Dependent Bicriteria Dynamic Shortest Path Problems **89**
6.1 Problem Formulation 91
6.2 Problem Characteristics 95
6.2.1 Complexity 95
6.2.2 Principle of Optimality 96
6.2.3 Negative Dynamic Cycle 99
6.3 Label Structure 100
6.4 Algorithms for All Ready Times with Nonnegative Attributes 101
6.4.1 Backward Dynamic Programming 102
6.4.2 Backward Label Setting Algorithm 103
6.5 Algorithm for All Ready Times with Arbitrary Attributes 108
6.6 Computational Results 118

7 Application to Evacuation Problems **129**
7.1 Introduction 129
7.2 Network Model of Evacuation Objects 131
7.3 The Need of Time-Dependent Modeling 134
7.4 Solution Methods 137
7.4.1 Optimal Evacuees Distribution 137
7.4.2 Multicriteria Evacuation Routes 138
7.5 Case Study 138

8 Conclusions and Future Research **143**

Bibliography **147**

A Building 42 and Its Network Representations **153**

B Dynamic Network Generator **159**

C Representative Operation Counts **163**

D Classification of Dynamic Network Problems **167**

List of Tables **169**

List of Figures 171

Index 175

The Author's Scientific Career 179

List of Symbols

Networks :

$G_{stat} = (N, A)$ ..... static network
N ..... set of nodes
A ..... set of arcs
n ..... cardinality of N
m ..... cardinality of A
$A^+(i)$ ..... set of outgoing arcs of node i
$A^-(i)$ ..... set of incoming arcs of node i
$\mathbf{f}$ ..... static flow in $G = (N, A)$
f_{ij} ..... static flow on arc (i, j)
G^f_{stat} ..... residual static network with respect to $\mathbf{f}$
u^f_{ij} ..... residual capacity of arc (i, j) with respect to $\mathbf{f}$
λ^f_{ij} ..... residual travel times of arc (i, j) with respect to $\mathbf{f}$
T ..... time horizon
$G = (N, A, T)$ ..... dynamic network
G_T ..... the associated time-expanded network of $G = (N, A, T)$
N_T ..... set of nodes of G_T
A_T ..... set of arcs of G_T
A^H ..... set of holdover arcs of G_T
A^M ..... set of movement arcs of G_T
s ..... source node
d ..... sink node
S ..... set of nodes with positive supplies
s^T ..... super source node at the time-expanded network G_T
d^T ..... super sink node at the time-expanded network G_T
$\lambda_{ij}(t)$ ..... travel time along an arc (i, j) at time t
$u_{ij}(t)$ ..... flow capacity of arc (i, j) at time t
$a_i(t)$ ..... holdover flow or waiting capacity of node i at time t
$c_{ij}(t)$ ..... cost of one unit flow on arc (i, j) at time t
$q_i(t)$ ..... supply/demand of node i at time t
$\mathbf{x}$ ..... dynamic flow in $G = (N, A, T)$
$x_{ij}(t)$ ..... movement flow on arc (i, j) at time t
$x_{ii}(t)$ ..... holdover flow at node i at time t

G_x ..... residual dynamic network with respect to $\mathbf{x}$
$u_{ij}^x(t)$ ..... residual capacity of arc (i,j) at time t with respect to $\mathbf{x}$
$c_{ij}^x(t)$ ..... residual cost of arc (i,j) at time t with respect to $\mathbf{x}$
$\lambda_{ij}^x(t)$ ..... residual travel time of arc (i,j) at time t with respect to $\mathbf{x}$
$a_i^{x+}(t)$ ..... residual waiting capacities at node i at time t with respect to $\mathbf{x}$
$a_i^{x-}(t+1)$ ..... maximum amount of waiting canceling at time t with respect to $\mathbf{x}$
$\lambda(P)$ ..... distance of path P with respect to the travel times
$v(P)$ ..... value of flow along a path P
U ..... the biggest capacity over all arc $(i,j) \in A$ and over time $t \leq T$

Maximum dynamic flow with time-dependent attributes:

$V_{\sum^T}(\mathbf{x})$ ..... value of $\mathbf{x}$ for a time horizon T
$\phi_i(t)$ ..... distance from node i departing at time t to the sink node d
$G_x(\triangle)$ ..... dynamic network containing only arcs and nodes whose residual capacity is at least $\triangle$
$(C_T, \overline{C}_T)$ ..... dynamic cut for a time horizon T
Γ_{ij}^T ..... set of times when the movement arc (i,j) crosses $(C_T, \overline{C}_T)$
Γ_{ii}^T ..... set of times when the holdover arc (i,i) crosses $(C_T, \overline{C}_T)$
$W_{\sum^T}(C_T)$ ..... value (capacity) of a discrete-time dynamic cut $(C_T, \overline{C}_T)$

Earliest arrival flow with time-dependent attributes:

$V_{\sum^{T' \leq T}}$ ..... value of an earliest arrival flow for a time horizon T
π_i ..... the earliest arrival time at node $i \in N$ from the source s
$pred_i(t)$ ..... predecessor of node i along an $s-i$ augmenting path that arrives at i at time t
$dep_i(t)$ ..... departure time of $pred_i(t)$ with respect to an arrival time t at i along an $s-i$ augmenting path

Quickest flow:

$q_s(0)$ ..... initial content of the network
$T_{\sum^{q_s(0)}}$ ..... minimum clearing time of a given initial content $q_s(0)$

Time dependent bicriteria dynamic shortest path:

$=$ ..... $z_1 = z_2$ iff $z_1^k = z_2^k,\ \forall k = 1,2$
$\leq$ ..... $z_1 \leq z_2$ iff $z_1^k \leq z_2^k,\ \forall k = 1,2$
$<$ ..... $z_1 < z_2$ iff $z_1^k \leq z_2^k,\ \forall k = 1,2$ and $\exists k \in \{1,2\} : z_1^k < z_2^k$
$c_{ij}^k(t)$ ..... k-th cost criteria of passing through an arc (i,j) at time t
$h_i(t)$ ..... holding or waiting cost at node i at time t
$w_i(t)$ ..... maximum allowable waiting time at node i starting at time t
$PO(\mathbb{P}_{id}(t))$ ..... minimal complete set of all $i-d$ Pareto optimal paths which is ready at time $t \in \{0, \ldots, T\}$

$\pi_i(t) := \left(\begin{pmatrix} \pi^1 \\ \pi^2 \end{pmatrix}; (t_{ready}, t_{depart}); succ\_ptr \right)$ ..... label of node i at time t

t_{ready} ..... ready time at i along an $i-d$ path associated to $\pi_i(t)$

t_{depart} ..... departure time at i along an $i-d$ path associated to $\pi_i(t)$

$succ\_ptr$ ..... a pointer points to the label of the successor node

$\Pi_i(t)_{prm}$ ..... set of permanent labels of node i at time t

$\Pi_i(t)_{tmp}$ ..... set of temporary labels of node i at time t

Set of numbers:

$\mathbb{R}_0^+$ ..... set of nonnegative real numbers

$\mathbb{Z}_0^+$ ..... set of nonnegative integer numbers

Chapter 1

Introduction

Classical (static) network flow models have been well known as valuable tools for many applications (see e.g., Ahuja, Magnanti, and Orlin [AMO93]). However, they fail to capture the dynamic property of many real-life problems, such as traffic planning, production and distribution systems, communication systems, and evacuation planning. A static flow can not properly consider the evolution of a system over time. The time here is an essential component, either because the flows of some commodity take time to pass from one location to another, or because the structure of the network changes over time. In real-life problems, both phenomena may appear together. Take, for example, a building evacuation problem: people have finite walking speed and from time to time the passageway (e.g., corridor and stairwell) can become unaccessible due to, for instance, smoke, fire, or falling debris. More examples are described in the survey paper of Aronson [Aro89].

To tackle this problem, Ford and Fulkerson [FF58, FF62] introduced flows which take time, called travel time, to pass an arc of the network over a finite time horizon T. This flow is called dynamic flow or flow over time. The time itself can take a continuous or discrete value. *In this thesis we focus on the discrete-time dynamic flow.*

Not surprisingly, dynamic network flow problems are more complex than static ones, since, for example, they require to keep track of when each unit of flow travels through an arc of the network so that no arc capacity is violated at any time. Only in some cases, these problems are polynomially solvable, see e.g., Ford and Fulkerson [FF58, FF62] and Burkard, Dlaska, and Klinz [BDK93]. In many cases dynamic network flow problems are NP-hard or at least there is no polynomial time algorithm known to solve them, see e.g., Hoppe and Tardos [HT94], Klinz and Woeginger [KW95], Fleischer [Fle01], and Fleischer and Skutella [FS02].

All of the existing results on dynamic network flow that we have mentioned above assume that the network attributes such as arc travel times (or costs), arc and node capacities, and the supply at the source nodes, are constant (i.e., time-independent). For many applications mentioned previously, this constant assumption is, however, certainly inade-

quate. To obtain more realistic results, the model must account for the time-dependent (or time-varying) nature of those network attributes. We make the distinction between the words "dynamic" and "time-dependent". Dynamic is associated with the movement of flow over time through the network whereas time-dependent is associated with the network attributes. Hence, we have dynamic network flow problems either with constant attributes or with time-dependent attributes. We call a dynamic network flow problem with time-dependent attributes a time-dependent dynamic network flow problem.

In general there are three approaches to solve dynamic network flow problems. The first approach is to create a so-called time-expanded network which is a static representation of the dynamic network. This network is constructed by making a copy of the original network for each discrete-time period. The dynamic flow problems are then solved by applying existing static network flow algorithms to this time-expanded network. The size of the time-expanded network is in general exponential with respect to the input size of the problem and therefore is typically very large for realistic problems. The second approach is to reduce dynamic network flow problems to static ones and make use of existing algorithms to solve them. The third approach is to explore only the time-dependent property of network attributes without doing the time-expansion. In some cases, the second approach is much better than the other approaches, since it gives strongly polynomial time algorithms, see e.g., Ford and Fulkerson [FF58, FF62] and Burkard, Dlaska, and Klinz [BDK93]. In general this approach is used when the network attributes are constant. The first and third approaches are commonly used when the network attributes are time-dependent. Minieka [Min74] used the second approach to solve a time-dependent problem in which arcs of the network may be added or removed in any time period. However, he indicated that, in the worst case, it is computationally more efficient to use the first approach than the second approach. *In this thesis, we use the third approach to solve discrete-time dynamic network flow problems with time-dependent attributes and show that, in some cases, this approach provides better computational complexity than the first one.*

Many dynamic network flow problems are considered as extensions of static network flow problems. These include maximum dynamic flow and minimum cost dynamic flow problems. Some examples of "original " dynamic network flow problems, where the time itself is in question, are the quickest flow and earliest arrival flow (or universal maximum flow). The maximum dynamic flow problem seeks a dynamic flow which sends as many as possible a commodity from a single source to a single sink of the network within the time horizon T. The minimum cost dynamic flow problem seeks a dynamic flow that minimizes the total shipment cost of a commodity in order to satisfy demands at certain nodes within T. The quickest flow problem seeks the minimum time T required to send a given flow value from the source to the sink. This problem is considered as the inverse of maximum dynamic flow problem. The earliest arrival flow problem is a variant of the maximum dynamic flow problem that seeks a dynamic flow which is maximum not only for T, but also for every time $T' < T$. *In this thesis we review some results on the maximum dynamic flow,*

earliest arrival flow, and quickest flow problems with constant attributes and develop some algorithms to solve these problems with time-dependent attributes.

When costs are equal to travel times, the supply of every node (except the sink) is one and all arc capacities are infinite, then the minimum cost dynamic flow problem reduces to the dynamic shortest path problem. There are various papers dealing with dynamic shortest path problems with time-dependent network attributes such as costs and travel times (i.e., time-dependent dynamic shortest path problems). These include the works of Cooke and Halsey [CH66], Orda and Rom [OR90, OR91], Ziliaskopoulos and Mahmassani [ZM93], and Wardell and Ziliaskopoulos [WZ00]. These problems consider only a single objective function. Without time dependency, there are several papers dealing with multicriteria (static) shortest path problems. These include the works of Hansen [Han80], Martins [Mar84], Corley and Moon [CM85], and Brumbaugh-Smith and Shier [BSS89]. Time-dependent multicriteria dynamic shortest path problems, as combination of the previous two problem classes, have attained relatively little attention in the literature. The works of Kostreva and Wiecek [KW93] and Getachew, Kostreva, and Lancaster [GKL00], to the best of our knowledge, are the only ones which have been published in the area of time-dependent multicriteria dynamic shortest path problems. *In this thesis we develop two algorithms to solve time-dependent bicriteria dynamic shortest path problems (TdBiDSP).* While the first algorithm deals only with the nonnegative network attributes, the second one allows the arcs of the network to have negative travel times and costs. *We develop the first algorithm to solve TdBiDSP with negative travel times and costs.* The possibility to wait (or park) at a node before departing on outgoing arc, in order to keep the total cost low, is also taken into account.

Most of the existing literature on computational testing rely on CPU time as the primary measure of performance. CPU time depends greatly on, for example, the specific computer used, chosen programming language, the programmer skills, etc.. Moreover, the typical CPU time analysis does not help us identify the bottleneck operations of the algorithm. To overcome these drawbacks, Ahuja, Magnanti, and Orlin [AMO93] proposed to use the so-called representative operation counts. These counts are determined by a small number of lines of code that represent the empirical behavior of the algorithm. By identifying the representative operation counts of an algorithm, we can use them to estimate the CPU time, to identify some asymptotic bottleneck operations, and to compare two algorithms. We will use this approach to analyze the performance of our proposed algorithms.

As an interesting application, *in this thesis we consider evacuation problems.* These problems are brought horrifically into focus by the New York World Trade Centre disaster on September 11, 2001. Figure 1.1 describes the chaos situation during an emergency evacuation. Evacuees must move to a safety area as quickly as possible in a crowded and smoky environment and also face the threat of falling debris from the ruined building or other disasters which may come along. Since time is a decisive parameter in these prob-

Figure 1.1: Walking through disaster

lems, dynamic models should be used instead of the static ones. In general, there are two approaches to model the evacuees distribution in order to estimate the evacuation time, namely *microscopic and macroscopic models*. In the microscopic models each evacuee is considered as a separate flow object, whereas in the macroscopic models, evacuees are treated as a homogeneous group where only common characteristics are taken into account. A survey of both approaches on evacuation problems was reported in Hamacher and Tjandra [HT02]. *In this thesis we focus on the macroscopic models and only review some of the results of the microscopic approaches.*

This thesis is organized as follows. In **Chapter 2**, we give the formulation of a discrete-time dynamic network flow problem (DTDNFP). Moreover we describe a time-expanded representation of a dynamic network which can be considered as a static network in the larger size. We use the idea of developing a residual static network to define the residual dynamic network. In **Chapter 3** we review approaches to solve the maximum dynamic flow problem with constant attributes. We give a dual formulation of the maximum dynamic flow problem with time-dependent attributes and its associated dynamic cut. A hybrid algorithm which combines the capacity scaling and the shortest augmenting path algorithms to solve the maximum dynamic flow problem with time-dependent attributes is discussed in this chapter. We also report a computational analysis of this algorithm. A special class of the maximum dynamic flow problem, known as the earliest arrival flow problem, is discussed in **Chapter 4**. After reviewing the approaches to solve the earliest arrival flow problem with constant attributes, we describe a new algorithm to solve the earliest arrival flow problem with time-dependent attributes and report a computational analysis. We also prove that the complexity of the new algorithm is reduced when infinite waiting is considered. In **Chapter 5** we review the approaches to quickest flow problems with constant attributes. By a simple network modification, we show that we can use the new algorithm proposed in Chapter 4 to solve the quickest flow problems with time-

dependent attributes. In **Chapter 6** we review the existing results on dynamic shortest path problems and multicriteria (static) shortest path problems. We propose two new algorithms, label setting and label correcting algorithms, to solve TdBiDSP and compare their performance to that of the existing algorithm. In **Chapter 7**, after reviewing both micro and macro approaches to the evacuation problems, we describe how to apply the results developed in the previous chapters to find optimal evacuee distributions and optimal evacuation paths. We apply our algorithms to find the lower bound of the evacuation time of Building 42 in the University of Kaiserslautern, Germany. Finally, in **Chapter 8** we conclude our discussion on dynamic network optimization with some ideas for possible further research works.

Chapter 2

Dynamic Network Flow

Dynamic network flow models describe network-structured, decision-making problems over a time horizon T. We can formulate the dynamic network flow problem in two ways depending on whether we use a discrete or continuous representation of time. The discrete-time dynamic network flow problem is a discrete-time expansion of a static network flow problem. In this case we distribute the flow over a set of predetermined time periods $t = 0, \ldots, T$. In a continuous-time dynamic network flow problem we look for the flow which is distributed continuously over time within the time horizon T, i.e., the time parameter is treated as a real number.

In this chapter, we focus our discussion on a discrete-time dynamic network flow. In the next section we give the definition of a discrete-time dynamic network flow problem. In Section 2.2 we describe the time-expanded representation of a discrete-time dynamic network. In Section 2.3 we discuss the First-In First-Out (FIFO) property of a discrete-time dynamic network with time-dependent attributes. Here attributes mean travel time, cost, and arc and node capacities. The development of a discrete-time dynamic residual network in Section 2.4 concludes this chapter.

2.1 Discrete-Time Dynamic Network Flow

A discrete-time dynamic network $G = (N, A, T)$ is a directed graph, where N is a set of nodes, A is a set of directed arcs, and T is a finite time horizon of interest discretized into the set $\{0, \ldots, T\}$. We denote the cardinality of N and A by n and m, respectively. Here we assume that 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 $u_{ij}(t) \in \mathbb{R}_0^+$, an associated time-dependent travel time $\lambda_{ij}(t) \in \mathbb{Z}_0^+$, and time-dependent cost $c_{ij}(t) \in \mathbb{R}$ for $t = 0, \ldots, T$. The time-dependent capacity $u_{ij}(t)$ defines the maximum number of flow units that can enter arc (i, j) at time t. But, it does not bound the total flow on that arc at a given time t.

The travel time $\lambda_{ij}(t)$ defines the time period needed to traverse the arc (i, j), depart-

ing 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. Therefore if a flow starts departing an arc from i to j at time t_0, then it will arrive at j at time $t_0 + \lambda_{ij}(t_0)$. This model of travel time is known as the *frozen arc model* (see e.g., Orda and Rom [OR90]). Another model of travel time is referred to as the *elastic arc model*. In this elastic arc model, a flow that starts departing an arc from i to j at time t_0 will arrive at j at the first instance of time $t_1 > t_0$ for which $t_1 - t_0 \geq \lambda_{ij}(t_1)$, i.e.,

$$t_1 := \min\{t' : t' > t_0,\ t' \geq t_0 + \lambda_{ij}(t')\} \tag{2.1}$$

This elastic arc model has a property that leaving node i earlier guarantees that the flow will arrive no later at node j along (i, j) than leaving later. This property is well known as the First-In First-Out (FIFO) property and will be discussed in more detail in Section 2.3. In the frozen arc model non-FIFO behavior is possible, therefore it is considered more general than the elastic arc model. In this thesis *we focus on the frozen arc model.*

Each node i of the network G has a time-dependent node capacity $a_i(t) \in \mathbb{R}_0^+,\ t = 0, \ldots, T$ which defines the maximum number of flow units that can be held over one time unit at node i.

As in the static network, there is also a vector of demand-supply of the network nodes at time $t \in \{0, \ldots, T\}$ denoted by $\mathbf{q}(t) = (q_i(t))_{i \in N}$ as described below.

$$q_i(t) \begin{cases} > 0 & \text{, node } i \text{ is a source (supply node) at time } t \\ < 0 & \text{, node } i \text{ is a sink (demand node) at time } t \\ = 0 & \text{, node } i \text{ is a transshipment node at time } t \end{cases} \tag{2.2}$$

Different from the static network, in dynamic network flow models there may exist some nodes as transshipment nodes at one time and then become source nodes later, and vice versa.

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

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

For every $t \in \{0, \ldots, T\}$, the value $x_{ij}(t)$ determines the number of movement 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 may limit the flow $x_{ij}(t)$ for time t with $t + \lambda_{ij}(t) \leq T$. Flows from node i at time t to the same node with travel time $\lambda_{ii}(t) = 1$ represent the amount of holdover flows. This flow is denoted by $x_{ii}(t)$.

To determine the total flow arriving at a given node i at a given time t, we need to consider flows on every predecessor node j of i at every departure time t' in which the sum

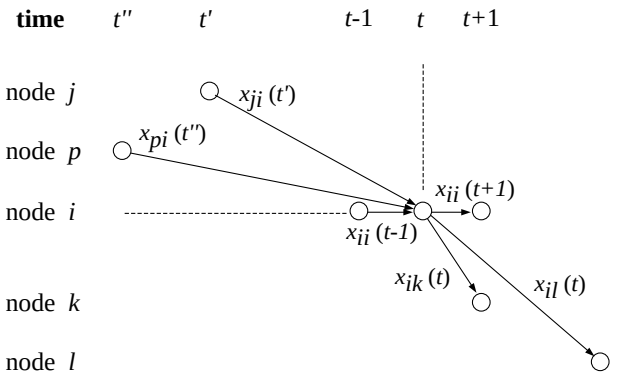


Figure 2.1: Conservation flows

of t' and the travel time $\lambda_{ji}(t')$ equals the given arrival time t (see Figure 2.1). Therefore, the total flow arriving at node i at time t is given by

$$\sum_{(j,i)\in A\cup\{(i,i)\ :\ i\in N\}} \ \sum_{\{t':t'+\lambda_{ji}(t')=t\}} x_{ji}(t')$$

Let $\phi_T(\mathbf{x}(t))$ be the general objective function of a discrete-time dynamic network flow problem (DTDNFP) with argument $\mathbf{x}(t)$. The formulation of DTDNFP is given by

$$\text{min (or max)} \quad \phi_T(\mathbf{x}) = \sum_{t=0}^{T} \phi_T(\mathbf{x}(t)) \tag{2.3}$$

$$\sum_{(j,i)\in A} \ \sum_{\{t':t'+\lambda_{ji}(t')=t\}} x_{ji}(t') -$$

$$\sum_{\{(i,j)\ :\ (i,j)\in A,\ t+\lambda_{ij}(t)\le T\}} x_{ij}(t) = x_{ii}(t) - x_{ii}(t-1) - q_i(t),\ i\in N,\ t=0,\dots,T \tag{2.4}$$

$$0 \le x_{ij}(t) \le u_{ij}(t),\ (i,j)\in A,\ t+\lambda_{ij}(t)\le T \tag{2.5}$$

$$0 \le x_{ii}(t) \le a_i(t),\ i\in N,\ t=0,\dots,T \tag{2.6}$$

Equation (2.4) formulates the discrete-time dynamic network flow conservation constraints. The formulation of movement flow and holdover flow (i.e., waiting) capacity constraints are given by (2.5) and (2.6), respectively.

Now, let us discuss the integrality property of DTDNFP. Let $\mathbf{A}$ be an $n(T+1)\times(m+n)(T+1)$ matrix, called the node-arc incidence matrix of $G=(N,A,T)$. The rows of $\mathbf{A}$ are indexed by it with $i\in N$ and $t\in\{0,\dots,T\}$, while the columns are indexed by ijt with $(i,j)\in A\cup\{(i,i)\ :\ i\in N\}$. Each column ijt of matrix $\mathbf{A}$ corresponds to the variable $x_{ij}(t)$ with $(i,j)\in A\cup\{(i,i)\ :\ i\in N\}$, $t\in\{0,\dots,T\}$. The column ijt of matrix $\mathbf{A}$ has a $+1$ in the it-th row, a -1 in the $j(t+\lambda_{ij}(t))$-th row, and the rest of its

entries are zero. We denote the vector of supply or demand of size $n(T+1)$ and the vector of holdover capacity of size $n(T+1)$ by $\mathbf{q}$ and $\mathbf{a}$, respectively. Let's also define $\mathbf{I}$ as an $(m+n)(T+1) \times (m+n)(T+1)$ identity matrix and $\mathbf{u}'$ as an $m(T+1)+n(T+1)$ vector. Vector $\mathbf{u}'$ combines vectors $\mathbf{u}$ of arc capacities and $\mathbf{a}$ of holdover capacities as

$$\mathbf{u}' := \begin{pmatrix} \mathbf{u} \\ \cdots \\ \mathbf{a} \end{pmatrix}$$

Hence, in matrix form, DTDNFP can be represented as follows.

$$\begin{aligned} \text{min (or max)} \quad & \phi_T(\mathbf{x}) \\ \text{Subject to} \quad & \\ \mathbf{Ax} &= \mathbf{q} && (2.7) \\ \mathbf{Ix} &\leq \mathbf{u}' && (2.8) \\ \mathbf{x} &\geq \mathbf{0} && (2.9) \end{aligned}$$

Theorem 2.1 *The node-arc incidence matrix* $\mathbf{A}$ *of* $G = (N, A, T)$ *is totally unimodular.*

Proof:
The proof is done when we can show that every $k \times k$ submatrix $\mathbf{B}$ of $\mathbf{A}$ has determinant $0, +1$, or -1. We do the proof by using induction on k. Since each element of $\mathbf{A}$ is $0, +1$, or -1, the theorem is true for $k = 1$. Suppose that the theorem holds for some k. Let $\mathbf{B}$ be any $(k+1) \times (k+1)$ submatrix of $\mathbf{A}$. This matrix satisfies exactly one of the three following possibilities:

1. $\mathbf{B}$ contains a column with only zero elements,

2. every column of $\mathbf{B}$ has exactly two nonzero elements, in which case, the product of these two elements must be -1,

3. some j-th column of $\mathbf{B}$ have exactly one nonzero element.

In case 1, the theorem holds since the determinant of $\mathbf{B}$ is zero. In case 2, the rows in $\mathbf{B}$ are linearly dependent, since summing all of the rows in $\mathbf{B}$ yields a zero vector. Consequently, the determinant of $\mathbf{B}$ is zero. In case 3, let i be the row of $\mathbf{B}$ which has exactly one nonzero element. Let $\mathbf{B}'$ be the submatrix of $\mathbf{B}$ obtained by deleting the i-th row and the j-th column. Then the absolute value of the determinant of $\mathbf{B}$ equals that of $\mathbf{B}'$. By the induction hypothesis, the determinant of $\mathbf{B}'$ is $0, +1$, or -1, so the determinant of $\mathbf{B}$ is also $0, +1$, or -1. ■

To describe the integrality property of the optimal solution of DTDNFP, we need the following unimodularity theorem.

Theorem 2.2 (Unimodularity theorem, see e.g., Ahuja, Magnanti, and Orlin [AMO93])
Let $\mathcal{A}$ be an integer matrix with linear independent rows. Every basic feasible solution defined by the constraints $\mathcal{A}\mathbf{z} = \mathbf{b},\ \mathbf{z} \geq 0$ is integer for any integer vector $\mathbf{b}$ if and only if $\mathcal{A}$ is unimodular.

Using this theorem, we can establish the following result.

Theorem 2.3 *If $\mathbf{u}, \mathbf{a}$, and $\mathbf{q}$ are integer vectors, then DTDNFP has an integer optimal dynamic flow.*

Proof:
Since constraints (2.4)-(2.6) define a compact feasible region of DTDNFP, *the fundamental theorem of linear programming* (see e.g., Hamacher and Klamroth [HK00]) implies that DTDNFP has an optimal solution. Let matrix $\mathbf{A}$ be partitioned into matrices $\mathbf{A_u}$ and $\mathbf{A_a}$ of size $n(T+1)\times m(T+1)$ and $n(T+1)\times n(T+1)$, respectively. Vector $\mathbf{x}$ is also partitioned into vectors $\mathbf{x_u}$ and $\mathbf{x_a}$ of size $m(T+1)$ and $n(T+1)$, respectively. Vector $\mathbf{x_u}$ corresponds to the movement flow $x_{ij}(t), (i,j) \in A,\ t \leq T$ and vector $\mathbf{x_a}$ corresponds to the holdover flow $x_{ii}(t), i \in N,\ t \leq T$. By adding a vector $\mathbf{y_u}$ of size $m(T+1)$ as the slack variables of the arc capacity constraints and a vector $\mathbf{y_a}$ of size $n(T+1)$ as the slack variables of the holdover capacity constraints, (2.8) can be written as an equality. The constraints of DTDNFP can thus be constructed as

$$\begin{pmatrix} \mathbf{A_u} & \vdots & \mathbf{A_a} & \vdots & \mathbf{0} & \vdots & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{I} & \vdots & \mathbf{0} & \vdots & \mathbf{I} & \vdots & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{0} & \vdots & \mathbf{I} & \vdots & \mathbf{0} & \vdots & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{x_u} \\ \cdots \\ \mathbf{x_a} \\ \cdots \\ \mathbf{y_u} \\ \cdots \\ \mathbf{y_a} \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ \cdots \\ \mathbf{u} \\ \cdots \\ \mathbf{a} \end{pmatrix} \tag{2.10}$$

Let us define $\mathcal{A}$, $\mathbf{z}$, and $\mathbf{b}$ as follows.

$$\mathcal{A} := \begin{pmatrix} \mathbf{A_u} & \vdots & \mathbf{A_a} & \vdots & \mathbf{0} & \vdots & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{I} & \vdots & \mathbf{0} & \vdots & \mathbf{I} & \vdots & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{0} & \vdots & \mathbf{I} & \vdots & \mathbf{0} & \vdots & \mathbf{I} \end{pmatrix}, \mathbf{z} := \begin{pmatrix} \mathbf{x_u} \\ \cdots \\ \mathbf{x_a} \\ \cdots \\ \mathbf{y_u} \\ \cdots \\ \mathbf{y_a} \end{pmatrix}, \mathbf{b} := \begin{pmatrix} \mathbf{q} \\ \cdots \\ \mathbf{u} \\ \cdots \\ \mathbf{a} \end{pmatrix}$$

Since $\mathbf{A}$ is totally unimodular (TU), $\mathcal{A}$ is TU. Therefore $\mathcal{A}$ is unimodular. Consequently, Theorem 2.2 implies that there is an integer optimal dynamic flow $\mathbf{x}$ of DTDNFP. ■

2.2 Time-Expanded Network

DTDNFP requires to keep track of when each unit travels through an arc so that no arc capacity is violated at any time. Therefore, finding an optimum solution of DTDNFP is more complicated than that of static network flow problems. However, this complication can be resolved by transforming DTDNFP into a static network flow problem on a time-expanded replica of the original network.

Definition 2.2 (Time-dependent version of the definition given in Ford and Fulkerson [FF58]) *The time expansion of $G = (N, A, T)$ over a time horizon T defines a time-expanded network $G_T = (N_T, A_T)$ where*

$$N_T := \{i(t) \ : \ i \in N, \ t = 0, \ldots, T\}$$

and A_T consists of the set of movement arcs

$$A^M := \{(i(t), j(t') \ : \ (i,j) \in A, \ t' = t + \lambda_{ij}(t) \leq T\}$$

and the set of holdover arcs

$$A^H := \{(i(t), i(t+1)) \ : \ i \in N, \ t = 0, 1, \ldots, T-1\}$$

i.e.,

$$A_T := A^M \cup A^H$$

The capacity $u_{ij}(t)$ of the movement arc $(i(t), j(t'))$ is determined by the capacity $u_{ij}(t)$ and the capacity $u_{ii}(t)$ of the holdover arc $(i(t), i(t+1))$ is determined by the node capacity $a_i(t)$.

Figure 2.2 shows a time expansion of a network for $T = 4$. Every static flow $\mathbf{f}$ in G_T from the sources $s(0), \ldots, s(T)$ to the sinks $d(0), \ldots, d(T)$ corresponds to a dynamic flow $\mathbf{x}$ in G, and vice versa. The one-to-one correspondence is given by

$$x_{ij}(t) = f_{i(t),j(t+\lambda_{ij}(t))}$$

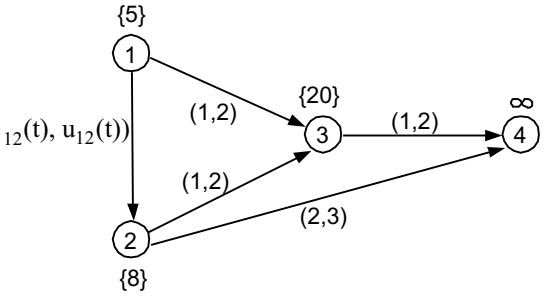
Since static flows in G_T and dynamic flows in G are equivalent, discrete-time dynamic network flow problems can always be solved as static network flow problems in the larger network. Thus, no additional algorithm is required to solve the dynamic network flow problems. However, if T is very large, then the network G_T becomes very large and the number of calculations needed to solve the dynamic flow problems on G_T becomes prohibitively large. The dependence of the network size on T is a disadvantage of this approach. In Section 3.3, we will discuss an approach to avoid this disadvantage.

In a discrete-time dynamic network, the time period t depends on the basic unit θ in which the travel times are measured. For example, if we choose 5 seconds as the length of

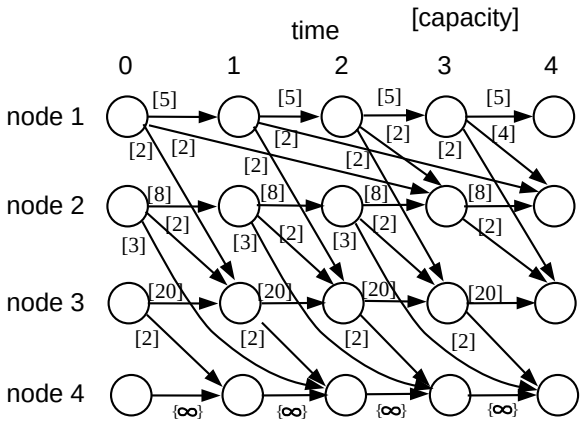
$\{a_i(t)\}$
$(\lambda_{ij}(t), u_{ij}(t))$

$$\lambda_{12}(t) := \begin{cases} 3 & , t \leq 1 \\ 1 & , t > 1 \end{cases} \quad ; \quad u_{12}(t) := \begin{cases} 2 & , t \leq 2 \\ 4 & , t > 2 \end{cases}$$

The other arcs have constant travel times and capacities.
Every node has constant capacity.



(a) The network $G = (N, A, T)$



(b) The associated time-expanded network G_T with $T = 4$

Figure 2.2: The network $G = (N, A, T)$ and its associated time-expanded network G_T

the basic unit (i.e., $\theta = 5$), then specifying three time periods for traversing an arc (i, j) at time t (i.e., $\lambda_{ij}(t) = 3$) means we need fifteen seconds to do so. The number of time periods T is obtained by dividing the planning horizon of interest by the length θ of the basic unit. The smaller θ, the more accurately the model represents the actual flow's evolution. If θ tends to zero, then the solution of the discrete-time dynamic network flow problems will approximate well the solution of the corresponding continuous-time dynamic network flow problems.

Now consider the case when the travel times are constant, i.e.,

$$\lambda_{ij}(t) := \lambda_{ij}, \ \forall (i, j) \in A, \ t \in \{0, \ldots, T\}$$

The upper bound of the number of nodes and arcs in G_T are described by the following proposition.

Proposition 2.1 *Let n and m denote the cardinality of N and A, respectively. If the travel times are constant, then $n(T+1)$ and $(n+m)T+m-\sum_{(i,j)\in A}\lambda_{ij}$ is the upper bound of the number of nodes and arcs in G_T, respectively.*

Proof:
Since we generate at most $(T+1)$ copies of each node, the upper bound of the number of nodes is $n(T+1)$. A movement arc $(i,j) \in A,\ i \neq j$, is repeated from $t=0$ till at most $t = T - \lambda_{ij}$, i.e., it is repeated $(T+1-\lambda_{ij})$ times. Since there are m arcs, the total number of movement arcs in A_T is $m(T+1) - \sum_{(i,j)\in A}\lambda_{ij}$. With additional holdover arcs $(i(t), i(t+1))$, we get the upperbound of the number of arcs in G_T. ■

Hence, $G_T = (N_T, A_T)$ has $\mathcal{O}(nT)$ many nodes and $\mathcal{O}((n+m)T)$ many arcs. In fact, we will never use any path from the source which arrives at the sink after time T. Therefore, we can reduce the size of the time-expanded network by eliminating these paths (including the corresponding nodes). To build a compact time-expanded network based on the dynamic network G with a single source node s and a single sink node d, we first describe some important definitions.

Definition 2.3

- *If P is a path in G, we denote by $\lambda(P)$ the length of P with respect to the travel times as given by*

$$\lambda(P) := \sum_{(i,j)\in P} \lambda_{ij} \tag{2.11}$$

- *We denote by λ_i and $\overline{\lambda}_i$ the length of shortest path from the source node s to the node i and the length of shortest path from the node i to the sink node d, respectively. For every node $i \in N$, we define a set of times T_i for which node i at time $t \in T_i$ is reachable from the source and also can reach the sink within the time horizon T, as follows*

$$T_i := \{t+\lambda_i \ :\ t+\lambda_i+\overline{\lambda}_i \leq T, t \in \{0,\ldots,T\}\} \tag{2.12}$$

- *For every arc $(i,j) \in A$, we define a set of times T_{ij} for which node i of arc (i,j) at time $t \in T_{ij}$ is reachable from the source and the sink is also reachable within T from node j at time $t+\lambda_{ij}$, as follows*

$$T_{ij} := \{t+\lambda_i \ :\ t+\lambda_i+\lambda_{ij}+\overline{\lambda}_j \leq T, t \in \{0,\ldots,T\}\} \tag{2.13}$$

By definition, we have $\lambda_s = 0$ and $T_s = \{0, \ldots, T - \overline{\lambda}_s\}$.

The set T_i determines a set of times where we can copy the node i and guarantee that there exists a path from the source to the sink that passes through node i at time $t \in T_i$ within the time horizon T. Whereas the set T_{ij} determines a set of times where we can copy the arc (i, j) and guarantee that the time-copy of this arc belongs to at least one path from the source to the sink within T. While the values of λ_i for all $i \in N$ can be calculated by using the forward version of Djikstra's label setting algorithm, the values of $\overline{\lambda}_i$ for all $i \in N$ can be obtained by applying the backward version of this algorithm. Therefore the sets T_i and T_{ij} can be found for all $i \in N$ and $(i, j) \in A$ in $\mathcal{O}(n^2)$. A detailed procedure is summarized in Algorithm 2.1.

Algorithm 2.1 : Building a compact G_T from G with constant travel times

INPUT	Network $G = (N, A, T)$ with constant travel times.
OUTPUT	Time-expanded network $G_T = (N_T, A_T)$.
0	Calculate λ_i and $\overline{\lambda}_i$ for every $i \in N$.
1	Determine T_i for every $i \in N$ and T_{ij} for every $(i, j) \in A$.
2	Determine $N_T := \{i(t) \ : \ i \in N \ ; \ t \in T_i\}$, $A^M := \{(i(t), j(t')) \ : \ (i, j) \in A \ ; \ t \in T_{ij}\}$ and $A^H := \{(i(t), i(t+1)) \ : \ i \in N - \{d\} \ ; \ t, t+1 \in T_i\}$ $A_T := A^M \cup A^H$

To deal with multiple sources and sinks, one can add a super source s^* and super sink d^* to the original network. Let S and D be the set of sources and sinks, respectively. We then build arcs (s^*, s) for every $s \in S$ and (d, d^*) for every $d \in D$ with zero travel times and infinite capacities. Consequently, the definitions of λ_i and $\overline{\lambda}_i$ are modified with respect to s^* and d^*. Figure 2.3 shows a compact version of G_T in Figure 2.2(b) under the assumption that all arcs have constant capacities and travel times. Note that the network in Figure 2.3 has three sources, namely node 1, 2, and 3, and a single sink, node 4.

Since the time-expanded network may have several time-copies of each source node and each sink node, we may introduce a *super source* s^T and a *super sink* d^T to create a single source/single sink time-expanded network G_T (see Figures 2.4 and 2.5). How the super source is connected to the source is actually depends on the problem. In the discrete-time maximum dynamic network flow problem (DTMDNFP) that will be discussed in Chapter 3, the super source is connected to all time-copies of the source nodes. Arcs from the super source to every time-copy of the source node have zero travel time and infinite capacities. In this case, we do not have holdover arcs for source nodes which do not have predecessors

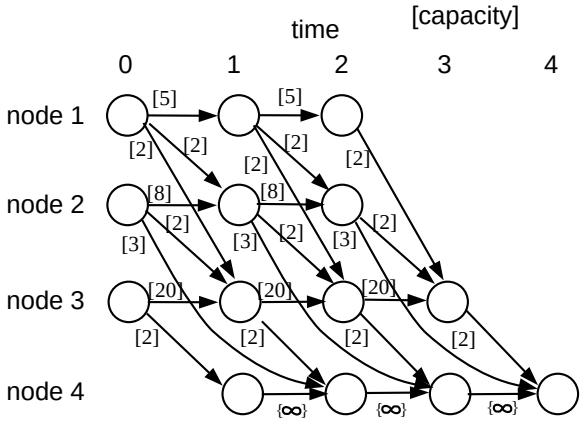


Figure 2.3: Compact version of G_T for $T = 4$ in Figure 2.2 under an assumption that $\lambda_{12}(t) := 1,\ t \leq 4$ and $u_{12}(t) := 2,\ t \leq 4$

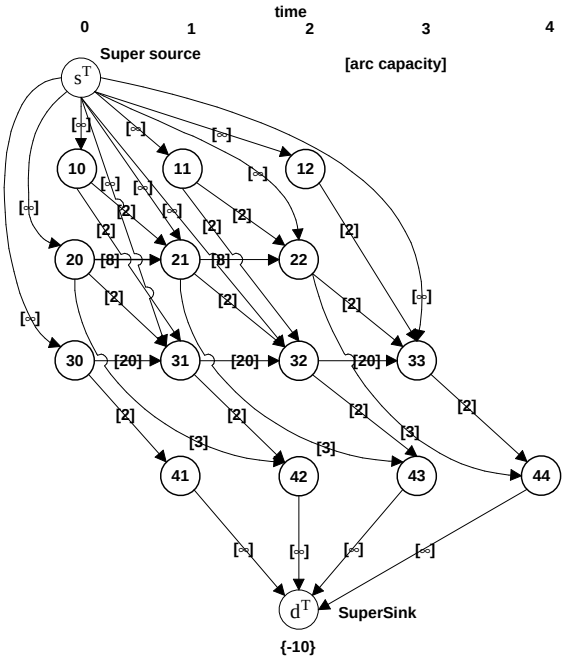


Figure 2.4: Time-expanded network G_T with $T = 4$ for the discrete-time maximum dynamic network flow problem

as shown in Figure 2.4 (e.g., node 1 in Figure 2.2a). In the discrete-time minimum cost dynamic network flow problem (DTMCDNFP), the super source is connected only to the time zero copy of the source nodes (see Figure 2.5). In this case, we may have holdover arcs for source nodes. Arcs from the super source have zero travel time and their capacities are equal to the initial occupancies. On the other hand, for every problem, generally all time-copies of every sink node are connected to the super sink and there is no holdover arc

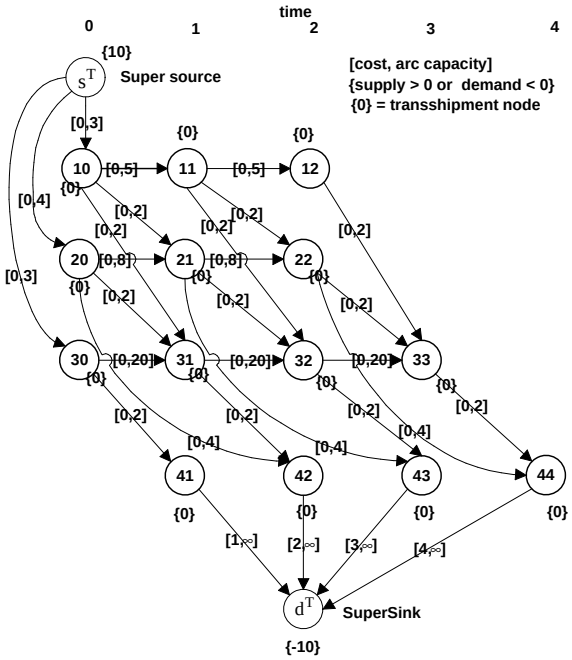


Figure 2.5: Time-expanded network G_T with $T = 4$ for the discrete-time minimum cost dynamic network flow problem

for the sink nodes. All connections to the super sink have zero travel times and infinite flow capacities.

2.3 First-In First-Out

Since the travel times are not constant, we may have two possible properties, First-In First-Out (FIFO) and non-FIFO. An arc $(i, j) \in A$ is said to have a FIFO property if leaving node i earlier guarantees that one will arrive no later at node j along (i, j) than leaving i later. Hence, in FIFO, the travel time associated with arc (i, j) is a nondecreasing function, i.e.,

$$t' + \lambda_{ij}(t') \leq t'' + \lambda_{ij}(t''),\ t' < t''$$

Therefore, waiting at node i before traversing arc (i, j) is not necessary, since it will not save the time. This FIFO property is also known as *non-overtaking* property in traffic disciplines.

Definition 2.4 (e.g., Pallottino and Scutella [PS97]) *An arc having a FIFO property is called a FIFO arc. A dynamic network $G = (N, A, T)$ is called FIFO network when all of its arcs are FIFO.*

On the other hand, when the FIFO property is not satisfied, sometimes it may give an advantage to wait for a certain amount of time at the beginning node of an arc before

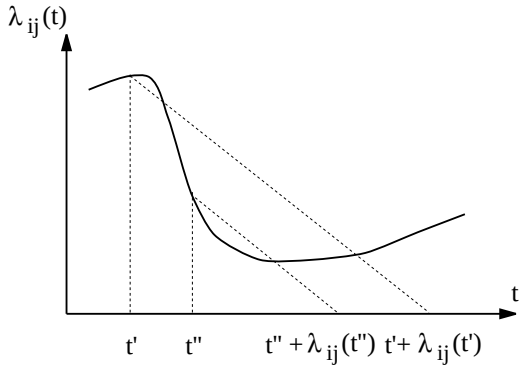


Figure 2.6: Non-FIFO property

traversing that arc. Figure 2.6 describes a non-FIFO property.

Recal the definition of a travel time according to the elastic arc model discussed in Section 2.1.

Property 2.1 *FIFO property is maintained in the elastic arc model.*

Proof:
Consider two different departure times t_1 and t_2 with $t_1 < t_2$. Let $t_1^{'}$ and $t_2^{'}$ be the associated arrival times of t_1 and t_2 along an arc (i, j), respectively. FIFO property is maintained if $t_1^{'} \leq t_2^{'}$. By (2.1), we obtain

$$t_1^{'} = \min\{t^{'} : \ t^{'} > t_1, \ t^{'} \geq t_1 + \lambda_{ij}(t^{'})\}$$

and

$$t_2^{'} = \min\{t^{'} : \ t^{'} > t_2, \ t^{'} \geq t_2 + \lambda_{ij}(t^{'})\}$$

Let B_1 and B_2 be the set $\{t^{'} : \ t^{'} > t_1, \ t^{'} \geq t_1 + \lambda_{ij}(t^{'})\}$ and $\{t^{'} : \ t^{'} > t_2, \ t^{'} \geq t_2 + \lambda_{ij}(t^{'})\}$, respectively. Lets take any $t^* \in B_2$. Since $t_1 < t_2$, $t^* > t_1$ and $t^* \geq t_1 + \lambda_{ij}(t^*)$, i.e., $t^* \in B_1$. Therefore $B_2 \subset B_1$ that implies $t_1^{'} \leq t_2^{'}$. ■

On the other hand, a travel time according to the frozen arc model, in general, does not maintain the FIFO property.

2.4 Residual Dynamic Network

Here we extend the idea of residual network in the case of static network (see e.g., Ahuja, Magnanti, and Orlin [AMO93]) to the dynamic one. Suppose that arc (i, j) at time t carries $x_{ij}(t)$ units of flow. Then we can send the 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. Whereas sending a unit flow from i departing at time t to j along (i,j) increases the flow cost by $c_{ij}(t)$ units, sending flow in the reverse direction from j departing at time $t+\lambda_{ij}(t)$ to i on the same arc decreases the flow cost by $c_{ij}(t)$ units.

The similar ideas can be applied to the waiting capacities. Suppose that $x_{ii}(t)$ units occupy node i at time t for at least one unit of 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. We denote by $a_i^{x+}(t)$ the maximum free waiting space at node i at time t. The capacity of waiting canceling at node i at time t is denoted by $a_i^{x-}(t+1)$.

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

Definition 2.5 *The residual dynamic network with respect to a given feasible dynamic flow* $\mathbf{x}$ *is defined as* $G_x := (N, A_x^+ \cup A_x^-, T)$ *with* $A_x := A_x^+ \cup A_x^-$ *where*

$$A_x^+ := \{(i,j) : (i,j) \in A,\ \exists t \le T - \lambda_{ij}(t) \text{ with } u_{ij}(t) > x_{ij}(t)\} \tag{2.14}$$

and

$$A_x^- := \{(i,j) : (j,i) \in A,\ \exists t \le T - \lambda_{ji}(t) \text{ with } x_{ji}(t) > 0\} \tag{2.15}$$

G_x *is provided with attributes*

- *residual travel times*

$$\lambda_{ij}^x(t) := \begin{cases} \lambda_{ij}(t) & ,(i,j) \in A,\ t+\lambda_{ij}(t) \le T \\ -\lambda_{ji}(t') & ,(j,i) \in A,\ t'+\lambda_{ji}(t') = t \le T,\ x_{ji}(t') > 0 \end{cases} \tag{2.16}$$

- *residual arc costs*

$$c_{ij}^x(t) := \begin{cases} c_{ij}(t) & ,(i,j) \in A,\ t+\lambda_{ij}(t) \le T \\ -c_{ji}(t') & ,(j,i) \in A,\ t'+\lambda_{ji}(t') = t \le T,\ x_{ji}(t') > 0 \end{cases} \tag{2.17}$$

- *residual arc capacities*

$$u_{ij}^x(t) := \begin{cases} u_{ij}(t) - x_{ij}(t) & ,(i,j) \in A,\ t+\lambda_{ij}(t) \le T \\ x_{ji}(t') & ,(j,i) \in A,\ t'+\lambda_{ji}(t') = t \le T \end{cases} \tag{2.18}$$

- *residual waiting capacities*

$$a_i^{x+}(t) := a_i(t) - x_{ii}(t),\ i \in N,\ t \leq T, \tag{2.19}$$

$$a_i^{x-}(t+1) := x_{ii}(t),\ i \in N,\ t < T, \tag{2.20}$$

Since the travel times are time-dependent, there may exist an arrival time t at node j corresponds to k different departure times $t_1, \ldots, t_k$ from node i along the arc $(i,j) \in A$, i.e., $t^{'} + \lambda_{ij}(t^{'}) = t,\ \forall t^{'} \in \{t_1, \ldots, t_k\}$. 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 2.7.

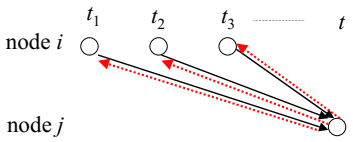


Figure 2.7: A single arrival time associates with several departure times. The dashed lines correspond to the backward arcs

Chapter 3

Maximum Dynamic Network Flows

The problem of maximum dynamic flow was introduced by Ford and Fulkerson [FF58, FF62]. This problem seeks a discrete-time dynamic flow which sends a commodity as many as possible from a single source to a single sink within the time horizon T, without exceeding the arc capacity. All flow leaving the source must arrive at the sink by the time horizon T. Ford and Fulkerson showed that if the network has constant or fixed travel times and capacities, then the solution of maximum dynamic flow problem can be obtained by solving a minimum cost circulation flow problem on the static network. Therefore, this maximum dynamic flow problem is polynomially solvable. The maximum dynamic flow is obtained by repeating a path decomposition of the minimum cost static flow as long as possible within T. This technique is called the *temporally repeated flow*. Minieka [Min74] modified this temporally repeated flow technique to solve a similar problem under a weaker assumption. Here it is assumed that arcs of the network may be added or removed in any time period. He indicated that if the number of arc changes is excessive, then it is computationally more efficient to apply the static maximum flow algorithm to the time-expanded network. For a network with multiple sources, Hoppe and Tardos [HT94] introduced a dynamic flow for the time horizon T that *lexicographically* maximizes the amounts leaving the sources. They showed that this *lexicographic maximum dynamic flow* can be obtained in a polynomial time. Anderson, Nash, and Philpott [APP82] generalized the discrete-time maximum dynamic flow to the continuous-time maximum dynamic flow. They assumed that the travel times are zero but the capacity may vary with time. To obtain an optimal dynamic flow, they developed a continuous-time version of Ford and Fulkerson's maximum static flow labeling algorithm. They also defined a continuous-time dynamic cut and proved the continuous-time version of static maximum flow-minimum cut theorem. Philpott [Phi90] relaxed the assumption of zero travel times from the previous model into constant nonnegative travel times.

In this chapter we focus the discussion on discrete-time maximum dynamic network flow problems. Three main topics will be covered

- the temporally repeated flow technique used to solve a maximum dynamic flow prob-

lem with constant attributes,

- the dynamic cut and its properties, and
- a time-dependent shortest augmenting path algorithm with a capacity scaling to solve a maximum dynamic flow problem with time-dependent attributes.

In the next section, we formulate a maximum dynamic flow problem. In Section 3.2 we give a dual formulation of the maximum dynamic flow problem, formal definition of a dynamic cut, and discuss a dynamic version of the maximum flow-minimum cut theorem. In Section 3.3 we review the temporally repeated flow technique of Ford and Fulkerson. The time-dependent shortest augmenting path algorithm with capacity scaling is discussed in Section 3.4. Computational results on several examples based on randomly generated networks in Section 3.5 conclude this chapter.

3.1 Problem Formulation

Given a time horizon T, a discrete-time $s-d$ maximum dynamic network flow problem (DTMDNFP) seeks a discrete-time dynamic flow which sends a commodity as many as possible from a single source s to a single sink d within the time horizon T, without exceeding the arc and node capacity. This problem assumes that there are neither supplies nor demands at every node and at every time, i.e., $q_i(t) = 0,\ \forall i \in N,\ \forall t$. Moreover, without loss of generality, we assume that there are no outgoing arc from the sink d and no incoming arc to the source s in the network $G = (N, A, T)$. By referring to the general DTDNFP discussed in the previous chapter, the objective function of a DTMDNFP is defined as follows.

$$\phi_T(\mathbf{x}) = V_{\sum^T}(\mathbf{x}) := \sum_{t=0}^{T} \sum_{(i,d)\in A} \sum_{\{t':t'+\lambda_{id}(t')=t\}} x_{id}(t') \tag{3.1}$$

Here, $V_{\sum^T}(\mathbf{x})$ denotes *the value of a discrete-time dynamic flow* $\mathbf{x}$ *for a time horizon* T. The formulation of a DTMDNFP is given as follows.

$$\text{(DTMDNFP)} \qquad \max\ V_{\sum^T}(\mathbf{x}) \tag{3.2}$$

Subject to

$$\sum_{(j,i)\in A} \sum_{\{t':t'+\lambda_{ji}(t')=t\}} x_{ji}(t') - \sum_{\{(i,j)\ :\ (i,j)\in A,\ t+\lambda_{ij}(t)\le T\}} x_{ij}(t) = x_{ii}(t) - x_{ii}(t-1),\ i \in N - \{s,d\},\ t = 0,\ldots,T \tag{3.3}$$

$$0 \le x_{ij}(t) \le u_{ij}(t),\ (i,j) \in A,\ t + \lambda_{ij}(t) \le T \tag{3.4}$$

$$0 \le x_{ii}(t) \le a_i(t),\ i \in N - \{s,d\},\ t = 0,\ldots,T \tag{3.5}$$

We denote *the value of maximum dynamic flow for a time horizon T* by $V_{\sum^T}$. Using the problem classification presented in Appendix D, we then denote this problem by

$$(s,d)/(\lambda(t),u(t),a(t))/V_{\sum^T}$$

As a consequence of Theorem 2.3, we establish the following integrality property of DT-MDNFP.

Theorem 3.1 *If* **u** *and* **a** *are integer vectors, then DTMDNFP has an integer optimal dynamic flow.*

3.2 Dynamic Cut

Since the discrete-time dynamic network can be considered as a static network in the larger network, the results on dynamic cuts follow directly from those of static network. However, the definition and properties of a dynamic cut must take into account the effect of travel times. This condition motivates the discussion on this section.

A dynamic cut is considered as a dynamic extension of the static cut.

Definition 3.1 (Dynamic Cut)

- *Consider two set-valued functions*

$$C_T : \{0,\ldots,T\} \rightarrow 2^N \tag{3.6}$$
$$\text{and} \quad \overline{C}_T(t) = N - C_T(t) \tag{3.7}$$

that satisfies $s \in C_T(t)$ 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

$$\begin{aligned}(C_T,\overline{C}_T) := \; & \{\,(i(t),j(t+\lambda_{ij}(t))) : i \in C_T(t), j \in \overline{C}_T(t+\lambda_{ij}(t)),\\ & \quad t+\lambda_{ij}(t) \leq T,\ (i,j) \in A\} \cup\\ & \{(i(t),i(t+1)) : i \in C_T(t) \cap \overline{C}_T(t+1),\ i \in N-\{s,d\},\ t<T\}\end{aligned} \tag{3.8}$$

- *The set of times when a movement arc $(i,j) \in A$ crosses a dynamic cut $(C_T, \overline{C}_T)$ is determined by*

$$\Gamma_{ij}^T := \{t \ : \ i \in C_T(t),\ j \in \overline{C_T}(t+\lambda_{ij}(t)),\ t+\lambda_{ij}(t) \leq T\} \tag{3.9}$$

and the set of times when the holdover arc of node i crosses $(C_T, \overline{C}_T)$ is determined by

$$\Gamma_{ii}^T := \{t \ : \ i \in C_T(t) \cap \overline{C_T}(t+1)\},\ i \in N-\{s,d\} \tag{3.10}$$

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

$$W_{\sum^T}(C_T) := \sum_{(i,j)\in A}\sum_{t\in\Gamma_{ij}^T} u_{ij}(t) + \sum_{i\in N-\{s,d\}}\sum_{t\in\Gamma_{ii}^T} a_i(t) \tag{3.11}$$

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

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

The value of a dynamic cut as defined by (3.11) contains contributions from capacities of arcs crossing the cut and from the node capacities at points in time where some nodes in N pass from the source side to the sink side of the cut.

Remark 3.1 *The definition of Γ_{ii}^T can be derived from the definition of Γ_{ij}^T by defining $\lambda_{ii}(t) = 1$ for all $i \in N - \{s,d\}$ and $t = 0, \ldots, T-1$.*

Lemma 3.1 (Net flow crossing the $s-d$ dynamic cut) *The net movement and waiting flow crossing an $s-d$ dynamic cut $(C_T, \overline{C}_T)$ is equal to the value of this flow.*

Proof :
Suppose that $\mathbf{x}$ is a feasible dynamic flow. By the conservation flow constraint (3.3), the value $V_{\sum^T}(\mathbf{x})$ of $\mathbf{x}$ can be described by

$$V_{\sum^T}(\mathbf{x}) = \sum_{t=0}^{T} \sum_{\{(s,j)\ :\ (s,j)\in A,\ t+\lambda_{sj}(t)\leq T\}} x_{sj}(t)$$

Since $s \in C_T(t)$ and $d \notin C_T(t)$ for all $t \in \{0, \ldots, T\}$, by adding (3.3) for all nodes $i \in C_T(t)$, for all $t \in \{0, \ldots, T\}$ and $V_{\sum^T}(\mathbf{x})$, we obtain

$$\begin{aligned} V_{\sum^T}(\mathbf{x}) \;=\; & \sum_{t=0}^{T}\sum_{i\in C_T(t)}\Bigg[\sum_{\{j\ :\ (i,j)\in A,\ t+\lambda_{ij}(t)\leq T\}} x_{ij}(t) - \\ & \sum_{\{j\ :\ (j,i)\in A\}}\ \sum_{\{t':t'+\lambda_{ji}(t')=t\}} x_{ji}(t')\Bigg] + \\ & \sum_{t=0}^{T}\sum_{i\in C_T(t)-\{s\}}\Big[x_{ii}(t) - x_{ii}(t-1)\Big] \end{aligned} \tag{3.12}$$

Furthermore, by exploring the fact that node j may be in C_T or $\overline{C}_T$, we then obtain

$$\begin{aligned} V_{\sum^T}(\mathbf{x}) \;=\; & \sum_{t=0}^{T}\Bigg[\sum_{i\in C_T(t)}\sum_{j\in\overline{C}_T(t+\lambda_{ij}(t)),(i,j)\in A} x_{ij}(t) + \\ & \sum_{i\in C_T(t)}\sum_{j\in C_T(t+\lambda_{ij}(t)),(i,j)\in A} x_{ij}(t) - \\ & \sum_{i\in C_T(t)}\sum_{j\in C_T(t'),t'+\lambda_{ji}(t')=t,(j,i)\in A} x_{ji}(t') - \\ & \sum_{i\in C_T(t)}\sum_{j\in\overline{C}_T(t'),t'+\lambda_{ji}(t')=t,(j,i)\in A} x_{ji}(t')\Bigg] + \\ & \sum_{t=0}^{T-1}\Bigg[\sum_{i\in C_T(t)\cap C_T(t+1)-\{s\}} x_{ii}(t) + \sum_{i\in C_T(t)\cap\overline{C}_T(t+1)-\{s\}} x_{ii}(t)\Bigg] - \\ & \sum_{t=1}^{T}\Bigg[\sum_{i\in C_T(t)\cap C_T(t-1)-\{s\}} x_{ii}(t-1) + \sum_{i\in C_T(t)\cap\overline{C}_T(t-1)-\{s\}} x_{ii}(t-1)\Bigg] \end{aligned} \tag{3.13}$$

Equation (3.13) can be simplified by noting that whenever node k belongs to $C_T(t_1)$, l belongs to $C_T(t_1+\lambda_{kl}(t_1))$, $(k,l)\in A$, and $t_1+\lambda_{kl}(t_1)\le T$, the variable $x_{kl}(t_1)$ in the second term of the right-hand side of (3.13) for node $i=k$ and $t=t_1$, cancels the variable $-x_{kl}(t_1)$ in the third term for node $i=l$ and $t'=t_1$. Also, the variable $x_{ii}(t_1)$ in the fifth term of the right-hand side of (3.13) for $t=t_1$ with $0\le t_1<T$ cancels the variable $-x_{ii}(t)$ in the seventh term for $t=t_1+1$. We obtain then

$$\begin{aligned} V_{\sum^T}(\mathbf{x}) \;=\; & \sum_{t=0}^{T}\Bigg[\sum_{i\in C_T(t)}\sum_{j\in\overline{C}_T(t+\lambda_{ij}(t)),(i,j)\in A} x_{ij}(t) - \\ & \sum_{i\in C_T(t)}\sum_{j\in\overline{C}_T(t'),t'+\lambda_{ji}(t')=t,(j,i)\in A} x_{ji}(t')\Bigg] + \\ & \sum_{t=0}^{T-1}\sum_{i\in C_T(t)\cap\overline{C}_T(t+1)-\{s\}} x_{ii}(t) - \sum_{t=1}^{T}\sum_{i\in C_T(t)\cap\overline{C}_T(t-1)-\{s\}} x_{ii}(t-1) \\ = \; & \sum_{(i,j)\in A}\sum_{t\in\Gamma_{ij}^T} x_{ij}(t) - \sum_{t=0}^{T}\sum_{i\in C_T(t)}\sum_{j\in\overline{C}_T(t'),t'+\lambda_{ji}(t')=t,(j,i)\in A} x_{ji}(t') + \\ & \sum_{i\in N-\{s,d\}}\sum_{t\in\Gamma_{ii}^T} x_{ii}(t) - \sum_{t=1}^{T}\sum_{i\in C_T(t)\cap\overline{C}_T(t-1)-\{s\}} x_{ii}(t-1) \end{aligned} \tag{3.14}$$

The first expression on the right-hand side of (3.14) denotes the amount of flow from the nodes in C_T to nodes in $\overline{C}_T$, while the second expression denotes the amount of flow returning from the nodes in $\overline{C}_T$ to the nodes in C_T. The third expression denotes the amount

of flow waiting at some points in time where the nodes are crossing the cut from C_T to $\overline{C}_T$, and the fourth denotes the amount of flow waiting at some points in time where the nodes are crossing the cut in the reverse direction. Therefore, the right-hand side denotes the net movement and waiting flow across the cut. ■

Theorem 3.2 *The value $V_{\sum^T}(\mathbf{x})$ of every dynamic flow $\mathbf{x}$ is bounded above by the capacity of every dynamic cut $(C_T, \overline{C}_T)$.*

Proof :
Substituting $x_{ij}(t) \leq u_{ij}(t)$ in the first expression of (3.14), $x_{ji}(t') \geq 0$ into the second expression, $x_{ii}(t) \leq a_i(t)$ in the third expression, and $x_{ii}(t-1) \geq 0$ into the fourth expression shows that

$$V_{\sum^T}(\mathbf{x}) \leq \sum_{(i,j)\in A} \sum_{t\in\Gamma^T_{ij}} u_{ij}(t) + \sum_{i\in N-\{s,d\}} \sum_{t\in\Gamma^T_{ii}} a_i(t) = W_{\sum^T}(C_T) \qquad \blacksquare$$

Corollary 3.1 *The value of every maximum dynamic flow is less than or equal to the capacity of every dynamic cut in the network.*

As in the static network maximum flow problem, a dynamic cut has a close relation with the dual of a maximum dynamic flow problem. A dual formulation DTMDNFP$'$ of the maximum dynamic flow problem (see (3.2) - (3.5)) is obtained by assigning dual multipliers $\pi_i(t)$, $\rho_{ij}(t)$ and $\rho_{ii}(t)$ to the constraint (3.3) and the upper bound constraints of (3.4) and (3.5), respectively.

$$\begin{aligned}
(\text{DTMDFP}') \quad \min\ V'_{\sum^T}(\pi,\rho) &= \sum_{(i,j)\in A} \sum_{\{t\ :\ 0\leq t+\lambda_{ij}(t)\leq T\}} \rho_{ij}(t)u_{ij}(t) + \\
&\quad \sum_{i\in N-\{s,d\}} \sum_{t=0}^{T-1} \rho_{ii}(t)a_i(t) && (3.15)
\end{aligned}$$

Subject to

$$\begin{aligned}
\pi_i(t) - \pi_j(t+\lambda_{ij}(t)) + \rho_{ij}(t) &\geq 0,\ i,j \in N-\{s,d\},\ t+\lambda_{ij}(t) \leq T && (3.16)\\
-\pi_j(t+\lambda_{ij}(t)) + \rho_{ij}(t) &\geq 0,\ i=s, j\neq d,\ t+\lambda_{ij}(t)\leq T && (3.17)\\
\pi_i(t) + \rho_{ij}(t) &\geq 1,\ i\neq s, j=d,\ t+\lambda_{ij}(t)\leq T && (3.18)\\
\rho_{ij}(t) &\geq 1,\ i=s, j=d,\ t+\lambda_{ij}(t)\leq T && (3.19)\\
\pi_i(t) - \pi_i(t+1) + \rho_{ii}(t) &\geq 0,\ i\in N-\{s,d\},\ t=0,\ldots,T-1 && (3.20)\\
\pi_i(t) &\leqq 0,\ i\in N-\{s,d\},\ t=0,\ldots,T && (3.21)\\
\rho_{ij}(t) &\geq 0,\ (i,j)\in A,\ t+\lambda_{ij}(t)\leq T && (3.22)\\
\rho_{ii}(t) &\geq 0,\ i\in N-\{s,d\},\ t=0,\ldots,T && (3.23)
\end{aligned}$$

Without loss of generality, we define $\pi_s(t) := 0$ and $\pi_d(t) := 1$ for all $t \in \{0, \ldots, T\}$. Consequently, the dual constraints (3.16) - (3.20) are simplified into

$$\pi_i(t) - \pi_j(t + \lambda_{ij}(t)) + \rho_{ij}(t) \geq 0, \ (i,j) \in A, \ t + \lambda_{ij}(t) \leq T \tag{3.24}$$

$$\pi_i(t) - \pi_i(t+1) + \rho_{ii}(t) \geq 0, \ i \in N - \{s,d\}, \ t = 0, \ldots, T-1 \tag{3.25}$$

The interrelation between the solution of the dual problem and the dynamic cut is described by the following theorem.

Theorem 3.3 *Every $s-d$ dynamic cut $(C_T, \overline{C}_T)$ determines a feasible solution to the dual of the maximum dynamic flow problem as follows.*

$$\rho_{ij}(t) := \begin{cases} 1 & , i \in C_T(t), \ j \in \overline{C_T}(t + \lambda_{ij}(t)) \\ 0 & , \text{otherwise} \end{cases} \tag{3.26}$$

$$\pi_i(t) := \begin{cases} 0 & , i \in C_T(t) \\ 1 & , i \in \overline{C_T}(t) \end{cases} \tag{3.27}$$

$$\rho_i(t) := \begin{cases} 1 & , i \in C_T(t), \ i \in \overline{C_T}(t+1) \\ 0 & , \text{otherwise} \end{cases} \tag{3.28}$$

Proof:

- $\pi_s(t) = 0$ since $s(t) \in C_T, \ \forall t$; $\pi_d(t) = 1$ since $d(t) \in \overline{C_T}, \ \forall t$
  These values do not violate any constraints of the dual problem.

- For every $(i,j) \in A - \{(s,d)\}$, we need to consider $2^3 = 8$ cases as a combination of six possible states of node i and j that may be in C_T or $\overline{C_T}$ as follows.

1	2	3
$i \in C_T(t)$	$i \in \overline{C_T}(t)$	$i \in C_T(t+1)$

4	5	6
$i \in \overline{C_T}(t+1)$	$j \in C_T(t + \lambda_{ij}(t))$	$j \in \overline{C_T}(t + \lambda_{ij}(t))$

These 8 cases for every $(i,j) \in A - \{(s,d)\}$ are shown by a tree diagram in Figure 3.1. It is easily showed that none of these 8 cases violates any dual constraint, for example:
Case 1:

$$i \in C_T(t), \ i \in C_T(t+1), \ j \in C_T(t + \lambda_{ij}(t))$$
$$\rho_{ij}(t) = 0, \ \pi_i(t) = 0, \ \pi_i(t+1) = 0, \ \pi_j(t + \lambda_{ij}(t)) = 0, \ \rho_{ii}(t) = 0$$

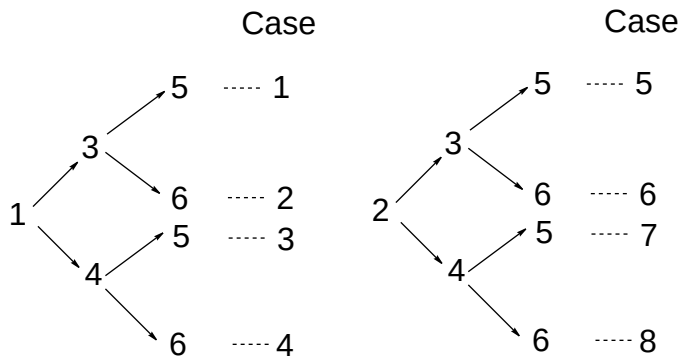


Figure 3.1: Eight possible cases of i and j that may be in C_T or $\overline{C_T}$ for every $(i,j) \in A - \{(s,d)\}$

Case 4:

$$i \in C_T(t),\ i \in \overline{C_T}(t+1),\ j \in \overline{C_T}(t+\lambda_{ij}(t))$$
$$\rho_{ij}(t) = 1,\ \pi_i(t) = 0,\ \pi_i(t+1) = 1,\ \pi_j(t+\lambda_{ij}(t)) = 1,\ \rho_{ii}(t) = 1,$$

These values satisfy all relevant dual constraints (3.16), (3.20) - (3.23). The proof can be also extended to prove the feasibility for arcs (s,i) and (i,d).

- By redefining ρ to

$$\rho_{ij}(t) := \begin{cases} 1 & ,t \in \Gamma_{ij}^T \\ 0 & ,\text{otherwise} \end{cases}$$

$$\rho_{ii}(t) := \begin{cases} 1 & ,t \in \Gamma_{ii}^T \\ 0 & ,\text{otherwise} \end{cases}$$

the cost of the dual feasible solution can be reformulized as

$$\begin{aligned} V'_{\sum^T}(\pi,\rho) &= \sum_{(i,j)\in A} \sum_{\{t\ :\ 0\le t+\lambda_{ij}(t)\le T\}} \rho_{ij}(t)u_{ij}(t) + \\ &\quad \sum_{i\in N-\{s,d\}} \sum_{t=0}^{T-1} \rho_{ii}(t)a_i(t) \\ &= \sum_{(i,j)\in A} \sum_{t\in\Gamma_{ij}^T} u_{ij}(t) + \sum_{i\in N} \sum_{t\in\Gamma_{ii}^T} a_i(t) \\ &= W_{\sum^T}(C_T) \end{aligned}$$

■

Consequently, Theorem 3.2 can be considered as the weak duality theorem for DTMDNFP. The strong duality theorem of DTMDNFP is known as the *maximum dynamic flow - minimum dynamic cut theorem.*

Theorem 3.4 (Maximum dynamic flow - minimum dynamic cut)
The value of maximum dynamic flow from a source node s to the sink node d equals the value of 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 value if and only if

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

The interrelation between a residual dynamic network and a dynamic cut is described by the following theorem.

Theorem 3.5 *For any dynamic flow $\mathbf{x}$ of value $V_{\sum^T}(\mathbf{x})$ in $G=(N,A,T)$, the additional flow that can be sent from the source s to the sink d is less than or equal to the residual capacity of any $s-d$ dynamic cut.*

Proof:
Suppose that $\mathbf{x}'$ is a dynamic flow having value $V_{\sum^T}(\mathbf{x}) + \triangle V$ with $\triangle V \geq 0$. By Theorem 3.2, $V_{\sum^T}(\mathbf{x}) + \triangle V \leq W_{\sum^T}(C_T)$, for any dynamic cut $(C_T, \overline{C}_T)$. This implies

$$\triangle V \ \leq \ W_{\sum^T}(C_T) - V_{\sum^T}(\mathbf{x})$$

By (3.14)

$$\begin{aligned}
\triangle V \ \leq \ & \sum_{(i,j)\in A} \sum_{t\in\Gamma_{ij}^T} (u_{ij}(t) - x_{ij}(t)) + \sum_{i\in N-\{s,d\}} \sum_{t\in\Gamma_{ii}^T} (a_i(t) - x_{ii}(t)) + \\
& \sum_{t=0}^{T} \sum_{i\in C_T(t)} \sum_{j\in\overline{C}_T(t'),t'+\lambda_{ji}(t')=t,(j,i)\in A} x_{ji}(t') + \sum_{t=1}^{T} \sum_{i\in C_T(t)\cap\overline{C}_T(t-1)-\{s\}} x_{ii}(t-1)
\end{aligned}$$

Since $G=(N,A,T)$ is assumed to be antisymmetric,

$$\sum_{t=0}^{T} \sum_{i\in C_T(t)} \sum_{j\in\overline{C}_T(t'),t'+\lambda_{ji}(t')=t,(j,i)\in A} x_{ji}(t') \ = \ \sum_{t=0}^{T} \sum_{i\in C_T(t)} \sum_{j\in\overline{C}_T(t'),t+\lambda_{ij}(t)=t'\leq T,(i,j)\in A_x^-} u_{ij}^x(t)$$

By extending the definition of Γ_{ij}^T in (3.9) for all $(i,j)\in A_x^+ \cup A_x^-$, we obtain

$$\sum_{t=0}^{T} \sum_{i\in C_T(t)} \sum_{j\in} \sum_{\overline{C}_T(t'),t'+\lambda_{ji}(t')=t,(j,i)\in A} x_{ji}(t') \ = \ \sum_{(i,j)\in A_x^-} \sum_{t\in\Gamma_{ij}^T} u_{ij}^x(t)$$

Since $\lambda_{ij}(t) = \lambda_{ij}^x(t)$ for any $(i,j) \in A_x^+$,

$$\sum_{(i,j)\in A} \sum_{t\in\Gamma_{ij}^T} (u_{ij}(t) - x_{ij}(t)) = \sum_{(i,j)\in A_x^+} \sum_{t\in\Gamma_{ij}^T} u_{ij}^x(t)$$

Since $\lambda_{ii}^x(t+1) = -1$ when $x_{ii}(t) > 0$, Remark 3.1 implies

$$\begin{aligned} \sum_{t=1}^{T} \sum_{i\in C_T(t)\cap\overline{C}_T(t-1)-\{s\}} x_{ii}(t-1) &= \sum_{t=0}^{T-1} \sum_{i\in C_T(t+1)\cap\overline{C}_T(t)-\{s\}} a_i^{x-}(t+1) \\ &= \sum_{i\in N-\{s,d\}} \sum_{t\in\Gamma_{ii}^T} a_i^{x-}(t) \end{aligned}$$

and

$$\sum_{i\in N-\{s,d\}} \sum_{t\in\Gamma_{ii}^T} (a_i(t) - x_{ii}(t)) = \sum_{i\in N-\{s,d\}} \sum_{t\in\Gamma_{ii}^T} a_i^{x+}(t)$$

Hence,

$$\triangle V \leq \sum_{(i,j)\in A_x} \sum_{t\in\Gamma_{ij}^T} u_{ij}^x(t) + \sum_{i\in N-\{s,d\}} \sum_{t\in\Gamma_{ii}^T} a_i^x(t) \qquad \blacksquare$$

3.3 Solution Algorithm for a Maximum Dynamic Network Flow Problem with Constant Attributes

Here constant attributes mean

$$\lambda_{ij}(t) := \lambda_{ij},\ u_{ij}(t) := u_{ij},\ (i,j) \in A,\ t \in \{0,\dots,T\}$$

and

$$a_i(t) := a_i,\ i \in N - \{s,d\},\ t \in \{0,\dots,T\}$$

For the purpose of the problem classification presented in Appendix D, we denote this problem by

$$(s,d)/(\lambda,u,a)/V_{\sum^T}$$

We will see later that either finite or infinite waiting assumptions do not influence the optimal flow distribution.

One idea of efficiently tackling DTMDNFP is to use information obtained by solving a related optimization problem in the associated (much smaller!) static network. Here, the

maximum dynamic flow is obtained by *repeating* feasible flows along some static paths (see Definition 3.2) from the source to the sink as long as possible starting from time zero within the time horizon T. This approach is called *temporally repeated flows* (see Definition 3.3).

We define $\mathbf{f} : A \to \mathbb{R}_0^+$ as a static (classical) flow from s to d in a static network $G_{stat} = (N, A)$ associated to a dynamic network $G = (N, A, T)$. The static network G_{stat} refers to the same graph as that of G. It is obtained by disregarding the time horizon T, node capacities, and considering the travel times as cost attribute.

Definition 3.2 (Path flow and path decomposition, see e.g., Hamacher and Klamroth [HK00])

- *A path flow $\gamma_P = (v(P), P)$ is a static flow of value $v(P)$ along the path P in the associated static network $G_{stat} = (N, A)$ of the dynamic network $G = (N, A, T)$, i.e.,*

$$\gamma_P(i,j) = \begin{cases} v(P) & , (i,j) \in P \\ 0 & , \text{otherwise} \end{cases}$$

- *Let $\mathbb{P} = \{P_1, P_2, \ldots, P_k\}$ be a set of paths with associated path flows $\gamma_{P_1}, \ldots, \gamma_{P_k}$ of values $v(P_1), \ldots, v(P_k)$. $\mathbb{P}$ is a path decomposition of the static flow $\mathbf{f}$ if $\mathbf{f} = \sum_{l=1}^{k} \gamma_{P_l}$.*

Recall the definition of $\lambda(P)$ given by (2.11).

Definition 3.3 (Temporally repeated flows, Ford and Fulkerson [FF58, FF62])

- *Let $\gamma_P = (v(P), P)$ be a path flow. The temporally repeated path flow γ_P^T is a dynamic flow obtained by repeating the path flow γ_P for $(T + 1 - \lambda(P))$ times, i.e., by sending $v(P)$ units of flow every time period from time zero to time $T - \lambda(P)$ along the same static path P.*

- *A temporally repeated flow (TRF) is a dynamic flow obtained by temporarily repeating all path flows of some path decomposition $\mathbb{P}$. The value of TRF is then denoted by $v(\mathbb{P}_T)$ and given by*

$$v(\mathbb{P}_T) = \sum_{P_l \in \mathbb{P}} v(P_l)(T + 1 - \lambda(P_l)) \tag{3.29}$$

Example 3.1

Figure 3.2 shows a network with six nodes and eight arcs. Node 1 and 6 are the source and the sink node, respectively. Travel time and capacity attributes are attached on each associated arc. We define the time horizon $T = 7$ time units. Let $\mathbb{P} = \{P_1, P_2, P_3\}$ where $\gamma_{P_1} = (v(P_1) = 1, P_1 = (1, 2, 6))$, $\gamma_{P_2} = (5, P_2 = (1, 2, 4, 3, 6))$, $\gamma_{P_3} = (1, P_3 = (1, 3, 6))$. The distance of each path with respect to the travel times are $\lambda(P_1) = 4, \lambda(P_2) = 7$, and

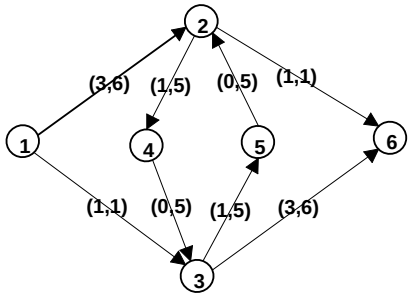


Figure 3.2: Network G for Example 3.1

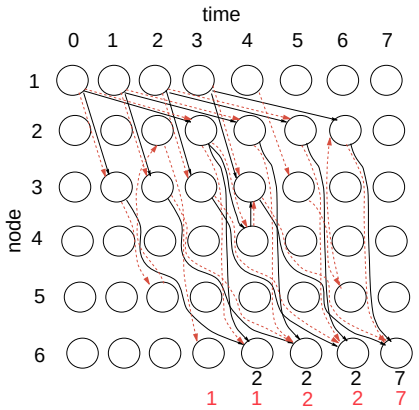


Figure 3.3: Discrete-time dynamic network flow for Example 3.1. The solid lines and dashed lines correspond to TRF and non-TRF, respectively

$\lambda(P_3) = 4$. The TRF obtained by temporarily repeating all path flows in $\mathbb{P}$ is represented by the solid lines in Figure 3.3. P_1 is repeated four times at $t = 0, 1, 2, 3$. P_2 is repeated only once at time $t = 0$, and P_3 is repeated four times at time $t = 0, 1, 2, 3$. As a comparison, a dynamic flow which is not a TRF is represented by dashed lines. □

Let $\mathbb{P} = \{P_1, P_2, \ldots, P_k\}$ be a path decomposition of $\mathbf{f}$. We denote by $\delta_{ij;P_l}$ the arc-path incidence value where

$$\delta_{ij;P_l} = \begin{cases} 1 & , (i,j) \in P_l \\ 0 & , \text{otherwise} \end{cases}, \ l = 1, \ldots, k \tag{3.30}$$

Obviously, we have

$$f_{ij} = \sum_{P_l \in \mathbb{P}} \delta_{ij;P_l} v(P_l), \ \forall (i,j) \in A \tag{3.31}$$

We denote by $\lambda_i(P_l)$ the distance with respect to the travel times from the source to node i along some $P_l \in \mathbb{P}$ as shown in Figure 3.4, i.e., if $P_l = \{s = j_1, j_2, \ldots, j_{|P_l|} = d\}$, then

$$\lambda_i(P_l) = \sum_{l'=1}^{j_{l'+1}=i} \lambda_{j_{l'} j_{l'+1}} \tag{3.32}$$

Suppose $P_l \in \mathbb{P}$ is repeated at time $t' \leq T - \lambda(P_l)$, we denote by $\zeta_{ij;P_l}^{t't}$ the dynamic arc-path incidence value.

$$\zeta_{ij;P_l}^{t't} = \begin{cases} 1 & , (i,j) \in P_l,\ t' + \lambda_i(P_l) = t \\ 0 & , \text{otherwise} \end{cases} ,\ t = 0, \ldots, T \tag{3.33}$$

In $\zeta_{ij;P_l}^{t't}$, t' denotes the departure time at node s and t denotes the arrival time at node i along P_l. Obviously $\sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} \leq 1$ for every $t \in \{0, \ldots, T\}$. The dynamic flow $x_{ij}(t)$ for every arc $(i,j) \in A$ and every time $t = 0, \ldots, T$ can be obtained by

$$x_{ij}(t) = \sum_{P_l \in \mathbb{P}} \sum_{t'=0}^{T-\lambda(P_l)} v(P_l) \zeta_{ij;P_l}^{t't} \tag{3.34}$$

For the incoming flow to some node $i \in N$, (3.34) implies

$$x_{ji}(t - \lambda_{ji}) = \sum_{P_l \in \mathbb{P}} \sum_{t'=0}^{T-\lambda(P_l)} v(P_l) \zeta_{ji;P_l}^{t'(t-\lambda_{ji})}$$

The interrelation between $\delta_{ij;P_l}$ and $\zeta_{ij;P_l}^{t't}$ can be explained as follows. By (3.30), $\zeta_{ij;P_l}^{t't}$ can be redefined as

$$\zeta_{ij;P_l}^{t't} = \begin{cases} 1 & , \delta_{ij;P_l} = 1,\ t' + \lambda_i(P_l) = t \\ 0 & , \text{otherwise} \end{cases} ,\ t = 0, \ldots, T \tag{3.35}$$

and

$$\begin{aligned} \zeta_{ji;P_l}^{t'(t-\lambda_{ji})} &= \begin{cases} 1 & , \delta_{ji;P_l} = 1,\ t' + \lambda_j(P_l) = t - \lambda_{ji} \\ 0 & , \text{otherwise} \end{cases} \\ &= \begin{cases} 1 & , \delta_{ji;P_l} = 1,\ t' + \lambda_i(P_l) = t \\ 0 & , \text{otherwise} \end{cases} ,\ t = 0, \ldots, T \end{aligned}$$

Moreover, (3.35) implies that $\sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} = 1$ for some $t \in \{0, \ldots, T\}$ if and only if $\delta_{ij;P_l} = 1$. This is equivalent to the condition that $\delta_{ij;P_l} = 0$ if and only if $\sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} = 0$ for all $t \in \{0, \ldots, T\}$. Therefore, we obtain

$$\sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} \leq \delta_{ij;P_l},\ t \in \{0, \ldots, T\} \tag{3.36}$$

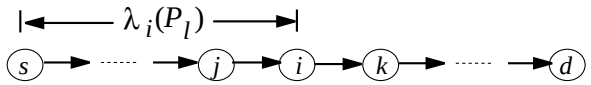


Figure 3.4: Travel time distance $\lambda_i(P_l)$ from the source node s to the node i along the path P_l

Since P_l is a simple path, we obtain

$$\sum_{(j,i)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ji;P_l}^{t'(t-\lambda_{ji})} \leq 1$$

for every $P_l \in \mathbb{P}$.

Furthermore, since P_l is a path from s to d, for some $j \in N$, $\sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ji;P_l}^{t'(t-\lambda_{ji})} = 1$, $\forall i \in N - \{s,d\}$, $\forall t \in \{0,\ldots,T\}$ if and only if $\exists j' \in N$ with $\sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij';P_l}^{t't} = 1$. Hence,

$$\sum_{(j,i)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ji;P_l}^{t'(t-\lambda_{ji})} = 1 \;\Leftrightarrow\; \sum_{(i,j)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} = 1,\ \forall i \in N - \{s,d\},\ \forall t \in \{0,\ldots,T\}$$

and

$$\sum_{(j,i)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ji;P_l}^{t'(t-\lambda_{ji})} = 0 \Leftrightarrow \sum_{(i,j)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} = 0$$

These imply

$$\sum_{(j,i)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ji;P_l}^{t'(t-\lambda_{ji})} - \sum_{(i,j)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} = 0 \tag{3.37}$$

Equation (3.37) represents a *dynamic path fomulation* of the dynamic network conservation flow constraint. This result will be used by Theorem 3.7 to prove that the TRF solves the maximum dynamic network flow problem with constant attributes (i.e., TRF solves $(s,d)/(\lambda,b,a)/V_{\sum^T}$). But first, we want to discuss how to find a static flow $\mathbf{f}$ in which its path decomposition can be used to obtain a maximum dynamic flow $\mathbf{x}$.

Let us denote by f_{ij} and f_{ds} the static flow on arc $(i,j) \in A$ and the *circulation flow* of the static network $G_{stat} = (N,A)$ from d to s, respectively. We define the cost c_{ij} per unit of flow in arc $(i,j) \in A \cup \{(d,s)\}$ as follows.

$$c_{ij} = \begin{cases} \lambda_{ij} & , (i,j) \in A \\ -(T+1) & , \text{ circulation arc } (d,s) \end{cases} \tag{3.38}$$

The circulation arc is defined to have infinite flow capacity. *The minimum cost circulation problem* (MCCP) on the static network is thus defined by

$$\text{(MCCP)} \quad \min \sum_{(i,j)\in A} \lambda_{ij} f_{ij} \quad - \quad (T+1) f_{ds} \tag{3.39}$$

Subject to

$$\sum_{(i,j)\in A\cup\{(d,s)\}} f_{ij} - \sum_{(j,i)\in A\cup\{(d,s)\}} f_{ji} \quad = \quad 0, \ i \in N \tag{3.40}$$

$$0 \leq f_{ij} \quad \leq \quad u_{ij}, \ (i,j) \in A \tag{3.41}$$

We denote this minimum cost static circulation flow problem by (see Appendix D)

$$(s,d)/(u,c)/\mathbf{c}_{\sum circ}$$

Figure 3.5 shows an example of the circulation network where the cost of each arc is defined by (3.38).

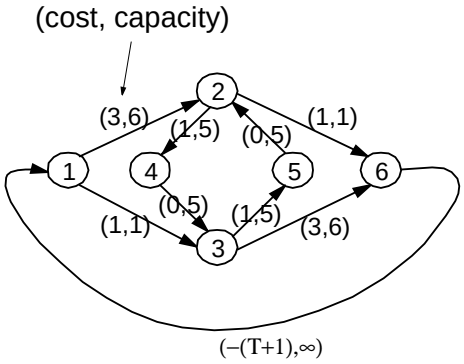


Figure 3.5: A circulation network with circulation arc $(6,1)$ and cost of each arc defined by (3.38)

The next theorem shows that TRF obtained from the path decomposition of MCCP in the static network, solves DTMDNFP with constant attributes.

Theorem 3.6 (Ford and Fulkerson [FF58, FF62]) *Finding a maximum temporally repeated flow in a dynamic network $G = (N, A, T)$ is equivalent to solving MCCP in the associated static network $G_{stat} = (N, A)$.*

Proof:
Let $\mathbf{f}$ be a feasible static flow in $G_{stat} = (N, A)$ satisfying Constraints (3.40) - (3.41) and let $\mathbb{P} = \{P_1, \ldots, P_k\}$ be its $s-d$ path decomposition. By using $\delta_{ij;P_l}$, the distance $\lambda(P_l)$ of path P_l can be determined as

$$\lambda(P_l) = \sum_{(i,j)\in A} \lambda_{ij} \delta_{ij;P_l}$$

Hence, (3.31) implies

$$\begin{aligned}\sum_{(i,j)\in A} \lambda_{ij} f_{ij} &= \sum_{P_l \in \mathbb{P}} v(P_l) \sum_{(i,j)\in A} \lambda_{ij} \delta_{ij;P_l} \\ &= \sum_{P_l \in \mathbb{P}} v(P_l)\lambda(P_l)\end{aligned}$$

The value of TRF in (3.29) implies

$$\begin{aligned}v(\mathbb{P}^T) &= \sum_{P_l \in \mathbb{P}} v(P_l)(T+1-\lambda(P_l)) \\ &= (T+1)\sum_{P_l \in \mathbb{P}} v(P_l) - \sum_{P_l \in \mathbb{P}} v(P_l)\lambda(P_l)\end{aligned}$$

Since $\sum_{P_l \in \mathbb{P}} v(P_l) = \sum_{(i,d)\in A} f_{id}$, we obtain

$$\begin{aligned}v(\mathbb{P}^T) &= (T+1)\sum_{(i,d)\in A} f_{id} - \sum_{(i,j)\in A} \lambda_{ij} f_{ij} \\ &= -(\sum_{(i,j)\in A} \lambda_{ij} f_{ij} - (T+1) f_{ds}\ ) \qquad (3.42)\end{aligned}$$

The negative of the right-hand side of (3.42) represents the objective function of MCCP. Therefore, the optimum solution of MCCP will generate the maximum TRF in $G = (N, A, T)$. ■

To prove that TRF solves the single source single sink DTMDNFP with constant attributes, we will use the maximum dynamic flow - minimum dynamic cut theorem (Theorem 3.4). Since the maximum dynamic flow can be obtained from the static flow, the corresponding dynamic cut can also be determined from the static network. Consider MCCP given by (3.39)-(3.41). This problem can be considered as the maximum TRF problem by writing the objective function in (3.39) as

$$-\sum_{(i,j)\in A} \lambda_{ij} f_{ij} + (T+1) f_{ds} \qquad (3.43)$$

If we assign the dual multipliers $\eta_i,\ \forall i \in N$ and $\rho_{ij},\ \forall (i,j) \in A$ to the constraints (3.40) and (3.41), respectively, then the dual formulation of the maximum TRF problem is given by

$$\begin{aligned}\min \quad & \sum_{(i,j)\in A} \rho_{ij} u_{ij} && (3.44)\\ -\eta_s + \eta_d &\geq T+1 && (3.45)\\ \eta_i - \eta_j + \rho_{ij} &\geq -\lambda_{ij},\ \forall (i,j) \in A && (3.46)\\ \rho_{ij} &\geq 0,\ \forall (i,j) \in A && (3.47)\end{aligned}$$

Without loss of generality, the values of η_s and η_d can be set to zero and $(T+1)$, respectively.

Lemma 3.2 (Ford and Fulkerson [FF58, FF62]) *The set valued function C_T and $\overline{C_T}$ with*

$$C_T(t) := \{i \ : \ \eta_i \leq t, \ i \in N\}, \ t \in \{0, 1, \ldots, T\} \tag{3.48}$$
$$\overline{C_T}(t) := N - C_T(t) \tag{3.49}$$

define a dynamic cut $(C_T, \overline{C_T})$ of G_T with

$$(C_T, \overline{C_T}) := \{(i(t), j(t + \lambda_{ij})) \in A_T \ : \ \eta_i \leq t < \eta_j - \lambda_{ij}\} \tag{3.50}$$

If (η, ρ) is an optimal solution of the dual problem defined by (3.44) - (3.47), then the capacity of the cut $(C_T, \overline{C_T})$ is equal to the value of temporally repeated flow.

Proof:
Since $\eta_s = 0$ and $\eta_d = T + 1$, by the definition of $C_T(t)$ we have that $s \in C_T(t)$ and $d \notin C_T(t)$ for every time $t \in \{0, \ldots, T\}$. Obviously, (3.50) defines an $s - d$ dynamic cut in the sense of (3.8). By (3.50), the set of times when the movement arc $(i(t), j(t+\lambda_{ij})) \in A_M$ crosses the dynamic cut is determined by the set

$$\Gamma_{ij}^T := \{t \ : \ \eta_i \leq t < \eta_j - \lambda_{ij}\} \tag{3.51}$$

Moreover the definition of $(C_T, \overline{C_T})$ in (3.50) implies that there is no holdover arc $(i(t), i(t+1)) \in A_H$ crossing the dynamic cut, i.e., $\Gamma_{ii}^T = \emptyset$. By using (3.11), the capacity $W_{\sum^T}(C_T)$ of $(C_T, \overline{C_T})$ is determined by

$$\begin{aligned} W_{\sum^T}(C_T) &= \sum_{(i,j)\in A} \sum_{t \in \Gamma_{ij}^T} u_{ij}(t) = \sum_{(i,j)\in A} \sum_{\eta_i}^{\eta_j - \lambda_{ij} - 1} u_{ij} \\ &= \sum_{(i,j)\in A} \max\{0, \eta_j - \eta_i - \lambda_{ij}\} u_{ij} \end{aligned} \tag{3.52}$$

If (η, ρ) is the optimal solution of the dual problem, then the complementary slackness theorem of the primal-dual optimality condition implies that for all $(i, j) \in A$

$$\eta_i - \eta_j + \rho_{ij} > -\lambda_{ij} \ \Rightarrow \ f_{ij} = 0 \tag{3.53}$$
$$\rho_{ij} > 0 \ \Rightarrow \ f_{ij} = u_{ij} \tag{3.54}$$

The dual constraints (3.46)-(3.47) imply that if $\eta_j - \eta_i - \lambda_{ij} > 0$, then $\rho_{ij} > 0$. Therefore, we can obtain the following condition.

$$\max\{0, \eta_j - \eta_i - \lambda_{ij}\}\big(u_{ij} - f_{ij}\big) = 0, \ \forall (i, j) \in A \tag{3.55}$$

Hence, using this equation, (3.52) is modified to

$$\begin{aligned} W_{\sum^T}(C_T) &= \sum_{(i,j)\in A} \max\{0, \eta_j - \eta_i - \lambda_{ij}\} f_{ij} \\ &= \sum_{(i,j)\in A} \sum_{t=\eta_i}^{\eta_j-\lambda_{ij}-1} f_{ij} = \sum_{(i,j)\in A} \sum_{t=\eta_i}^{\eta_j-1} f_{ij} - \sum_{(i,j)\in A} \sum_{t=\eta_j-\lambda_{ij}}^{\eta_j-1} f_{ij} \\ &= \sum_{(i,j)\in A} \big(\eta_j - \eta_i\big) f_{ij} - \sum_{(i,j)\in A} \lambda_{ij} f_{ij} \\ &= \eta_s\big(-\sum_{(s,i)\in A} f_{si}\big) + \sum_{i\in N-\{s,d\}} \eta_i\big(\sum_{(j,i)\in A} f_{ji} - \sum_{(i,j)\in A} f_{ij}\big) \\ &\quad +\eta_d\big(\sum_{(i,d)\in A} f_{id}\big) - \sum_{(i,j)\in A} \lambda_{ij} f_{ij} \end{aligned}$$

By the flow conservation constraint (3.40) and the assumptions that $\eta_s = 0$ and $\eta_d = T+1$, we obtain

$$\begin{aligned} W_{\sum^T}(C_T) &= (T+1)\big(\sum_{(i,d)\in A} f_{id}\big) - \sum_{(i,j)\in A} \lambda_{ij} f_{ij} \\ &= (T+1) f_{ds} - \sum_{(i,j)\in A} \lambda_{ij} f_{ij} \qquad \blacksquare \end{aligned}$$

Proposition 3.1 *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 every time $t' > t$.*

Proof:
The proof follows immediatelly from the definition of C_T given by (3.48). $\blacksquare$

Now, we are ready to prove that TRF solves $(s,d)/(\lambda,u,a)/V_{\sum^T}$.

Theorem 3.7 *A dynamic flow* $\mathbf{x}$ *defined as the temporally repeated flow by (3.34), solves $(s,d)/(\lambda,u,a)/V_{\sum^T}$. Moreover the dynamic cut $(C_T, \overline{C_T})$ defined by (3.50) is a minimum dynamic cut.*

Proof:
By (3.37), we obtain

$$\begin{aligned} \sum_{(j,i)\in A} x_{ji}(t-\lambda_{ji}) - \sum_{(i,j)\in A} x_{ij}(t) &= \sum_{P_l\in\mathbb{P}} v(P_l)\Big(\sum_{(j,i)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ji;P_l}^{t'(t-\lambda_{ji})} - \\ &\qquad \sum_{(i,j)\in A} \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't}\Big) \\ &= 0 \end{aligned}$$

By defining the holdover flow $x_{ii}(t) = 0, \forall i \in N;\ t = 0, \ldots, T$, the dynamic flow $\mathbf{x}$ satisfies the flow conservation constraint (3.3).

Furthermore, we have to prove that $0 \leq x_{ij}(t) \leq u_{ij},\ \forall (i,j) \in A;\ t = 0, \ldots, T$. Since $v(P_l) \geq 0$ for every $P_l \in \mathbb{P}$ and there is no negative coefficient in (3.34), $x_{ij}(t)$ must be non-negative. By (3.36), we obtain

$$\begin{aligned} x_{ij}(t) &= \sum_{P_l \in \mathbb{P}} v(P_l) \sum_{t'=0}^{T-\lambda(P_l)} \zeta_{ij;P_l}^{t't} \leq \sum_{P_l \in \mathbb{P}} v(P_l)\delta_{ij;P_l} = f_{ij} \\ &\leq u_{ij},\ \forall (i,j) \in A;\ t = 0, \ldots, T \end{aligned}$$

Hence, $\mathbf{x}$ is a feasible solution of the problem $(s,d)/(\lambda,u,a)/V_{\sum^T}$. Furthermore, by Theorem 3.6 and Lemma 3.2, $\mathbf{x}$ is indeed an optimal solution of $(s,d)/(\lambda,u,a)/V_{\sum^T}$ and $(C_T, \overline{C_T})$ defined by (3.50) is a minimum dynamic cut. ■

Hence, in order to solve the maximum dynamic flow problem with constant capacity, we only have to solve the minimum cost circulation problem in the small static network. Moreover, the proof shows that the maximum dynamic flow problem with constant capacities and travel times never requires hold-over at all nodes, i.e., $x_{ii}(t) = 0, \forall i \in N; \forall t = 0, \ldots, T$. Therefore, variables $x_{ii}(t)$ can be eliminated from the problem formulation.

The following theorem shows that every feasible circulation can be decomposed into path flows.

Theorem 3.8 (see e.g., Hamacher and Klamroth [HK00]) *If* $\mathbf{f}$ *is a feasible circulation in a static network* $G_{stat} = (N, A)$, *then there exists dicycles* $C_1, \ldots, C_k$ *with values* $v(C_1), \ldots, v(C_k)$ *such that*

(a) $\gamma_{C_l} = (v(C_l), C_l)$ *is a dicycle flow in* $G_{stat}, \forall l = 1, 2, \ldots, k$.

(b) $\mathbf{f} = \sum_{l=1}^{k} \gamma_{C_l}$

(c) $k \leq m$

Proof:
The assertion trivially holds if $\mathbf{f} = \mathbf{0}$. If $\mathbf{f} \neq \mathbf{0}$, we choose $(i_1, i_2) \in A$ with $f_{i_1 i_2} > 0$. Since $\mathbf{f}$ is the circulation in G, there exists an arc $(i_2, i_3) \in A$ with $f_{i_2 i_3} > 0$. Using the same argument, we iteratively obtain a (not simple) dipath $(i_1, \ldots, i_{k'}, \ldots, i_k)$ such that $f_{i_l i_{l+1}} > 0, l = 1, \ldots, k-1$ and $i_{k'} = i_k$ for $k > k' \geq 1$, where k is minimal with this property. Then $C := (i_{k'}, \ldots, i_k)$ is a dicycle with

$$v(C) := \min\{f_{ij} \ :\ (i,j) \in C\} > 0 \tag{3.56}$$

We set $\mathbf{f} := \mathbf{f} - \gamma_C$. Then $\mathbf{f}$ is again a feasible circulation in G. We can conclude that either $\mathbf{f} = \mathbf{0}$ or we iterate this procedure. By (3.56), there is at least one arc that attains

the value $f_{ij} = 0$ in each iteration. Therefore we need at most m iterations until $\mathbf{f} = 0$. ■

Corollary 3.2 *Every feasible flows in the network G_{stat} can be decomposed into*

(a) *Dicycle flows*

(b) *Path flows from the source to the sink*

By disregarding the circular arc (d, s), the decomposition of $\mathbf{f}$ yields $s - d$ path flows.

Theorem 3.6 - 3.7 and Corollary 3.2 trigger a simple algorithm, described in Algorithm 3.1, to solve the maximum dynamic flow problem with constant capacities and travel times.

Algorithm 3.1 : Solving $(s, d)/(\lambda, u, a)/V_{\sum^T}$ via TRF

INPUT	Dynamic network $G = (N, A, T)$, constant capacity functions u_{ij}, a_i, and travel time λ_{ij}.
OUTPUT	Maximum dynamic flow $x_{ij}(t)$.
0	Set the dynamic flow $x_{ij}(t) = 0,\ \forall (i,j) \in A;\ t = 0, \ldots, T$.
1	Apply the minimum cost circulation flow algorithm to the network G_{stat} with cost on each arc is defined by (3.38). Let the optimal solution be $\mathbf{f}^*$.
2	Decompose $\mathbf{f}^*$ into k path flows $P_1, P_2, \ldots, P_k$ such that $\mathbf{f}^* = \sum_{l=1}^{k} v(P_l)$.
3	$x_{ij}(t) = \sum_{l=1}^{k} \sum_{t'=0}^{T-\lambda(P_l)} v(P_l) \zeta_{ij;P_l}^{t't},\ (i,j) \in A,\ t = 0, \ldots, T$.

The decomposition in step 2 can be done by using the method contained in the proof of Theorem 3.8 which may run in $\mathcal{O}(mn)$ (see, for example Ahuja, Magnanti, and Orlin [AMO93]). The complexity of the above algorithm is thus dominated by the one of MCCP, that is, for example, $\mathcal{O}(m \log n(m + n \log n))$, due to Orlin [Orl88].

Example 3.2
Consider again the network in Figure 3.2. It is desired to calculate the maximum dynamic flow reaching the sink for $T = 7$ time units. The corresponding circulation network is shown in Figure 3.5. The optimal solution of MCCP is shown in Table 3.1 (only the positive flows).

Step 2 of Algorithm 3.1 gives the following path flows.

- $P_1 = \{1, 2, 6\}, \lambda(P_1) = 4, v(P_1) = 1$

Arc (i,j)	(1,2)	(1,3)	(2,4)	(2,6)	(3,6)	(4,3)	(6,1)
Flow f_{ij}	6	1	5	1	6	5	7

Table 3.1: Minimum cost circulation flow for static network in Example 3.2

- $P_2 = \{1, 2, 4, 3, 6\}, \lambda(P_2) = 7, v(P_2) = 5$
- $P_3 = \{1, 3, 6\}, \lambda(P_3) = 4, v(P_3) = 1$

Hence, P_1 must be repeated four times at time $t = 0, 1, 2$ and 3, P_2 must be repeated only once at time $t = 0$ and P_3 must be repeated four times at time $t = 0, 1, 2, 3$. The optimal discrete-time dynamic flows are shown by the solid lines in Figure 3.3. The value of dynamic flow equals 13 unit of flow, i.e., $V_{\sum^{T=7}} = 13$, in which 2 units of flow arrive at the sink at time $t = 4, 5, 6$ and 7 units at time $t = 7$.

The corresponding optimum dual solution of the static MCCP is
$\eta_1 = 0$; $\eta_2 = 3$; $\eta_3 = 4$; $\eta_4 = 4$; $\eta_5 = 3$; $\eta_6 = 8$
$\rho_{12} = 0$; $\rho_{13} = 3$; $\rho_{24} = 0$; $\rho_{26} = 4$; $\rho_{35} = 0$; $\rho_{36} = 1$; $\rho_{43} = 0$; $\rho_{52} = 0$
Using (3.48), the set-valued function $C_{T=7}$ is obtained as follows.

$$C_7(t) = \begin{cases} \{1\} & , t = 0, 1, 2 \\ \{1, 2, 5\} & , t = 3 \\ \{1, 2, 5, 3, 4\} & , t = 4, 5, 6, 7 \end{cases}$$

The set of times when the movement arc $(i(t), j(t + \lambda_{ij}))$ crossing the $s - d$ dynamic cut are
$\Gamma_{12}^{T=7} = \emptyset$; $\Gamma_{13}^{7} = \{0, 1, 2\}$; $\Gamma_{24}^{7} = \emptyset$; $\Gamma_{26}^{7} = \{3, 4, 5, 6\}$;
$\Gamma_{35}^{7} = \emptyset$; $\Gamma_{36}^{7} = \{4\}$; $\Gamma_{43}^{7} = \emptyset$; $\Gamma_{52}^{7} = \emptyset$
By using (3.11), the minimum value of dynamic cut is

$$W_{\sum^7}(C_7) = \sum_{(i,j)\in A} \sum_{t\in\Gamma_{ij}^7} u_{ij}(t) = 13$$

which is equal to the value of maximum dynamic flow. □

3.4 Solution Algorithm for a Maximum Dynamic Network Flow Problem with Time-Dependent Attributes

Consider the hybrid algorithm which combines the capacity scaling and the shortest augmenting path algorithms of the static network flow problem (see e.g., Ahuja, Magnanti, and Orlin [AMO93]). This algorithm solves the maximum static flow problem in $\mathcal{O}(nm \log U)$

with U the maximum capacity. Here we generalize this algorithm to take into account the time dependency of the network attributes. In this generalization, the associated residual dynamic network G_x of a dynamic network $G = (N, A, T)$ discussed in Section 2.4 will be put to use.

From the residual dynamic network G_x, we define a $\triangle$-residual dynamic network $G_x(\triangle)$ as a dynamic network containing only arcs and nodes whose residual capacity is at least $\triangle$. Note that $G_x(1) = G_x$ and $G_x(\triangle) \subset G_x$. Let us denote by U the biggest capacity over all $(i, j) \in A$ and over time $t \in \{0, \ldots, T\}$, i.e.,

$$U := \max_{(i,j)\in A} \max_{t\in\{0,\ldots,T\}} \{u_{ij}(t)\} \tag{3.57}$$

The capacity scaling procedure starts with $\triangle := 2^{\lfloor \log U \rfloor}$ and halves its value in every scaling phase until $\triangle = 1$. Consequently, there is $1 + \lfloor \log U \rfloor = \mathcal{O}(\log U)$ capacity scaling phases.

We define a dynamic augmenting path in $G_x(\triangle)$ as follows.

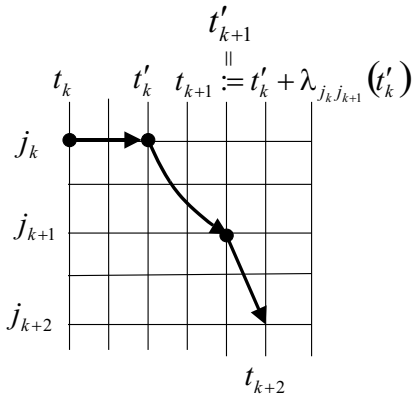


Figure 3.6: Dynamic augmenting path

Definition 3.4 (Dynamic augmenting path)

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

$$\begin{aligned} P_{sd}(t_1) \;=\; & \{s = j_1(t_1, t'_1), j_2(t_2, t'_2), \ldots, d = j_l(t_l, t'_l)\}, \\ & t_k, t'_k \in \{0, \ldots, T\},\ k = 1, \ldots, l \end{aligned} \tag{3.58}$$

where $t_{k+1} = t'_k + \lambda^x_{j_k j_{k+1}}(t'_k),\ k = 1, \ldots, l-1$ and define $t_l = t'_l$.
For every NTP $j_k(t_k, t'_k)$, t_k denotes the ready time at node j_k, and 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 $k = 2, \ldots, l$.

- *The length τ of $P_{sd}(t_1)$ with respect to the number of arcs in $P_{sd}(t_1)$ is defined by*

$$\tau(P_{sd}(t_1)) := l - 1 + \sum_{k=1}^{l-1} \mid t_k^{'} - t_k \mid \tag{3.59}$$

Here we consider one time unit of waiting as a one arc.

- *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 every waiting node in $P_{sd}(t_1)$ denoted by $\epsilon_u(P)$ and $\epsilon_a(P)$, respectively, i.e.,*

$$\epsilon_u(P) := \min_{1\le k\le l-1}\{u^x_{j_k j_{k+1}}(t_k^{'})\} \tag{3.60}$$

$$\epsilon_a(P) := \min_{1\le k\le l-1}\{\min_{t_k\le t^{''}\le t_k^{'}-1}\{a^{x+}_{j_k}(t^{''})\}, \min_{t_k^{'}+1\le t^{''}\le t_k}\{a^{x-}_{j_k}(t^{''})\}\} \tag{3.61}$$

$$\epsilon(P) := \min\{\epsilon_u(P), \epsilon_a(P)\} \tag{3.62}$$

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

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

By the definition of $G_x(\triangle)$, $\epsilon(P) \ge \triangle$. Figure 3.6 illustrates Definition 3.4. The new flow distribution $\mathbf{x}$ is computed as follows.

$$x_{ij}(\bar{t}) := \begin{cases} x_{ij}(\bar{t}) + \epsilon(P) & , \ i \ne j,\ i(t,\bar{t}),\ j(\bar{t} + \lambda^x_{ij}(\bar{t}), t^{''}) \in P \\ x_{ij}(\bar{t}) - \epsilon(P) & , \ i \ne j,\ j(t, \bar{t} + \lambda^x_{ij}(\bar{t})),\ i(\bar{t}, t^{''}) \in P \\ x_{ii}(\bar{t}) + \epsilon(P) & , \ i = j,\ t \le \bar{t} < t^{'},\ i(t,t^{'}) \in P \\ x_{ii}(\bar{t}) - \epsilon(P) & , \ i = j,\ t^{'} < \bar{t} \le t,\ i(t,t^{'}) \in P \\ x_{ij}(\bar{t}) & , \ \text{otherwise} \end{cases},$$
$$\forall\ (i,j) \in A \cup \{(i,i) : i \in N - \{s,d\}\},\ \bar{t} \in \{0,1,\ldots,T\} \tag{3.63}$$

We denote $\mathbf{x} = \mathbf{x} \pm \epsilon(P)$ as the dynamic flow given by (3.63).

The shortest dynamic augmenting path procedure always augment a flow along an $s-d$ dynamic path having the fewest number of arcs in $G_x(\triangle)$. We denote by $\phi_i(t)$ the distance from node i departing at time $t \in \{0,\ldots,T\}$ to the sink node d.

Definition 3.5 *We say that the distance function ϕ is* valid *in $G_x(\triangle)$ if it satisfies*

$$\begin{aligned} \phi_d(t) &= 0,\ t = 0, \ldots, T && (3.64)\\ \phi_i(t') &\leq \phi_j(t) + 1,\ \forall (i,j) \in A_x,\ t, t' = 0, \ldots, T \text{ with } t' + \lambda_{ij}^x(t') = t,\ u_{ij}^x(t') \geq \triangle && (3.65) \end{aligned}$$

$$\begin{aligned} \phi_i(t) &\leq \phi_i(t+1) + 1,\ \forall i \in N - \{s,d\},\ t = 0, \ldots, T-1 \text{ with } a_i^{x+}(t) \geq \triangle && (3.66)\\ \phi_i(t) &\leq \phi_i(t-1) + 1,\ \forall i \in N - \{s,d\},\ t = 1, \ldots, T \text{ with } a_i^{x-}(t) \geq \triangle && (3.67) \end{aligned}$$

Conditions (3.64) - (3.67) are referred to as *validity conditions*.

Proposition 3.2 *If ϕ is valid in $G_x(\triangle)$, then $\phi_i(t)$ is a lower bound on the length (with respect to the number of arcs) of the shortest $i-d$ dynamic augmenting path in $G_x(\triangle)$.*

Proof:
Let $P_{id}(t_1)$, as defined in (3.58), be any $i-d$ dynamic path with $i_1 = i$ and $t_1 = t$. The validity conditions imply that

$$\begin{aligned} \phi_{i_{l-1}}(t'_{l-1}) &\leq \phi_{i_l}(t_l) + 1 = \phi_{i_d}(t_l) + 1 = 1\\ \phi_{i_{l-2}}(t'_{l-2}) &\leq \phi_{i_{l-1}}(t_{l-1}) + 1 = \phi_{i_{l-1}}(t'_{l-1}) + \mid t'_{l-1} - t_{l-1} \mid + 1\\ &\leq 2 + \mid t'_{l-1} - t_{l-1} \mid\\ \phi_{i_{l-3}}(t'_{l-3}) &\leq \phi_{i_{l-2}}(t_{l-2}) + 1 = \phi_{i_{l-2}}(t'_{l-2}) + \mid t'_{l-2} - t_{l-2} \mid + 1\\ &\leq 3 + \sum_{k=l-2}^{l-1} \mid t'_k - t_k \mid\\ &\vdots\\ \phi_i(t'_1) = \phi_{i_1}(t'_1) &\leq l - 1 + \sum_{k=2}^{l-1} \mid t'_k - t_k \mid\\ \phi_i(t) = \phi_i(t_1) &\leq l - 1 + \sum_{k=1}^{l-1} \mid t'_k - t_k \mid \end{aligned}$$

■

Proposition 3.3 *If $\phi_s(t) \geq n(T+1)$, the residual dynamic network contains no $s-d$ dynamic augmenting path departing from node s at time t and arriving at d within the time horizon T.*

Proof:
Proposition 3.2 implies that

$$\phi_s(t) \leq n - 1 + (n-2)T < n(T+1)$$

Therefore, there will be no any $s-d$ dynamic augmenting path departing from node s at time t and arriving at d within the time horizon T having $\phi_s(t) \geq n(T+1)$. ■

Corollary 3.3 *Each flow augmentation requires $\mathcal{O}(nT)$ time.*

To describe the procedure to find the shortest dynamic augmenting path (SDAP) in $G_x(\triangle)$, we need the notion of *admissible dynamic path.*

Definition 3.6 (Admissible arc) *An arc (i,j) is called admissible at time $t \in \{0,\ldots,T\}$ in the residual dynamic network $G_x(\triangle)$ if it satisfies*

$$\phi_i(t) = \phi_j(t') + 1 \text{ with } u_{ij}^x(t) \geq \triangle$$

for some $t' = t + \lambda_{ij}^x(t)$ with $t' \in \{0,\ldots,T\}$. Moreover, waiting at node i for $t'-t$ time units starting at time t is called admissible in $G_x(\triangle)$ if it satisfies

$$\phi_i(t) = \phi_i(t') + t' - t \text{ with } a_i^{x+}(t'') \geq \triangle,\ t \leq t'' \leq t' - 1$$

Waiting canceling at node i for $t - t'$ time units starting at time t is called admissible in $G_x(\triangle)$ if it satisfies

$$\phi_i(t) = \phi_i(t') + t - t' \text{ with } a_i^{x-}(t'') \geq \triangle,\ t' + 1 \leq t'' \leq t$$

We call a dynamic augmenting path $P_{sd}(t_1)$ admissible if each arc and the waiting in the path are admissible. The shortest dynamic augmenting path procedure proceeds by augmenting flows along the admissible dynamic path from the source node to the sink node. The procedure maintains an admissible dynamic path $P_{si}(t_s)$ that arrives at node i at time t_i. To complete this path, we define $t_i' := t_i$ and set node i as the *current node.* If there is an arc (i,j) admissible at time t_i', then we perform the *advance* step by joining this arc to $P_{si}(t_s)$. If no admissible arc (i,j) at time t_i is found but a waiting or a waiting canceling at node i for one time unit is admissible, then we define $t_i' = t+1$ or $t_i' = t-1$, respectively. Otherwise we perform the *relabel* step by updating the label of current node i at time t_i'. This is done as follows. Define the temporary set $\Upsilon_i(t')$ as

$$\begin{aligned}\Upsilon_i(t') \;:=\; & \{\phi_j(t)+1 \;:\; u_{ij}^x(t') \geq \triangle,\ t = t' + \lambda_{ij}^x(t') \leq T\} \cup \\ & \{\phi_i(t'+1)+1 \;:\; a_i^{x+}(t') \geq \triangle, t'+1 \leq T\} \cup \\ & \{\phi_i(t'-1)+1 \;:\; a_i^{x-}(t') \geq \triangle, t'-1 \geq 0\}\end{aligned} \tag{3.68}$$

The value of $\phi_i(t')$ is updated by

$$\phi_i(t') := \min\{\varphi \;:\; \varphi \in \Upsilon_i(t')\} \tag{3.69}$$

If the set $\Upsilon_i(t')$ is empty, then $\phi_i(t')$ is set to infinity. After relabeling, we backtrack by one arc. If $t_i' \neq t_i$, then we reduce t_i' by one (if $t_i' > t_i$) or increase t_i' by one (if $t_i' < t_i$) and repeat the process of checking the admissible arc from node i at the new time t_i'. Otherwise, if $t_i' = t_i$ and $i \neq s$, then we delete node i at time t_i from $P_{si}(t_s)$ and repeat the process of checking the admissible arc from the last node in $P_{si}(t_s)$. We repeat these steps until

the dynamic path reaches the sink node d, at which time we perform flow augmentation. If node i is the source node and the new value of $\phi_i(t')$ is equal to infinity, then Proposition 3.3 implies that there is no augmenting path from the source node at time t in $G_x(\triangle)$.

The pseudocode of shortest dynamic augmenting path with capacity scaling is given in Algorithm 3.2. The initial distance label ϕ can be computed by performing the *backward breadth-first search* (see e.g., Ahuja, Magnanti, and Orlin [AMO93]) in the residual network starting at the sink node at every time $t \in \{0, \ldots, T\}$. This can be done in $\mathcal{O}((n+m)T)$.

Proposition 3.4 *For every $G_x(\triangle)$, Algorithm 3.2 maintains valid distance labels at each step. Moreover, each relabel operation strictly increases the distance label of a node-time pair.*

Proof :
By performing backward breadth-first search algorithm, we obtain initial valid distance labels ϕ for every $G_x(\triangle)$. Assume, inductively, that ϕ is valid prior to an operation. The advance operation in step 3 does not change any residual capacity or ϕ.

The relabel operation in step 4 modifies $\phi_i(t_i')$. This happens only if there is no

- arc $(i,j) \in A_x$ satisfies $u_{ij}^x(t_i') \geq \triangle$ with $t' + \lambda_{ij}^x(t_i') \leq T$ and $\phi_i(t_i') = \phi_j(t_i' + \lambda_{ij}^x(t_i')) + 1$,
- possibility to wait for one unit time, i.e., $a_i^{x+}(t_i') \geq \triangle$ with $t_i' + 1 \leq T$ and $\phi_i(t_i') = \phi_i(t_i' + 1) + 1$, and
- possibility to cancel the waiting, i.e., $a_i^{x-}(t_i') \geq \triangle$ with $t_i' - 1 \geq 0$ and $\phi_i(t_i') = \phi_i(t_i' - 1) + 1$.

The validity conditions (3.64) - (3.67) imply that

- $\phi_i(t_i') < \phi_j(t_i' + \lambda_{ij}^x(t_i')) + 1$ for every arc $(i,j) \in A_x$ satisfies $u_{ij}^x(t_i') \geq \triangle$ with $t' + \lambda_{ij}^x(t_i') \leq T$,
- $\phi_i(t_i') < \phi_i(t_i' + 1) + 1$ if $a_i^{x+}(t_i') \geq \triangle$ with $t_i' + 1 \leq T$, and
- $\phi_i(t_i') < \phi_i(t_i' - 1) + 1$ if $a_i^{x-}(t_i') \geq \triangle$ with $t_i' - 1 \geq 0$.

Therefore, if $\Upsilon_i(t_i')$ defined by (3.68) is not empty, then

$$\phi_i(t_i') < \min\{\varphi \ : \ \varphi \in \Upsilon_i(t')\} =: \phi_i'(t_i')$$

with $\phi_i'(t_i')$ the new distance label after the relabel operation. Hence, the relabel operation preserves the validity conditions for all arcs emanating from i or for the possibility to wait or cancel the waiting at time t_i'. For every incoming arc (k,i) with $t_k' + \lambda_{ki}(t_k') = t_i'$, by the induction hypothesis, it satisfies the validity conditions. Since $\phi_i(t_i') < \phi_i'(t_i')$, the relabel operation preserves the validity conditions for (k,i) departing at time t_k' and strictly increases the value of $\phi_i(t_i')$.

Algorithm 3.2 : Solving $(s,d)/(\lambda(t),u(t),a(t))/V_{\sum^T}$

INPUT Dynamic network $G=(N,A,T)$ with $\lambda(t),u(t),a(t))$.
OUTPUT Maximum dynamic flow $\mathbf{x}$.

0 $\mathbf{x} := 0$ and $\triangle := 2^{\lfloor \log U \rfloor}$
1 Initialize the distance label ϕ and set $t=0$.
2 Define $P := \{s(t_s, t_s^{'})\}$ with $t_s := t$ and $t_s^{'} := t, \quad i := s$
3 *Advance* node-time pair $i(t_i, t_i^{'})$.
While $\phi_s(t) < n(T+1)$
{ If $\exists j$ with $(i,j) \in A_x$, $u_{ij}^x(t_i^{'}) \geq \triangle$,
$t^{'} + \lambda_{ij}^x(t_i^{'}) \leq T$, and $\phi_i(t_i^{'}) = \phi_i(t_i^{'} + \lambda_{ij}^x(t_i^{'})) + 1$
{ $t_j := t_i^{'} + \lambda_{ij}^x(t_i^{'}), \quad t_j^{'} := t_j, \quad P := P \cup \{j(t_j, t_j^{'})\}, \quad i := j$ }
Else if $t_i^{'} + 1 \leq T, \quad a_i^{x+}(t_i^{'}) \geq \triangle,$ and $\phi_i(t_i^{'}) = \phi_i(t_i^{'}+1) + 1$
{ $t_i^{'}++$ }
Else if $t_i^{'} - 1 \geq 0, \quad a_i^{x-}(t_i^{'}) \geq \triangle,$ and $\phi_i(t_i^{'}) = \phi_i(t_i^{'}-1) + 1$
{ $t_i^{'}--$ }
Else { go to step 4 } } go to step 5
4 *Relabel* node-time pair $i(t_i, t_i^{'})$.
Determine $\Upsilon_i(t_i^{'})$ as defined in (3.68)
If $\Upsilon_i(t_i^{'}) \neq \emptyset$ { relabel $\phi_i(t_i^{'})$ as defined by (3.69) }
Else { set $\phi_i(t_i^{'}) = \infty$ }
if $t_i^{'} > t_i$ $\{t_i^{'}--\}$
else if $t_i^{'} < t_i$ $\{t_i^{'}++\}$
else { delete $i(t_i, t_i^{'})$ from P,
If $i \neq s$ { defines the last node in P as i }
Else { $\phi_s(t) := \infty$ }
}
Return to step 3
5 If d is reachable through P
{ *augment* flow along P, update $G_x(\triangle)$, go to step 2 }
Else if $t < T$ { $t++$ and go to step 2 }
Else if $\triangle > 1$ { $\triangle := \triangle/2$, go to step 1 }
Else { $\mathbf{x}$ is a maximum dynamic flow }

The flow augmentation in step 5 might remove some arcs in the dynamic augmenting path P from $G_x(\triangle)$. This modification to $G_x(\triangle)$ does not affect the validity conditions of ϕ. The flow augmentation might also create some additional backward arcs or waiting cancelings. By the admissibility property of augmenting path (see Definition 3.6), the va-

lidity conditions of these additional arcs or waiting cancelings are also preserved. ■
Before we continue discussing the number of augmentation process in Algorithm 3.2, we define the dynamic cut $(C_T, \overline{C_T})$ in $G_x(\triangle)$ as follows.

$$C_T(t) := \{i : \phi_i(t) \geq n(T+1),\ i \in N\} \tag{3.70}$$

and

$$\overline{C_T}(t) := \{i : \phi_i(t) < n(T+1),\ i \in N\} \tag{3.71}$$

By this definition, if there is no dynamic augmenting path from the source node to the sink node in $G_x(\triangle)$, nodes s and d are in $C_T(t)$ and $\overline{C_T}(t)$, respectively.

Proposition 3.5 *Algorithm 3.2 has $\mathcal{O}((m+n)T\log U)$ augmentations.*

Proof :
Let's consider the dynamic flow $\mathbf{x}'$ at the end of $\triangle$-scaling phase, having value V'. Furthermore, we denote by $(C'_T, \overline{C'_T})$ the corresponding dynamic cut in $G_x(\triangle)$. By definition, the residual capacity of every arc and every waiting in $(C'_T, \overline{C'_T})$ is strictly less than $\triangle$. Therefore, the capacity of this cut is at most $(m+n)\triangle T$. Theorem 3.5 implies $V_{\sum^T} - V' \leq (m+n)\triangle T$. In the next scaling phase, each augmentation carries at least $\triangle/2$ units of flow, so this scaling phase can perform at most $2(m+n)T$ such augmentations. Since there are $\mathcal{O}(\log U)$ capacity scaling phases, Algorithm 3.2 has at most $\mathcal{O}((m+n)T\log U)$ augmentations. ■

Proposition 3.6 *The total number of augmentations in Algorithm 3.2*

$$\mathcal{O}((m+n)nT^2\log U)$$

Proof :
Corollary 3.3 and Proposition 3.5 prove the proposition. ■

Proposition 3.7 *In Algorithm 3.2 each distance label increases at most $n(T+1)$ times. Consequently, the total number of relabel operations is at most $n^2(T+1)^2$.*

Proof :
Each relabel operation at node i at time t'_i increases the value of $\phi_i(t'_i)$ by at least one unit. After the algorithm has relabeled node i at time t'_i at most $n(T+1)$ times, $\phi_i(t'_i) \geq n(T+1)$. This node-time pair will never been selected again during an advance operation (i.e., step 3) since every node-time pair in the partial augmenting path have label values less than $n(T+1)$. Therefore, Algorithm 3.2 relabels every node at any time at most $n(T+1)$ times and the total number of relabel operations is bounded by $n^2(T+1)^2$. ■

To prove that Algorithm 3.2 gives a maximum dynamic flow, we use the maximum dynamic flow-minimum dynamic cut theorem.

Proposition 3.8 *When Algorithm 3.2 terminates, i.e.,* $\triangle < 1$*, the set of time-expanded arcs* $(C_T, \overline{C_T})$ *as defined by* (3.70)-(3.71) *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 $i \in C_T(t)$ for $t+\lambda_{ij}(t) \leq T$ and $(i,j) \in A$, the residual movement capacity $u^x_{ij}(t) = 0$ for each $(i,j) \in (C_T(t), \overline{C_T}(t+\lambda_{ij}(t)))$. Therefore $x_{ij}(t) = u_{ij}(t)$ for every arc $(i,j) \in (C_T(t), \overline{C_T}(t+\lambda_{ij}(t)))$. Moreover by the definition of a residual dynamic network, $u^x_{ji}(t+\lambda_{ij}(t)) = x_{ij}(t)$, which implies $x_{ji}(t+\lambda_{ij}(t)) = 0$ 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 (2.18), $u^x_{ji}(t+\lambda_{ij}(t)) > 0$. 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 $a^{x+}_i(t) = 0$. Let us denote by $x^-_{ii}(t+1)$ the number of waiting canceling units at node i at time t. Since $a^{x+}_i(t) = a_i(t) - x_{ii}(t)$, $x_{ii}(t) = a_i(t)$ for every node $i \in C_T(t) \cap \overline{C_T}(t+1)$. Moreover, by the definition of a residual dynamic network, $a^{x-}_i(t+1) = x_{ii}(t)$, which implies $x^-_{ii}(t+1) = 0$ 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 $a^{x-}_i(t+1) = 0$ for such an arc. If $a^{x-}_i(t+1) > 0$, 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 $a^{x-}_i(t+1) = 0$, implying $x_{ii}(t) = 0$ for every holdover arc (i,i) crossing from $\overline{C_T}(t)$ to $C_T(t+1)$.

Furthermore, Theorem 3.4 implies that $\mathbf{x}$ is a maximum dynamic flow and the corresponding dynamic cut $(C_T, \overline{C_T})$ is a minimum dynamic cut. ■

Proposition 3.9 *Algorithm 3.2 correctly computes a maximum dynamic flow for the time horizon* T *in* $\mathcal{O}((m+n)nT^2 \log U)$ *time.*

Proof :
Algorithm 3.2 terminates when $\triangle < 1$ and $\phi_s(t) \geq n(T+1)$ for every $t \in \{0, \ldots, T\}$. Propositions 3.3 and 3.8 imply that the dynamic flow obtained at the end of the algorithm is a maximum dynamic flow for the time horizon T. The computational complexity comes from Propositions 3.6 and 3.7.

The following Example 3.3 illustrates the implementation of Algorithm 3.2.

Example 3.3
Figure 3.7 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 3.2, while the time-dependent waiting capacities are given in Table 3.3. We define the time horizon T equals 7 time units. Initially, we set the flow variables $x_{ij}(t) = 0,\ \forall (i,j) \in$

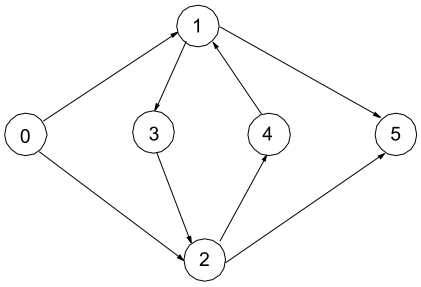


Figure 3.7: A network for Example 3.3

(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 3.2: Time-dependent travel times and capacities for the network in Example 3.3

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

Table 3.3: Time-dependent waiting capacities for the network in Example 3.3

$A;\ \forall t \le T$. We obtain $U = 6$ and $\triangle := 4$. It is found that there is no augmenting path in $G_x(4)$. The process is continued with $\triangle = 2$. The initial label for every node $i \in N$ and for every $t \in \{0, \ldots, T\}$ in the residual network $G_x(2)$ is shown in Table 3.4. The algorithm

t / $i \in N$	$\phi_i(t)$							
	0	1	2	3	4	5	6	7
0	2	2	∞	∞	∞	∞	∞	∞
1	1	1	1	1	∞	∞	∞	∞
2	1	1	1	1	1	∞	∞	∞
3	2	2	2	2	2	∞	∞	∞
4	2	2	2	2	∞	∞	∞	∞
5	0	0	0	0	0	0	0	0

Table 3.4: Initial label for the residual network $G_x(2)$ in Example 3.3

finds the following shortest dynamic augmenting path

$$P_1 = \{0(0,0), 2(2,2), 5(5,5)\} \text{ with } \epsilon(P_1) = 2$$

The dynamic flow $\mathbf{x}$ and the residual dynamic network $G_x(2)$ are then updated. The node labels are updated during the relabel operation in step 4 of the algorithm. The new node labels are shown in Table 3.5. These labes show that no augmenting path can be found

t / $i \in N$	$\phi_i(t)$							
	0	1	2	3	4	5	6	7
0	∞	2	∞	∞	∞	∞	∞	∞
1	1	1	1	1	∞	∞	∞	∞
2	1	1	1	1	1	∞	∞	∞
3	2	2	2	2	2	∞	∞	∞
4	2	2	2	2	∞	∞	∞	∞
5	0	0	0	0	0	0	0	0

Table 3.5: The node labels after the first shortest dynamic augmenting path in $G_x(2)$ is found

from the source node 0 at ready time $t = 0$. The process is then continued for $t = 1$. The algorithm finds the second augmenting path

$$P_2 = \{0(1,1), 2(3,3), 5(6,6)\} \text{ with } \epsilon(P_1) = 2$$

The dynamic flow $\mathbf{x}$ and the residual dynamic network $G_x(2)$ are then again updated. Since no more augmenting path exists in $G_x(2)$, the process is continued with $G_x(1)$. Two additional shortest augmenting paths are found.
$P_3 = \{0(0,0), 1(4,4), 5(7,7)\}$ with $\epsilon(P_3) = 1$ and
$P_4 = \{0(0,0), 1(4,5), 5(6,6)\}$ with $\epsilon(P_4) = 1$.
The maximum dynamic flow is shown in Table 3.6. The value of maximum dynamic flow

$x_{ij}(t)$	time t							
	0	1	2	3	4	5	6	7
$x_{01}(t)$	2	0	0	0	0	0	0	0
$x_{02}(t)$	2	2	0	0	0	0	0	0
$x_{13}(t)$	0	0	0	0	0	0	0	0
$x_{15}(t)$	0	0	0	0	1	1	0	0
$x_{24}(t)$	0	0	0	0	0	0	0	0
$x_{25}(t)$	0	0	2	2	0	0	0	0
$x_{32}(t)$	0	0	0	0	0	0	0	0
$x_{41}(t)$	0	0	0	0	0	0	0	0
Value of the flow arriving at the sink						2	3	1

Table 3.6: Maximum dynamic flow of Example 3.3.

is

$$V_{\sum^{T=7}} = 6 \qquad \square$$

3.5 Computational Results

In computational testing, we rely on the representative operation counts introduced by Ahuja, Magnanti, and Orlin in [AMO93] (see Appendix C)

- to identify the asymptotic bottleneck operations and
- to estimate the running time for different problems sizes

of Algorithm 3.2. To reach these goals, a series of experiments is made based on randomly generated dynamic networks. For these experiments, Algorithm 3.2 is implemented in C++ and run on a PC Pentium III, 500 MHz, and RAM of 256 MB.

To conduct the experiments, a dynamic random network generator is developed by extending the idea of NETGEN (Klingman, Napier, and Stutz [KNS74]) to include the time-dependent attributes (see Appendix B). The generated networks are connected (weakly). Eleven parameters must be specified in order to generate the network topology, arc travel times and capacities, and node waiting capacities. These parameters are

- random seed,
- time horizon T,
- number of nodes,
- indegree and outdegree of each node,

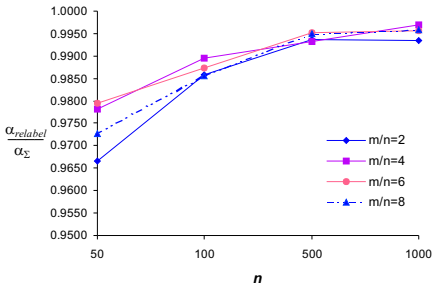
- minimum and maximum value of travel times (must be nonnegative),
- minimum and maximum value of arc capacities (must be nonnegative), and
- maximum value of waiting capacities (must be nonnegative).

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 have zero indegree and zero outdegree, respectively. This degree setting implies that the generated networks have $2n$, $4n$, $6n$, and $8n$ arcs. We denote by δ the density of the network, that is $\delta = m/n$. The minimum and maximum travel time is set to 1 and 10, respectively, and the minimum and maximum capacitiy is set to 25 and 50, respectively. The maximum waiting capacity is set to 10. For each specific setting of n and m, we test five random dynamic networks. Therefore, the total number of observations is 80.

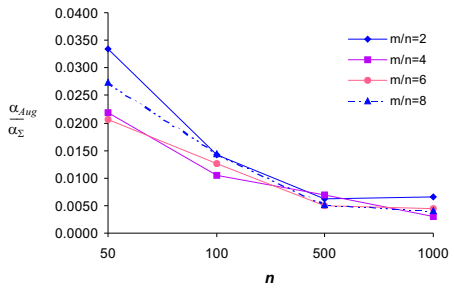
Here we identify the following set of representative operations:

(a) relabeling process and

(b) flow augmentation process



(a) A plot of the ratio of relabel node-time-pairs in relabeling operation to the total number of operations



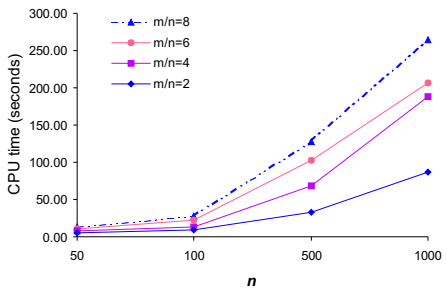
(b) A plot of the ratio of flow augmentation to the total number of operations

Figure 3.8: Identifying asymptotic bottleneck operations in Algorithm 3.2

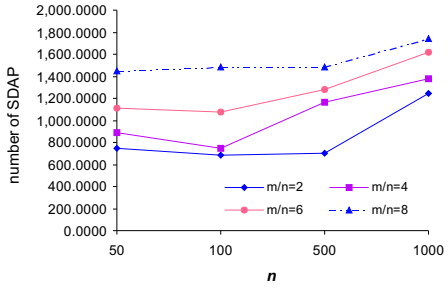
We denote by $\alpha_{relabel}(I)$ the number of node-time-pairs scanned in the representative operation (a), summed over all current nodes and all existing augmenting paths and denote by $\alpha_{aug}(I)$ the number of flow augmentation in the representative operation (b), summed over all existing augmenting paths, of a problem instance I. The value of $\alpha_{relabel}$ and α_{aug} is determined from the relabel operation in step 4 and step 5 of Algorithm 3.2, respectively. Let

$$\alpha_{\sum}(I) = \alpha_{relabel}(I) + \alpha_{aug}(I)$$

denote the sum of the representative operation counts. Figure 3.8(a) - (b) give the plots of $\alpha_{relabel}(I)/\alpha_{\sum}(I)$, and $\alpha_{aug}(I)/\alpha_{\sum}(I)$ for increasing larger problem instances and then look for a trend. Meanwhile, the trend on CPU time is shown in Figure 3.9(a). A plot for each different network density δ is given to help usto visualize the effects of n and m on the growth of either representative operation counts or CPU time. The plot in Figure 3.8(a) suggests that *the relabeling process is an asymptotic bottleneck operation* in Algorithm 3.2, and the plot in Figure 3.8(b) suggests that the flow augmentation is an *asymptotic nonbottleneck* operation. Moreover, Figure 3.9(a) shows that CPU time increases exponentially in denser networks. A plot of the number of augmenting paths for various network size n and δ is shown in Figure 3.9(b).

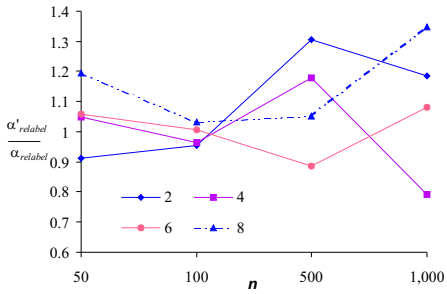


(a) CPU time (in seconds)

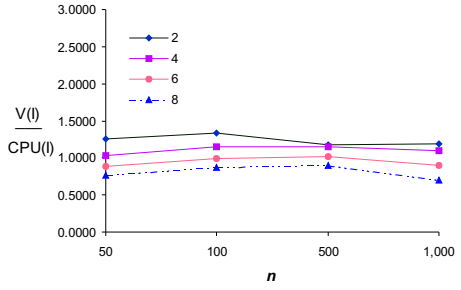


(b) The number of SDAP

Figure 3.9: CPU time and the number of shortest dynamic augmenting paths for various n and densities $\delta = m/n$



(a) The ratio of $12,206.377\ n^{0.826}\delta^{0.406}$ to $\alpha_{relabel}$



(b) The ratio of $3.729 \times 10^{-7}\ \alpha_{relabel}(I)^{1.082} \times \alpha_{aug}(I)^{0.277}$ to the CPU time

Figure 3.10: Determine the quality of the estimators of performance functions of Algorithm 3.2

To estimate the growth rate of the count of asymptotic bottleneck operation, $\alpha_{relabel}$, we

define an estimator function $\alpha'_{relabel}(I) = cn^{e_1}\delta^{e_2}$ for some choice of constants c, e_1, and e_2. The regression analysis yields

$$\alpha'_{relabel}(I) = 12,206.377\ n^{0.826}\delta^{0.406}$$

as a best fit for $\alpha_{relabel}(I)$ with adjusted R^2 value 0.899 and standard error 0.493. The values of adjusted R^2 and standard error indicate that the fit is moderately good. Figure 3.10(a) shows a plot of the ratio between the estimated and the true values which also support the goodness of the fit obtained by the regression analysis. Moreover, we obtain the virtual running time of an instance I, $V(I)$, the estimator function of CPU time, as follows:

$$V(I) = 3.729 \times 10^{-7}\ \alpha_{relabel}(I)^{1.082}\ \alpha_{aug}(I)^{0.277}$$

with adjusted R^2 value 0.966 and standard error 0.342. These values indicate that the fit is indeed very good. The plot of the ratio $V(I)/CPU(I)$ is shown in Figure 3.10(b), where $CPU(I)$ denotes the CPU time (in seconds) obtained by running Algorithm 3.2 for a problem instance I. It is also found that the linear function of $\alpha_{relabel}(I)$ and $\alpha_{aug}(I)$ does not fit well the CPU time.

Chapter 4

Earliest Arrival Flows

Gale [Gal59] introduced the earliest arrival flow problem as a variant of maximum dynamic flow problem that seeks 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. In an 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.

Minieka [Min73] and Wilkinson [Wil71] 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 [Fle01]). Under the same assumption of network attributes, Hoppe and Tardos [HT94] developed the first polynomial-time approximation algorithm. Instead of using path decomposition as in the *temporally repeated flow* technique, Hoppe and Tardos used the chain decomposition which allowing to use some backward arcs. We will discuss this approach in Section 4.2.

In a continuous-time environment, a dynamic flow represents the rate at which a commodity enters an arc at each point in time. Several results dealing with the earliest arrival flow problem in this context can be mentioned here. Ogier [Ogi88] worked on the earliest arrival flow problem on a continuous-time dynamic network with 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 [Fle01] improved Ogier's algorithm by using a generalization of parametric maximum static flow algorithm of Gallo, Grigoriadis, and Tarjan [GGT89]. The computational complexity is improved to $\mathcal{O}(k^2mn\log(kn^2/m))$. Fleischer and Skutella [FS02] generalized the earliest arrival flow problem with constant attributes by considering multiple sources.

In this chapter we focus the discussion on the discrete-time earliest arrival flow problem (DTEAFP). In the next section, we formulate the earliest arrival flow problem. In Section 4.2 we review the chain decomposition and the approximation algorithm of Hoppe and Tardos [HT94]. A new, successive earliest arrival augmenting path algorithm to solve the earliest arrival flow problem with time-dependent attributes is discussed in Section 4.3. In Section 4.4 we show that a faster algorithm is obtained when the finite waiting assumption is relaxed to infinite waiting. An illustrative example is also given in this section. Computational results on several examples based on randomly generated networks in Section 4.5 conclude this chapter.

4.1 Problem Formulation

Definition 4.1 *Earliest arrival flow for the time horizon T is a dynamic flow in which as much flow as possible arrives at the sink during any time horizon T', for every $T' \leq T$.*

Hence, an optimal solution of DTEAFP is a solution of the maximum dynamic flow problem, not only for the allotted time horizon T, but also for every smaller time horizons $T' \leq T$. Referring to DTMDNFP in Chapter 3, the objective function of a DTEAFP with single source s and single sink d can be formulated as

$$\phi_T(\mathbf{x}) = V_{\sum^{T'\leq T}}(\mathbf{x}) := \sum_{t=0}^{T'} \sum_{(i,d)\in A} \sum_{\{t':t'+\lambda_{id}(t')=t\}} x_{id}(t'),\ T' = 0,\ldots,T$$

Hence, DTEAFP is formulated as follows.

$$\begin{aligned} \text{(DTEAFP)}\quad \max\ V_{\sum^{T'\leq T}}(\mathbf{x}) \ &= \sum_{t=0}^{T'} \sum_{(i,d)\in A} \sum_{\{t':t'+\lambda_{id}(t')=t\}} x_{id}(t'), \\ &\qquad T' = 0,\ldots,T \qquad (4.1) \\ \text{Subject to}\quad &(3.3)-(3.5) \end{aligned}$$

We denote by $V_{\sum^{T'\leq T}}$ *the value of a discrete-time earliest arrival flow for a time horizon* T. When the waiting capacity is infinite, we denote this problem by (see Appendix D)

$$(s,d)/(\lambda(t),u(t))/V_{\sum^{T'\leq T}}$$

and by

$$(s,d)/(\lambda(t),u(t),a(t))/V_{\sum^{T'\leq T}}$$

when the waiting capacity is finite.

By definition, every earliest arrival flow is a maximum dynamic flow, but the converse is not true as illustrated by the flow distribution in Figure 4.1 using the data of Example 3.2. A dynamic flow indicated by the solid line in Figure 4.1 is a maximum dynamic flow for $T = 7$ but not an earliest arrival flow, since the value of this flow for $T^{'} = 3$ is only zero. In fact, the maximum dynamic flow for $T^{'} = 3$ has value 1.

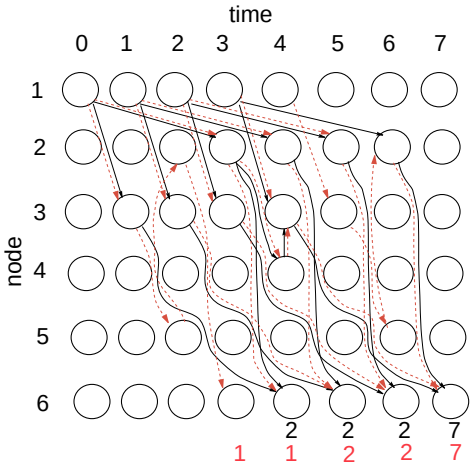


Figure 4.1: Discrete-time maximum dynamic flow vs discrete-time earliest arrival flow. The solid lines indicate a maximum dynamic flow without having the earliest arrival property, and the dashed lines indicate an earliest arrival flow

Theorem 4.1 (Minieka [Min73]) *There is always a maximum dynamic flow for the time horizon T with the earliest arrival property.*

Proof:
The proof is done if we can generate an earliest arrival flow for the time horizon T. This flow will be generated on a time-expanded network. Let us denote by s^T and d^T the super source and the super sink of the associated time-expanded network G_T of $G = (N, A, T)$, respectively. The super source s^T is connected to every time-copy of source node $s \in N$. Furthermore, every time-copy of sink node $d \in N$ is connected to the super sink d^T (see again Figures 2.4). All these connections have infinite arc capacities and zero travel times. Algorithm 4.1 finds the earliest arrival flow of G. For every times $t_1, t_2 \in \{0, \ldots, T\}$ with $t_1 < t_2$, when step 1 of the Algorithm 4.1 maximizes the flow via arc $(d(t_2), d^T)$, it does not change the flow entering $(d(t_1), d^T)$, because the maximization algorithm will

never reroute a flow already at the sink. Consequently, step 1 applied on $t = t_2$ yields a maximum dynamic flow for any time $t \leq t_2$. Hence, an earliest arrival flow can always be generated for any time horizon T. ■

Algorithm 4.1 (Minieka [Min73]): Solving $(s,d)/(\lambda(t),u(t),a(t))/V_{\sum^{T'\leq T}}$ on the time-expanded network

INPUT	Time expanded network G_T.
OUTPUT	Earliest arrival flow $\mathbf{x}$.
0	Set the time period t to 1.
1	Maximize the flow $\mathbf{x}$ from s^T to d^T via arc $(d(t), d^T) \in A_T$ by setting temporarily the capacity of arcs $(d(t'), d^T) \in A_T, t' > t$ to zero (i.e., close temporarily those arcs).
2	If $t = T$ stop. Otherwise, increase t by 1, set the capacity of arc $(d(t), d^T)$ back to infinity and go to step 1 using the current flow $\mathbf{x}$ as the starting flow.

4.2 Solution Algorithm for an Earliest Arrival Flow Problem with Constant Attributes

In this section we review an approximation algorithm of Hoppe and Tardos [HT94]. Before going further, we need the notion of residual network of a static flow. In a static network flow, the travel times are considered as costs.

Definition 4.2 (Residual static network) *Consider a static network $G_{stat} = (N, A)$. The residual network with respect to the feasible static flow $\mathbf{f}$ is defined as $G^f_{stat} = (N, A^+_f \cup A^-_f)$ with arc set $A^+_f := \{(i,j) : (i,j) \in A,\ f_{ij} < u_{ij}\}$ and $A^-_f := \{(j,i) : (i,j) \in A,\ f_{ij} > 0\}$. The residual travel times are*

$$\lambda^f_{ij} := \begin{cases} \lambda_{ij} & ,(i,j) \in A^+_f \\ -\lambda_{ji} & ,(i,j) \in A^-_f \end{cases}$$

and the residual capacities are

$$u^f_{ij} := \begin{cases} u_{ij} - f_{ij} & ,(i,j) \in A^+_f \\ f_{ji} & ,(i,j) \in A^-_f \end{cases}$$

Recall the antisymmetry of the dynamic network G and Definition 3.2 on path flow and path decomposition. Hoppe and Tardos [HT94] introduced a *chain decomposition* (also called *non-standard path decomposition*) in G^f_{stat}.

Definition 4.3 (chain decomposition)
$\mathbb{P} = \{P_1, P_2, \ldots, P_k\}$ *is a chain decomposition of a static flow* $\mathbf{f}$ *if* $\mathbf{f} = \sum_{l=1}^{k} \gamma_{P_l}$ *and it may contains path flow in* $\mathbb{P}$ *that using arcs in opposite directions, i.e. the path contains some arc* (j,i) *with* $(i,j) \in A$. *Consequently, we call* $P_l \in \mathbb{P}$ *as a chain flow instead of path flow.*

Hence, a chain flow may use residual arcs with negative travel times.

Example 4.1
Consider a static network shown in Figure 4.2(a) with node 1 the source node and node 4 the sink node. Suppose $\mathbb{P}$ is a set of paths with $\mathbb{P} = \{P_1, P_2\}$ where :

$$P_1 = \{1,2,3,4\} \quad ; \quad P_2 = \{1,3,2,4\}$$

with $v(P_1) = 1$ and $v(P_2) = 1$, respectively. $\mathbb{P}$ is a chain decomposition since $P_2 \in \mathbb{P}$ uses arcs $(2,3) \in A$ in the opposite direction (see Figure 4.2 (b)) , i.e. P_2 uses arc $(3,2) \notin A$. Figure 4.2(c) shows a dynamic flow induced by the chain decomposition in Figure 4.2(b). □

Suppose that the arc capacities are integral. Let us denote by U the maximum arc capacity, i.e.

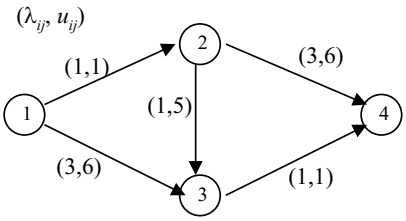
$$U := \max_{(i,j)\in A} \{u_{ij}\}$$

By using the chain decomposition, Hoppe and Tardos [HT94] developed the first polynomial-time approximation algorithm, with time complexity $\mathcal{O}(\frac{m}{\epsilon}(m+n \log n)\log U)$. It is proved to be within $(1+\epsilon)$ of optimality, i.e. if $\mathbf{x}$ is a dynamic flow for a time horizon T provided by the algorithm and $V_{\sum^{T'}}$ is the value of maximum dynamic flow for any time horizon $T' \leq T$, then

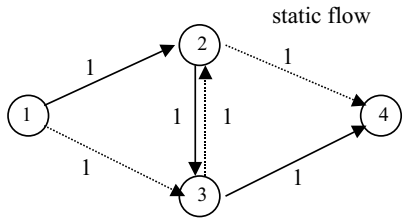
$$V_{\sum^{T'}} \leq (1+\epsilon) V_{\sum^{T'}}(\mathbf{x}) \tag{4.2}$$

This algorithm is a *capacity scaling shortest augmenting path* algorithm applied to the static network. The capacity scaling is done in an *upward* direction. The following example shows that the downward capacity scaling algorithm does not solve the earliest arrival flow problem.

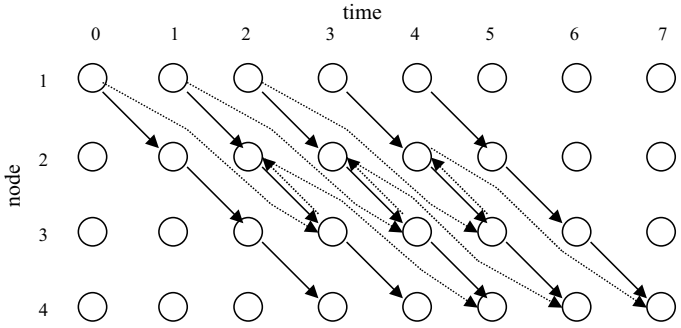
Example 4.2
Consider a network shown in Figure 4.3 with node 0 the source node and node 3 the sink



(a) A static network $G_{stat} = (N, A)$



(b) A chain decomposition



(c) The induced dynamic flows

Figure 4.2: A chain decomposition and its associated induced dynamic flow

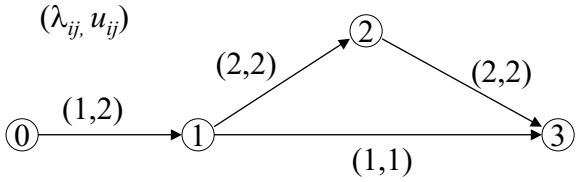


Figure 4.3: A network for Example 4.2

node. Suppose that the time horizon T is 5 time units. An optimal solution $\mathbf{x}$, obtained by using the downward capacity scaling, and the earliest arrival flow $\mathbf{x}^*$ are given in the following table.

T'	0	1	2	3	4	5
$V_{\sum^{T'}}(\mathbf{x})$	0	0	0	1	1	3
$V_{\sum^{T'}}(\mathbf{x}^*)$	0	0	1	1	1	2

□

This example shows a fact that a small capacity arc that is short might carry more flow than a large capacity that is long.

The detail procedure of the approximation algorithm given in Algorithm 4.2 is explained as follows. Let us denote by $\triangle$ the capacity scaling factor. The algorithm starts with $\triangle = 1$ and iteratively doubles this factor until no augmenting path of length less than or equal to T exists in the residual static network G^f_{stat}. The initial iteration uses G^f_{stat} with all residual capacities are evenly divisibly by $\triangle$ and find consecutively shortest augmenting paths of distance (with respect to the travel times) $\leq T$ until the value of the static flow exceeds $m\triangle/\epsilon$ with m the cardinality of A. At this point, the scaling factor is doubled (i.e. $\triangle := 2\triangle$) and the residual capacities are rounded down by $\triangle$ as

$$u^f_{ij} := u^f_{ij} - (u^f_{ij} \bmod \triangle),\ \forall (i,j) \in A_f \tag{4.3}$$

Since all residual capacities are integer multiplies of $\triangle$, the consecutive augmentation has at least $\triangle$ units of flows. If $\mathbf{f}$ is the current static flow and P is the shortest augmenting path in G^f_{stat}, we define the maximum flow augmentation along P as

$$v_{max}(P) := \min \{u^f_{ij} \ :\ (i,j) \in P\} \tag{4.4}$$

A new static flow is obtained as follows.

$$f_{ij} \ := \ \begin{cases} f_{ij} + v_{max}(P) & ,\text{if } (i,j) \in P \\ f_{ij} - v_{max}(P) & ,\text{if } (j,i) \in P \\ f_{ij} & ,\text{otherwise} \end{cases} ,\ \forall (i,j) \in A \tag{4.5}$$

We denote the static flow given by (4.5) by $\mathbf{f} = \mathbf{f} \pm v_{max}(P)$. In each augmentation, the flow along a path is added to a set of path flows $\mathbb{P}$ that will induce a dynamic flow at the end of the algorithm.

4.3 Solution Algorithm for an Earliest Arrival Flow Problem with Time-Dependent Attributes

To solve a DTEAFP with time-dependent attributes, we adapt a well known successive shortest augmenting path technique for solving the static maximum flow problem (see e.g. [AMO93]). Instead of looking for a shortest $s-d$ augmenting path in the residual static network, we look for an $s-d$ augmenting path with the earliest arrival time at node d in the residual dynamic network. Therefore, we call this technique as the *successive earliest arrival augmenting path algorithm*. A dynamic augmenting path is defined as in Definition 3.4. A new flow distribution $\mathbf{x}$ is computed as in (3.63).

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 dynamic network. Since the residual dynamic 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

Algorithm 4.2 (Hoppe and Tardos [HT94]) :
Solving $(s,d)/(\lambda,u,a)/V_{\sum^{T'}\leq T}$

INPUT	Dynamic network $G=(N,A,T)$, constant capacity function u_{ij}, travel time λ_{ij}, and ϵ.
OUTPUT	Earliest arrival flow $\mathbf{x}$.
0	Set $\triangle = 1$, $\mathbf{f} = 0$, $\mathbb{P} = \emptyset$.
1	If there exists $s-d$ path P in G^f_{stat} of total travel time $\leq T$, then set $\sigma = 0$. Otherwise, go to Step 5.
2	If $\sigma < \frac{m\triangle}{\epsilon}$ and there exists a shortest $s-d$ path P in G^f_{stat} of total travel time $\leq T$, then find P. Otherwise, go to step 4.
3	Augment $\mathbf{f}$ with $v_{max}(P)$ following (4.5) and extend $\mathbb{P} := \mathbb{P} \cup \{P\}$. Update the residual network G^f_{stat}, $\sigma := \sigma + v_{max}(P)$, and go to step 2.
4	Increase the capacity scaling $\triangle := 2\triangle$. Modify the residual arc capacities according to (4.3). Continue the iterative process by going to step 1.
5	If $\mathbb{P} \neq \emptyset$, then the dynamic flow $\mathbf{x}$ is obtained by repeating all path flows in $\mathbb{P}$ and $V_{\sum^{T'}} \leq (1+\epsilon) V_{\sum^{T'}}(\mathbf{x}),\ \forall T' \leq T.$ Otherwise, the problem is infeasible.

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 a 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 a 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 that $pred_j(t+\lambda_{ij}(t)) = \infty$), 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., $a_j^{x+}(t'-1) > 0$) 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 condition $a_j^{x-}(t'+1) > 0$ 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 a 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 4.3.

Algorithm 4.3 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 the construction of a 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 $t+\lambda_{ji}(t) = t'$. Suppose that the cycle continues to reach node k (there may be some nodes in between) at time t_k and arrive back at node j from k at time $t_k+\lambda_{kj}(t_k)$. We denote the distance (with respect to the travel times) between node i at time $t+\lambda_{ji}(t)$ and node k at time t_k along the cycle as Λ_{ik}. The total travel time to complete one cycle is $-\lambda_{ij}(t')+\Lambda_{ik}+\lambda_{kj}(t_k)$. Since the cycle is a negative cycle, $-\lambda_{ij}(t')+\Lambda_{ik}+\lambda_{kj}(t_k) < 0$. This implies

$$\Lambda_{ik}+\lambda_{kj}(t_k) < \lambda_{ij}(t') \tag{4.6}$$

Algorithm 4.3 (Earliest Arrival Augmenting Path Algorithm) : finding an $s-d$ earliest arrival augmenting path

INPUT The residual dynamic network $G_x := (N, A_x^+ \cup A_x^-, T)$ as given by Definition 2.5
OUTPUT An $s-d$ 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 $t \in \{0, \ldots, T\}$

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

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 the 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 $(t' \le T, \ a_j^{x+}(t'-1) > 0, \text{ and } \ pred_j(t') = \infty))$
do $\{pred_j(t') = j \quad ; \quad dep_j(t') := t' - 1 \ ; \ t'++\}$
Define $t' := t + \lambda_{ij}(t) - 1$
While $(t' \ge 0, \ a_j^{x-}(t'+1) > 0, \text{ and } \ pred_j(t') = \infty)$
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 an $s-d$ 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 $P := P + \{i'(t', t'')\}$ }
}

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

$$\lambda_{ij}(t') \le \Lambda_{ik} + \lambda_{kj}(t_k)$$

contradicting (4.6). ■

To analyze the complexity of Algorithm 4.3, we divide its execution into *passes*. We define a pass as follows.

Definition 4.4

- *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 *If the residual dynamic network* G_x *has some* $s-d$ *dynamic augmenting paths, then Algorithm 4.3 finds an* $s-d$ *earliest arrival augmenting path in* $\mathcal{O}(nmT^2)$.

Proof:
Let us denote a set $\{t \ : \ pred_i(t) \neq \infty, \ t \leq T\}$ by $PRED_i$. When Algorithm 4.3 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 4.3 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 4.4.

Algorithm 4.4 : Solving $(s,d)/(\lambda(t),u(t),a(t))/V_{\sum^{T'\leq T}}$

INPUT	Network $G=(N,A,T)$, time-dependent travel time $\lambda_{ij}(t)$, capacity $u_{ij}(t)$, holdover capacity $a(t)$.
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 4.3 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 4.4 produces maximum dynamic flow, we will use the dynamic cut. The termination of Algorithm 4.4 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 give a specific definition to the function C_T in (3.6) as follows.

$$C_T(t) \ := \ \{i : pred_i(t)\neq\infty,\ i\in N\} \tag{4.7}$$

and

$$\overline{C_T}(t) \ := \ \{i : pred_i(t)=\infty,\ i\in N\} \tag{4.8}$$

By this definition, when Algorithm 4.4 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

$$\Gamma_{ij}^T \ = \ \{\ t\ :\ pred_i(t)\neq\infty,\ pred_j(t+\lambda_{ij}(t))=\infty,\ t+\lambda_{ij}(t)\leq T\} \tag{4.9}$$

and

$$\Gamma_{ii}^T \ = \ \{\ t\ :\ pred_i(t)\neq\infty,\ pred_i(t+1)=\infty,\ t+1\leq T\}, \tag{4.10}$$

respectively.

Proposition 4.3 *When Algorithm 4.4 terminates, the set of time-expanded arcs* $(C_T, \overline{C_T})$ *as defined by* (4.7)-(4.8) *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.*

The proof of this proposition is similar to the proof of Proposition 3.8.

Finally, the correctness that Algorithm 4.4 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.*

$$U := \max_{(i,j)\in A} \max_{t\in\{0,\ldots,T\}} \{u_{ij}(t)\} \tag{4.11}$$

Algorithm 4.4 solves

$$(s,d)/(\lambda(t), u(t), a(t))/V_{\sum^{T'\leq T}}$$

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 4.4 produces a maximum dynamic flow with the 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 $\mathcal{O}(nm^2T^3U)$. ■

4.4 Infinite Waiting

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

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

Proposition 4.5 *Suppose that the waiting 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 4.5 has the following direct consequence.

Corollary 4.1 *Consider the case when infinite waiting in every node $i \in N - \{s, d\}$ is allowed. Assume that during an iteration in Algorithm* 4.3*, 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 4.3 can be simplified as given in Table 4.1.

Scan the 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 $(t' \geq 0,\ a_j^{x-}(t' + 1) > 0,$ 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 4.1: Modified step 2 of Algorithm 4.3 when infinite waiting is allowed

Another important consequence of Proposition 4.5 is stated by the following proposition.

Proposition 4.6 *If the waiting is infinite, by applying the scaning and updating processes given in Table 4.1 to step 2 of Algorithm 4.3, 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 4.5, there are at most $n - 1$ passes (instead of $n(T +$

1) − 1 passes in the case of finite waiting). Consequently, an earliest arrival augmenting path can be found in $\mathcal{O}(nmT)$. ■

Corollary 4.2 *Algorithm 4.4 solves* $(s,d)/(\lambda(t),u(t),a(t))/V_{\sum^{T'\leq T}}$ *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 4.3 *Under the assumption of infinite waiting, Algorithm 4.4 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 4.7 *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 (4.7) and Proposition 4.5. ■

The following Example 4.3 illustrates the implementation of Algorithms 4.3 and 4.4.

Example 4.3
Consider again the network problem of Example 3.3. The network structure is shown in

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 4.2: Labels on nodes after completing Algorithm 4.3

Figure 3.7. 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 4.4 are given in Table 4.2. Since $\pi_5 = 4 < T = 7$, an augmenting path P_1 and the corresponding maximum flow augmentation $\epsilon(P_1)$ can be obtained by a backtracking procedure as described in step 3 of Algorithm 4.3. We obtain $P_1 = \{0(0,0), 2(2,2), 4(3,3), 1(3,3), 5(4,4)\}$ with $\epsilon(P_1) = 2$.

(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 4.3: Time-dependent travel times and capacities for the residual network after the first pass of Algorithm 4.3

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 4.4: Labels on nodes after completing Algorithm 4.3 for the fifth times

The time-dependent travel times and capacities of the new residual network are given in Table 4.3. 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 (3.63). The process is then continued to look for other earliest augmenting paths. Three additional earliest augmenting paths have been found before the stopping criterion of Algorithm 4.4 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 $\epsilon(P_4) = 1$.

$x_{ij}(t)$	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
Value of the flow arriving at the sink					2	0	3	1
					2	0	3	2

Table 4.5: Earliest arrival flow of Example 4.3. The upper position of each row $x_{ij}(t)$ contains an optimal flow when the waiting at a node is limited, while the lower one deals with the case when infinite waiting is allowed for every node

The final labels (i.e. after the fifth pass of Algorithm 4.3) which are not able to generate any augmenting path, are shown in Table 4.4. The earliest arrival flow $\mathbf{x}$ having value $V_{\sum^{T=7}}(\mathbf{x}) = 6$ is given in Table 4.5. Moreover, $\mathbf{x}$ has the same value as the maximum dynamic flow of Example 3.3. However, it has two units of flow which arrive one unit time earlier at the sink node.

From the final labels given in Table 4.4, we can construct the sets Γ_{ij}^T for every arc $(i,j) \in A$ and Γ_{ii}^T for every node $i \in N - \{0,5\}$. The results are shown in Table 4.6. Figure 4.4 (a) depicts a minimum dynamic cut $(C_7, \overline{C_7})$ and the corresponding optimal dynamic flow.

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 4.6: 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 4.3

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

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

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

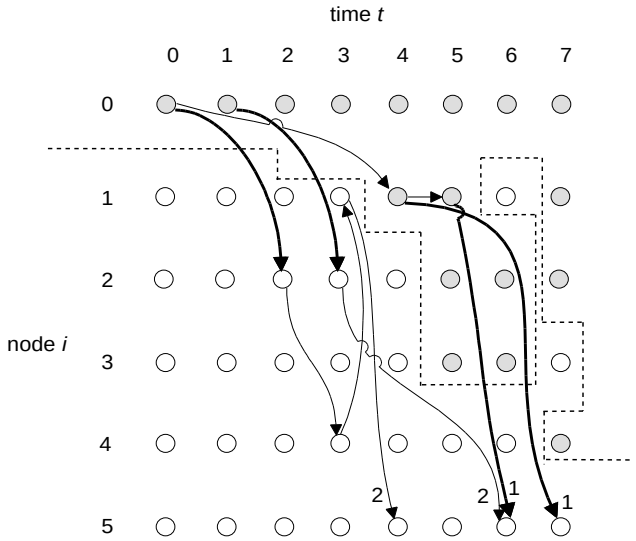
When infinite waiting at any node is allowed, including the waiting at the source node 0, the value of earliest arrival flow changes to $V_{\sum^{T=7}} = 7$. The optimal flow distribution of this case is given in the lower position of each row $x_{ij}(t)$ in Table 4.5. Figure 4.4 (b) depicts the optimal flow distribution and dynamic cut when infinite waiting at any node is allowed. □

4.5 Computational Results

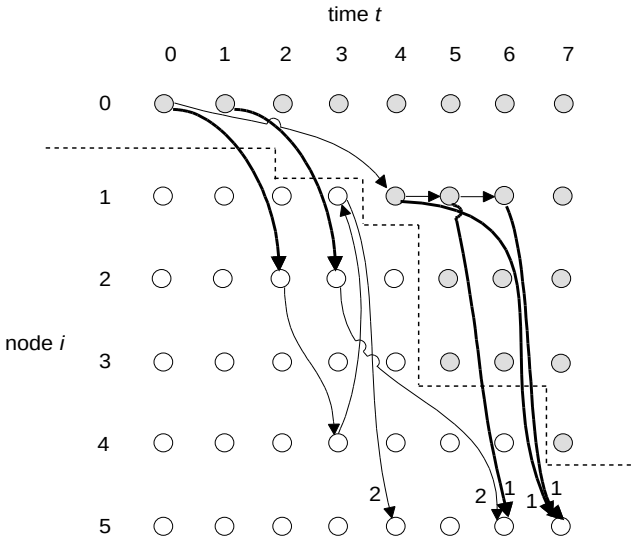
In computational testing, we again rely on the representative operation counts, as described by Ahuja, Magnanti, and Orlin in [AMO93],

- to identify the asymptotic bottleneck operations and
- to estimate the running time for different problems sizes

of Algorithm 4.4. For these experiments, Algorithm 4.4 is implemented in C++ and run on a PC Pentium III, 500 MHz, and RAM of 256 MB. The same random networks, generated in Section 3.5 for testing Algorithm 3.2, are put to use.



(a) The minimum dynamic cut with finite waiting.



(b) The minimum dynamic cut with infinite waiting

Figure 4.4: The minimum dynamic cut of Example 4.3 The shaded circles are in the source side of the cut. The bold arrows represent the optimal flows of the arcs in the cut. The numbers beside the arrows define the values of the flows

Here we identify the following set of representative operations:

(a) scanning process of node-time pairs by a current node to find an earliest arrival augmenting path

(b) flow augmentation process

We denote by $\alpha_{scan}(I)$ the number of node-time-pairs scanned in the representative operation (a), summed over all current nodes and all existing augmenting paths and denote by $\alpha_{aug}(I)$ the number of flow augmentation in the representative operation (b), summed over all existing augmenting paths, of a problem instance I. The value of α_{scan} and α_{aug} is determined from the scanning operation in step 2 of Algorithm 4.3 and step 1 of Algorithm 4.4, respectively. Let

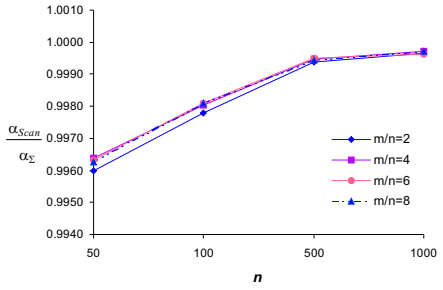
$$\alpha_{\sum}(I) = \alpha_{scan}(I) + \alpha_{aug}(I)$$

denote the sum of the representative operation counts. Figure 4.5(a) - (b) give the plots of $\alpha_{scan}(I)/\alpha_{\sum}(I)$, and $\alpha_{aug}(I)/\alpha_{\sum}(I)$ for increasing larger problem instances and then look for a trend. While, the trend on CPU time is shown in Figure 4.6(a). The plot in Figure 4.5(a) suggests that *the scanning process is an asymptotic bottleneck operation* in Algorithm 4.3, and the plot in Figure 4.5(b) suggest that the flow augmentation is an *asymptotic nonbottleneck* operation. Moreover, Figure 4.6(a) shows that CPU time increases faster in the more dense networks. A plot of the number of augmenting paths for various network size n and δ is shown in Figure 4.6(b).

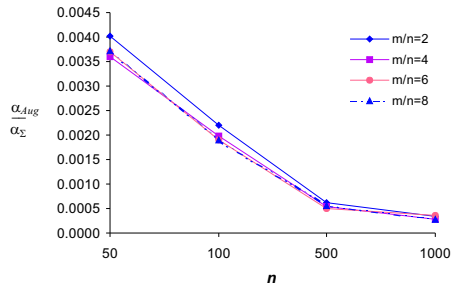
To estimate the growth rate of the count of asymptotic bottleneck operation α_{scan}, we define an estimated function $\alpha'_{scan}(I) = cn^{e_1}\delta^{e_2}$ for some choices of constants c, e_1, and e_2. The regression analysis yields

$$\alpha'_{scan}(I) = 18,729.768\ n^{1.180}\delta^{0.530}$$

as a best fit for $\alpha_{scan}(I)$ with adjusted R^2 value 0.960 and standard error 0.429. The value of adjusted R^2 which close to 1 and a fairly small standard error indicate that the fit is indeed very good. Figure 4.7(a) shows a plot of the ratio between the estimated and the true values which also support the goodness of the fit obtained by the regression analysis.

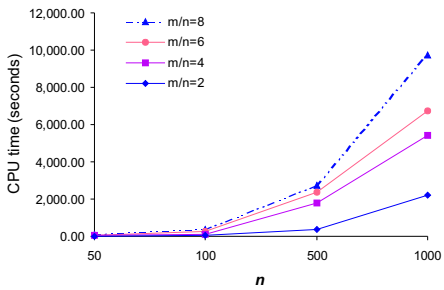


(a) A plot of the ratio of scanned node-time-pairs in scanning operation to the total number of operations

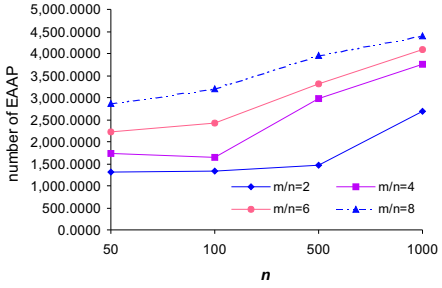


(b) A plot of the ratio of flow augmentation to the total number of operations

Figure 4.5: Identifying asymptotic bottleneck operations in Algorithm 4.4



(a) CPU time (in seconds)



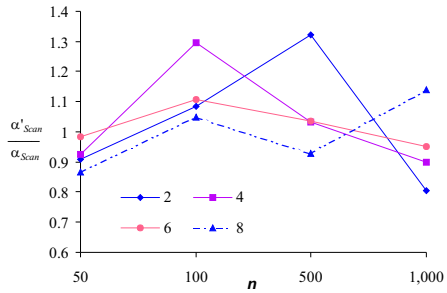
(b) The number of EAAP

Figure 4.6: CPU time and the number of earliest arrival augmenting paths for various n and densities $\delta = m/n$

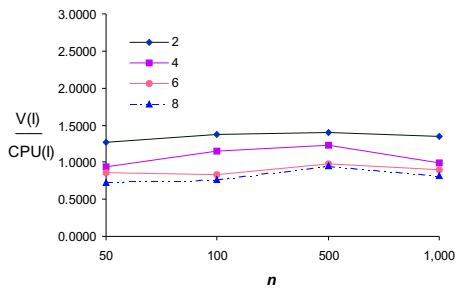
Moreover, we obtain the virtual running time of an instance I, $V(I)$, as an estimate function of CPU time, as follows:

$$V(I) = 1.48810^{-8}\ \alpha_{scan}(I)^{1.086}\ \alpha_{aug}(I)^{0.564}$$

with adjusted R^2 value 0.973 and standard error 0.459. A plot of the ratio $V(I)/CPU(I)$ is shown in Figure 4.7(b), where $CPU(I)$ denotes the CPU time (in seconds) obtained by running Algorithm 4.4 for a problem instance I. It is also found that the linear function of $\alpha_{scan}(I)$ and $\alpha_{aug}(I)$ does not fit well the CPU time.



(a) The ratio of $18,729.768\ n^{1.180}\delta^{0.530}$ to α_{scan}



(b) The ratio of $1.48810^{-8}\ \alpha_{scan}(I)^{1.086} \times \alpha_{aug}(I)^{0.564}$ to the CPU time

Figure 4.7: Determining the quality of the estimators of performance functions of Algorithm 4.4

Chapter 5

Quickest Flow Problems

The quickest flow problem seeks a discrete-time dynamic flow which sends the given ν units of flow from the source s to the sink d of the network in minimum time (see e.g., Burkard, Dlaska, and Klinz [BDK93] and Fleischer and Tardos [FT98]). Concerning the evacuation problem, Hamacher and Tjandra [HT02] used this quickest flow model to find the minimum evacuation time of a given number of building occupants. In this application, ν is considered as the initial contents of the network, i.e., $\nu := q_s(0)$.

When only one path can be used for sending the flow, this problem is known as the *quickest path problem* (see e.g., Chen and Chin [CC90] and Rosen, Sun, and Xue [RSX91]). When a network has multiple sources and multiple sinks, the corresponding quickest flow problem is called the *quickest transshipment problem*. Jarvis and Ratliff [JR82] showed that the earliest arrival flow with value at least ν solves the quickest flow problem. Burkard, Dlaska, and Klinz [BDK93] gave several polynomial algorithms and a strongly polynomial algorithm to solve the quickest flow problem with constant attributes. Fleischer and Skutella [FS02] generalized the quickest flow problem by considering several commodities.

In this chapter we focus the discussion on the discrete-time quickest flow problems (DTQFP). Three different assumptions on the given value ν and the network attributes will be covered.

- $\nu := q_s(0)$ and the network attributes are constant, i.e.,

$$\lambda_{ij}(t) := \lambda_{ij},\ u_{ij}(t) := u_{ij},\ (i,j) \in A,\ \text{and} \quad a_i(t) := a_i,\ i \in N,\ t \in \{0, \ldots, T\}$$

  Waiting capacities are considered infinite. We denote this problem by

$$(s,d)/(\lambda,u,a)/T_{\sum^{q_s(0)}}$$

- $\nu := q_s(0)$ and time-dependent network attributes, where we denote this problem by

$$(s,d)/(\lambda(t),u(t),a(t))/T_{\sum^{q_s(0)}}$$

Here, we also discuss how to tackle a similar problem with multiple sources and single sink.

- In a more general DTQFP, we consider $\nu := \sum_{t=0}^{T} q_s(t)$ with $q_s(t) > 0$ for some $t \in \{0, \ldots, T\}$ as the supply to the source s which may vary over time. We denote this problem by

$$(s, d)/(\lambda(t), u(t), a(t))/T_{\sum^{q(t)}}$$

In the next section, we formulate the quickest flow problem with $\nu := q_s(0)$. In Section 5.2 we review the polynomial algorithm of Burkard, Dlaska, and Klinz [BDK93]. We discuss the solution algorithm for the network with time-dependent attributes in Section 5.3. The discussion on a solution algorithm for the quickest flow problem with time-dependent supplies in Section 5.4 concludes this chapter.

5.1 Problem Formulation

The objective function of DTQFP with $q_s(0)$ units of flow can be formulated simply as minimizing the time T to clear the network with $q_s(0)$ initial contents. This DTQFP with a single source s and a single sink d is formulated as follows.

$$\text{(DTQFP)} \qquad \min \quad T \tag{5.1}$$

Subject to

$$\sum_{(j,i)\in A} \sum_{\{t' : t'+\lambda_{ji}(t')=t\}} x_{ji}(t') - \sum_{\{(i,j)\,:\,(i,j)\in A,\ t+\lambda_{ij}(t)\le T\}} x_{ij}(t) = \begin{cases} x_{ii}(t) - q_s(0) & , \ i = s,\ t = 0 \\ x_{ii}(t) - x_{ii}(t-1) & , \ \text{otherwise} \end{cases},$$

$$i \in N - \{d\},\ t = 0, \ldots, T \tag{5.2}$$

$$x_{ii}(t) = 0,\ t \ge T,\ i \in N \tag{5.3}$$

$$x_{ij}(t) = 0,\ t + \lambda_{ij}(t) > T,\ (i,j) \in A \tag{5.4}$$

$$0 \le x_{ii}(t) \le a_i(t),\ t = 0, \ldots, T,\ i \in N - \{d\} \tag{5.5}$$

$$0 \le x_{ij}(t) \le u_{ij}(t),\ t + \lambda_{ij}(t) \le T,\ (i,j) \in A \tag{5.6}$$

We denote by $T_{\sum^{q_s(0)}}$ *the travel time of a discrete-time quickest flow (i.e., the minimum clearing time) of a given initial content* $q_s(0)$, namely

$$T_{\sum^{q_s(0)}} = \min\{T \ : \ V_{\sum^T} \ge q_s(0)\} \tag{5.7}$$

5.2 Quickest Flow Problem with Constant Attributes

Recall, $V_{\sum^T}$ is the value of a maximum dynamic flow for a time horizon T. In order to tackle DTQFP, some properties of $V_{\sum^T}$ are stated next.

Theorem 5.1 (Burkard, Dlaska, and Klinz [BDK93])
Let λ be the length of a shortest $s-d$ path with respect to the travel times. Then the following results hold.

(a) $V_{\sum^T}$ *is a monotonously increasing function and for* $T \geq \lambda$ *it increases strictly.*

(b) $\triangle(T) := V_{\sum^T} - V_{\sum^{T-1}}$ *is monotonously increasing, i.e.,*

$$\triangle(T+1) \geq \triangle(T), \forall T \geq 1$$

(c) *Let v_{max} be the value of a maximum static flow (i.e., maximum flow in the static network $G_{stat} = (N, A)$). Then, $\triangle(T)$ can attain its value only from the set $\{0, 1, \ldots, v_{max}\}$.*

Corollary 5.1 (Burkard, Dlaska, and Klinz [BDK93]) $V_{\sum^T} \geq q_s(0)$ *is achieved by at least* $\lceil \frac{q_s(0)-V_{\sum^0}}{v_{max}} \rceil$ *time units, i.e.,*

$$T_{\sum}(q_s(0)) \geq \lceil \frac{q_s(0) - V_{\sum^0}}{v_{max}} \rceil$$

The interrelations between the maximum dynamic flow and the quickest flow are described by the following lemma.

Lemma 5.1 (Burkard, Dlaska, and Klinz [BDK93])

(a) *Let* $\mathbf{x}$ *be a dynamic flow of value $q_s(0)$ for the time horizon T with $T \geq 0$. If $V_{\sum^{T-1}} < q_s(0)$ then* $\mathbf{x}$ *is a quickest flow of value $q_s(0)$ and $T_{\sum^{q_s(0)}} = T$*

(b) *For $T' = T_{\sum^{q_s(0)}} - 1$ we get $V_{\sum^{T'}} < q_s(0)$*

Proof :
The proofs of (a) and (b) follow immediately from the assumption of the lemma and (5.7). ■

The solution is obtained by applying an iterative process with two main steps. The first step estimates the time period T by utilizing a *binary search*, *Newton* or *interpolation technique*. The second one solves the maximum dynamic flow problem for the time horizon T. By Theorem 3.6, the maximum dynamic flow can be obtained by solving a minimum cost circulation static flow problem with parameter T. The process is repeated until $V_{\sum^T} \approx q_s(0)$. The solution algorithm by utilizing a binary search is shown in Algorithm 5.1.

The initial interval $[T_l, T_u]$ can be determined by following proposition.

Algorithm 5.1 (Burkard, Dlaska, and Klinz [BDK93]) :
Solving $(s,d)/(\lambda,u,a)/T_{\sum^{q_s(0)}}$

INPUT	Network $G=(N,A,T)$, constant capacity function u_{ij}, travel time λ_{ij}, and $q_s(0)$.
OUTPUT	$T_{\sum^{q_s(0)}}$.
0	Calculate the initial lower bound T_l and upper bound T_u of $T_{\sum^{q_s(0)}}$, i.e., $T_{\sum^{q_s(0)}} \in [T_l, T_u]$.
1	Calculate the mid point T_c of an interval $[T_l, T_u]$, i.e., $T_c = T_l + \frac{T_u - T_l}{2}$. Calculate $V_{\sum^{T_c}}$ by applying, for example, Algorithm 3.1.
2	If $V_{\sum^{T_c}} > q_s(0)$ then set $T_u = T_c$ and go to Step 1. Else if $V_{\sum^{T_c}} < q_s(0)$ then set $T_l = T_c$ and go to Step 1. Else, the quickest time is $T_{\sum^{q_s(0)}} = T_c$ and terminate the algorithm.

Proposition 5.1 (Burkard, Dlaska, and Klinz [BDK93]) *The initial values for T_l can be taken as*

$$T_l := \max\left\{\lambda, \left\lceil \frac{q_s(0) - V_{\sum^0}}{v_{max}} \right\rceil\right\} \tag{5.8}$$

And if v_l denotes the value of a static flow that induces a TRF of maximum value for $T = T_l$, then the initial value for T_u can be taken as

$$T_u := T_l + \left\lceil \frac{q_s(0) - V_{\sum^{T_l}}}{v_l} \right\rceil \tag{5.9}$$

Proof:
Corollary 5.1 and the fact that no flow can reach the sink for $T < \lambda$ validate (5.8). Since v_l induces a TRF of maximum value for $T = T_l$ and $\triangle(T)$ is a monotonously increasing, $\triangle(T_l + 1) \geq v_l$. Therefore, we need at most $\left\lceil \frac{q_s(0) - V_{\sum^{T_l}}}{v_l} \right\rceil$ additional time units in order to increase the current value $V_{\sum^{T_l}}$ to $q_s(0)$ which validates (5.9). ■

Using the initial values as defined in (5.8)-(5.9), step 0 of Algorithm 5.1 requires some calculations to solve one static shortest path problem (to determine λ), one static maximum flow problem (to determine v_{max}) and two minimum cost circulation problems (MCCP) to obtain $V_{\sum^{T_l}}$ and $V_{\sum^{T_u}}$. Since finding the midpoint T_c can be done in a constant time, the calculation in Step 1 is dominated by the calculation to solve a MCCP in order to find $V_{\sum^{T_c}}$. Moreover, since the length of the initial interval is less than $q_s(0)$ and maximum

value of $\triangle(T)$ is v_{max}, then the maximum number of binary search iteration is

$$\min\{\lceil \log q_s(0) \rceil + 1, v_{max}\}$$

Hence, the overall complexity of the Algorithm 5.1 is

$$\mathcal{O}(\min\{\lceil \log q_s(0) \rceil, v_{max}\}\ \mathcal{O}(MCCP))$$

5.3 Quickest Flow Problem with Time-Dependent Attributes

A solution of DTQFP with time-dependent attributes can be obtained by utilizing the solution of the corresponding DTEAF (i.e., Algorithm 4.4) as stated by the following theorem.

Theorem 5.2 *Suppose* $\mathbf{x}$ *is an earliest arrival flow for a time horizon* T *with value* $V_{\sum^T}(\mathbf{x})$. *If*

$$T^* := \min\{T' \ :\ V_{\sum^{T'}}(\mathbf{x}) \geq q_s(0),\ T' \leq T\} \tag{5.10}$$

exists, then T^* *solves the discrete-time quickest flow problem with initial contents* $q_s(0)$.

Proof :
If T^* does not solve the quickest flow problem, then there is $T'' < T^*$ and flow $\mathbf{x}''$ with value $V_{\sum^{T''}}(\mathbf{x}'') = q_s(0)$. Hence, $V_{\sum^{T''}}(\mathbf{x}) < V_{\sum^{T''}}(\mathbf{x}'')$. This contradicts the fact that $\mathbf{x}$ is also maximum for $T'' < T$. ■

Before implementing Algorithm 4.4 to solve this quickest flow problem, we need to modify the original network G to facilitate the initial supply $q_s(0)$. A super source s^* and an arc (s^*, s) is added to N and A, respectively, i.e.,

$$N := N \cup \{s^*\} \quad ; \quad A := A \cup \{(s^*, s)\}$$

The attributes of arc (s^*, s) and of node s^* are defined as follows.

$$\lambda_{s^*s}(t) := 0,\ t \geq 0 \tag{5.11}$$

$$u_{s^*s}(t) := \begin{cases} q_s(0) & ,\ t = 0 \\ 0 & ,\ t > 0 \end{cases} \tag{5.12}$$

$$a_{s^*}(t) := 0,\ t \geq 0 \tag{5.13}$$

The maximum dynamic flow in the modified G is bounded above by $q_s(0)$. The problem is infeasible if $V_{\sum^T} < q_s(0)$.

Proposition 5.2 *By implementing Algorithm 4.4, no additional process is required to calculate* $T_{\sum^{q_s(0)}} := \min\{T' \ :\ V_{\sum^{T'}}(\mathbf{x}) \geq q_s(0),\ T' \leq T\}$.

Proof:
Consider the step 1 of Algorithm 4.4 which implements Algorithm 4.3. Since the label π_d denotes the earliest arrival time at the sink node d, $T_{\sum^{q_s(0)}}$ is the value of π_d at the last augmentation before the stopping condition $\pi_d > T' := T$ reached (see step 3 of Algorithm 4.3). Therefore no additional process is required to calculate $T_{\sum^{q_s(0)}}$. ■

The solution algorithm for $(s,d)/(\lambda(t),u(t),a(t))/T_{\sum^{q_s(0)}}$ that uses Algorithm 4.4 is summarized in Algorithm 5.2.

Algorithm 5.2 : Solving $(s,d)/(\lambda(t),u(t),a(t))/T_{\sum^{q_s(0)}}$

INPUT	Network $G=(N,A,T)$, time-dependent capacity $u(t)$, travel time $\lambda(t)$, holdover capacity $a(t)$, and $q_s(0)$.
OUTPUT	$T_{\sum^{q_s(0)}}$.
0	Modify the original network $G=(N,A,T)$ by adding a dummy node and a dummy arc with data according to (5.11) - (5.13). Set the time T large enough such that the problem is feasible.
1	Apply Algorithm 4.4 to find $\mathbf{x}$ the solution of $(s^*,d)/(\lambda(t),u(t),a(t))/V_{\sum^{T'\le T}}$
2	$T_{\sum^{q_s(0)}}$ is determined by the value of π_d at the last augmentation before the stopping condition $\pi_d > T$ reached.

Proposition 5.3 *Algorithm* 5.2 *solves* $(s,d)/(\lambda(t),u(t),a(t))/T_{\sum^{q_s(0)}}$ *in* $\mathcal{O}(nmq_s(0)T^2)$. *The worst-case complexity reduces to* $\mathcal{O}(nmq_s(0)T)$ *when infinite waiting is allowed for every node* $i \in N - \{d\}$.

Proof:
The correctness of the algorithm follows immediately from Theorem 5.2. Since the maximum dynamic flow is bounded above by $q_s(0)$, the number of augmentations is also bounded above by $q_s(0)$. ■

Now, consider a DTQFP with multiple sources. Let us denote by S, a set of nodes with positive initial contents, i.e.,

$$S := \{i \in N \ : \ q_i(0) > 0\}$$

A time-dependent parameters DTQFP with a set of multiple sources S is denoted by (see Appendix D)

$$(S, d)/(\lambda(t), u(t), a(t))/T_{\sum^{\nu}}$$

with ν the total initial contents of all nodes in S, i.e.,

$$\nu := \sum_{l \in S} q_l(0) \tag{5.14}$$

We add a super source node s^* and connect it to each node in S. We denote by A_S the set of additional arcs $(s^*, i),\ \forall i \in S$. The set of nodes N and the set of arcs A of the network $G = (N, A, T)$ are extended to

$$N := N \cup \{s^*\} \quad \text{and} \quad A := A \cup A_S \tag{5.15}$$

The waiting capacity of s^* is defined by (5.13). The travel times and capacities of arcs $(s^*, i) \in A_S$ are defined by

$$\lambda_{s^*i}(t) := 0,\ t \geq 0 \tag{5.16}$$

$$u_{s^*i}(t) := \begin{cases} q_i(0) & ,\ t = 0 \\ 0 & ,\ t > 0 \end{cases} \tag{5.17}$$

The modified algorithm to solve $(S, d)/(\lambda(t), u(t), a(t))/T_{\sum^{\nu}}$ is given by Algorithm 5.3.

Algorithm 5.3 : Solving $(S, d)/(\lambda(t), u(t), a(t))/T_{\sum^{\nu}}$

INPUT	Dynamic network $G = (N, A, T)$ with time-dependent capacity $u_{ij}(t)$, travel time $\lambda_{ij}(t)$, holdover capacity $a(t)$, and the set S of nodes with positive initial contents.
OUTPUT	$T_{\sum^{\nu}}$.
0	Modify the original network $G = (N, A, T)$ by adding dummy node and dummy arc with data according to (5.13) and (5.16) - (5.17). Set the time T large enough such that the problem is feasible.
1	Apply Algorithm 4.4 to find the solution $\mathbf{x}$ of $(s^*, d)/(\lambda(t), u(t), a(t))/V_{\sum^{T' \leq T}}$
2	$T_{\sum^{\nu}}$ is determined by the value of π_d at the last augmentation before the stopping condition $\pi_d > T$ reached.

Proposition 5.4 *Algorithm* 5.3 *solves*

$$(S, d)/(\lambda(t), u(t), a(t))/T_{\sum^\nu}$$

in $\mathcal{O}(nmT^2\nu)$. *The complexity reduces to* $\mathcal{O}(nmT\nu)$ *when infinite waiting is allowed for every node in* N.

Example 5.1
Consider a network of Example 3.3 shown in Figure 3.7. We assume that $q_0(0) = 10$ and $q_1(0) = 4$, no initial supply at other nodes. Therefore, we define $S := \{0, 1\}$. The modified network is shown in Figure 5.1.

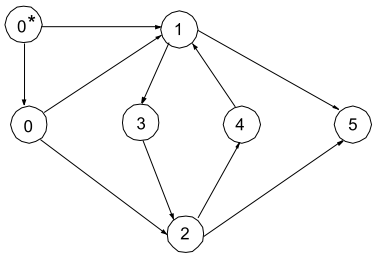


Figure 5.1: A Network for Example 5.1

The maximum dynamic flow is given in Table 5.1. The value of maximum dynamic flow is

$$V_{\sum^{T=7}} = 14$$

The minimum total time needed to clear the network is $T_{\sum^{14}} = 7$. □

$x_{ij}(t)$	time t							
	0	1	2	3	4	5	6	7
$x_{0^*0}(t)$	10	0	0	0	0	0	0	0
$x_{0^*1}(t)$	4	0	0	0	0	0	0	0
$x_{01}(t)$	0	0	0	0	0	0	0	0
$x_{02}(t)$	10	0	0	0	0	0	0	0
$x_{13}(t)$	0	0	0	0	0	0	0	0
$x_{15}(t)$	4	0	0	0	0	0	0	0
$x_{24}(t)$	0	0	0	0	0	0	0	0
$x_{25}(t)$	0	0	0	0	0	5	5	0
$x_{32}(t)$	0	0	0	0	0	0	0	0
$x_{41}(t)$	0	0	0	0	0	0	0	0

Table 5.1: The optimal dynamic flow of $(s^*, d)/(\lambda(t), u(t), a(t))/V_{\sum^{T' \le T}}$ of Example 5.1

5.4 Quickest Flow Problem with Time-Dependent Supplies

Consider now a more general time-dependent attributes DTQFP by considering $q(t)$ as the supply to the source s which may vary over time. We define the supply function at any node $i \in N$ and any time t as follows.

$$q_i(t) \begin{cases} \geq 0 & , i = s \\ = 0 & , i \neq s \end{cases} , \ t \in \{0, \ldots, T\}$$

Here, the objective function is to minimize the total time needed to send all supplies from the source to the sink. The formulation of this problem is given by

$$\text{(DTQFPqt)} \quad \min \qquad T \tag{5.18}$$

Subject to

$$\sum_{(j,i)\in A} \sum_{\{t':t'+\lambda_{ji}(t')=t\}} x_{ji}(t') - \sum_{(i,j)\in A} x_{ij}(t) = x_{ii}(t) - x_{ii}(t-1) - q_i(t), \quad \forall i \in N - \{d\}, \ t = 0, \ldots, T \tag{5.19}$$

$$\text{and constraints} \qquad (5.3) - (5.6)$$

Let us denote by Q_T the total supply of node s for the time horizon T, i.e.,

$$Q_T := \sum_{t=0}^{T} q_s(t) \tag{5.20}$$

The travel time of a discrete-time quickest flow (i.e., the minimum clearing time), $T_{\sum^{q(t)}}$, of a given time-dependent supplies $q(t)$ to the source, is given by

$$T_{\sum^{q(t)}} = \min\{T \ : \ V_{\sum^T} \geq Q_T\} \tag{5.21}$$

We denote this problem by (see Appendix D)

$$(s, d)/(\lambda(t), u(t), a(t))/T_{\sum^{q(t)}}$$

We can modify the network G and use again Algorithm 4.4 to solve this problem. The network modification of $G = (N, A, T)$ is done in a similar way as in the previous section. A super source s^* and an arc (s^*, s) is added to N and A, respectively, i.e., $N := N \cup \{s^*\}$; $A := A \cup \{(s^*, s)\}$. The parameters of arc (s^*, s) and of node s^* are defined as follows.

$$\lambda_{s^*s}(t) := 0, \ t \geq 0 \tag{5.22}$$

$$u_{s^*s}(t) := q_s(t), \ t \geq 0 \tag{5.23}$$

$$a_{s^*}(t) := 0, \ t \geq 0 \tag{5.24}$$

Algorithm 5.4 : Solving $(s,d)/(\lambda(t),u(t),a(t))/T_{\sum^{q(t)}}$

INPUT	Dynamic network $G=(N,A,T)$, time-dependent capacity $u_{ij}(t)$, travel time $\lambda_{ij}(t)$, holdover capacity $a(t)$, and the supply $q(t)$.
OUTPUT	$T_{\sum^{q(t)}}$.
0	Modify the original network $G=(N,A,T)$ by adding a dummy node and a dummy arc with data according to (5.22) - (5.24). Set the time T large enough such that the problem is feasible.
1	Apply Algorithm 4.4 to find $\mathbf{x}$ the solution of $(s^*,d)/(\lambda(t),u(t),a(t))/V_{\sum^{T'\leq T}}$
2	$T_{\sum^{q(t)}}$ is determined by the value of π_d at the last augmentation before the stopping condition $\pi_d > T$ reached.

An algorithm to solve $(s,d)/(\lambda(t),u(t),a(t))/T_{\sum^{q(t)}}$ is given in Algorithm 5.4.

Since the total supply of node s for the time horizon T is Q_T, the maximum dynamic flow is bounded above by Q_T. The complexity of Algorithm 5.4 is thus $\mathcal{O}(nmT^2Q_T)$, and $\mathcal{O}(nmTQ_T)$ when inifinite waiting for every node is allowed.

Remark 5.1 *By a suitable network modification, the solution algorithm can be easily extended to solve the multi source problem with time-dependent supplies.*

Chapter 6

Time-Dependent Bicriteria Dynamic Shortest Path Problems

Shortest path problems are one of the best studied network optimization problems (see e.g., Bertsekas [PB91] and Ahuja, Magnanti, and Orlin [AMO93]) with many applications, such as in transportations (see e.g., Ziliaskopoulos and Mahmassani [ZM93], and Chabini [Cha98]) and evacuation planning (see e.g., Kostreva and Wiecek [KW93], Yamada [Yam96] and Getachew, Kostreva, and Lancaster [GKL00]). In this chapter we consider time-dependent bicriteria dynamic shortest path problems (TdBiDSP) which differ from the classical shortest path problems in two ways

- two, generally contradicting, objective functions instead of a single one
- time dependency of travel times of arcs, waiting at nodes, and cost function

TdBiDSP is a valuable tool in several of the applications mentioned above, since most of these processes are time dependent and more than one set of time dependent cost attributes are to be taken into account. Our main motivation is the application in evacuation modelling, where shortest paths represent evacuation routes. Obviously, these routes are in case of emergeny subject to changes over time (expressable as time varying costs and travel times) and different types of cost representing for instance speed or reliability may be of interest. A set of possible evacuation routes may also be put together to a dynamic flow which can be used to model a complete evacuation plan. Paths and cycles play in this context the role of building up and improving existing plans, respectively. In this situation cost may be negative as well as positive, a situation which will be analyzed later in this chapter.

Without time dependency, there are several papers dealing with multicriteria shortest path problems. Hansen [Han80] introduced ten different types of bicriteria path problems. He also showed that one can construct a specific static network in which the number of Pareto optimal paths grows exponentially with the number of nodes in the network. Martins [Mar84] developed a multiple labeling version of Djikstra's label setting algorithm to generate all Pareto shortest paths from the source node to every other node in the network

(i.e., *one-to-all* shortest paths). Corley and Moon [CM85] used dynamic programming to solve the multicriteria shortest path problem. Brumbaugh-Smith and Shier [BSS89] proposed a label correcting with a multiple labeling for the bicriteria shortest path problem. They also proposed a linear time algorithm for merging two Pareto optimal sets.

On the other hand, there are various papers, dealing with time dependency, but with a single objective function. Here, time-dependent dynamic shortest path problems can be classified into fastest path and minimum cost dynamic path problems. In a fastest path problem, the cost of an arc is the travel time of that arc. Its objective is then to seek for paths having minimum length with respect to time-dependent travel times. A minimum cost dynamic path problem seeks for dynamic paths having minimum length with respect to the cost while considering the time needed to travel from one node to another. Hence, the fastest path problem is a particular case of the minimum cost dynamic path problem. The first paper dealing with a fastest path problem appears to be by Cooke and Halsey [CH66]. They extended Bellman's optimality principle (Bellman [Bel58]) to obtain fastest paths from every node in the network to one destination (i.e., sink) node, i.e., *all-to-one* fastest paths. The travel times on arcs are defined to have positive integer values for every time period taken from a *time-grid* $\{t_0, t_0+1, \ldots, t_0+M\}$. Here, the integer number M must be chosen in such a way that the problem is feasible. Dreyfus [Dre69] proposed a modification of Djikstra's static shortest path algorithm to calculate a fastest path between two nodes for a given departure time. His algorithm works well only for the FIFO (First-In First-Out) network (see Definition 2.4). The complexity of Dreyfus's algorithm is the same as that of Djikstra. Ziliaskopoulos and Mahmassani [ZM93] proposed label-correcting algorithms to calculate both *all-to-one* fastest paths and minimum cost dynamic paths for every possible departure times. Their algorithms do not require a FIFO assumption. Similar algorithms were proposed by Wardell and Ziliaskopoulos [WZ00]. Chabini [Cha98] proposed backward label setting algorithms in which the labeling is done in a decreasing manner with respect to the time period. His algorithms solve both fastest and minimum cost dynamic paths for all possible departure times without FIFO requirement. In a continuous-time environment, many results on time-dependent dynamic shortest paths (i.e., fastest path or minimum cost dynamic path) can be found in the literature, including the results from Orda and Rom [OR90, OR91], Philpott and Mees [PM93], and Philpott [Phi94]).

In the literature, multicriteria dynamic shortest path problems, as combination of the above two problem classes, have attained relatively little attention. Kostreva and Wiecek [KW93] generalized the result of Cooke and Halsey [CH66] to the multicriteria case. The cost function of an arc is assumed to be positive vector valued function of time and may be discontinuous. A backward dynamic programming is developed to generate all Pareto minimum cost dynamic paths leading from every node in the network to the sink node. A forward dynamic programming is also discussed by these authors to generate all Pareto minimum cost dynamic paths from the source to every other nodes in the network under FIFO and nondecreasing arc cost assumptions. Getachew, Kostreva, and Lancaster

[GKL00] generalized the results of Kostreva and Wiecek by replacing their nondecreasing arc cost assumption by lower and upper bounds of the cost (for the forward dynamic programming) and by relaxing the time grid assumption. To the best of our knowledge, these two papers are the only ones which have been published in the area of multicriteria time-dependent shortest path problems.

In this chapter we will develop two algorithms solving TdBiDSP. While the first algorithm deals only with the nonnegative attributes, the second one allows arcs of the network to have negative travel times and costs. The possibility to wait (or park) at a node before departing on an outgoing arc, in order to keep the total cost low, is also taken into account. In the next section, we formally introduce TdBiDSP. In Section 6.2 we classify the problem as an NP-hard problem, show that Belman's optimality principle holds for the all-to-one problem, but not the one-to-all problem and introduce the concept of negative dynamic cycles. Section 6.3 deals with the necessary data structures for the labels, which take into consideration the time and bicriteria environment. In Section 6.4 we assume nonnegative attributes, work out details of the algorithm of Kostreva and Wiecek [KW93] and present a new, label setting algorithm. The non-negativity assumption is dropped in the subsequent Section 6.5, where a label correcting algorithm is developed which can be used to detect negative dynamic cycles, or solve TdBiDSP if such cycles do not exist. Computational results on several examples based on randomly generated networks in Section 6.6 conclude this chapter.

6.1 Problem Formulation

A discrete-time dependent network $G = (N, A, T)$ is a directed graph, where N is a set of nodes, A is a set of directed arcs, and T is a finite time horizon of interest discretized into the set $\{0, \dots, T\}$. We denote by n and m the cardinality of N and A, respectively. Moreover, it is assumed that G has a single sink d. Each arc (i, j) carries a travel time function $\lambda_{ij}(t)$ with

$$\lambda : A \times \{0, \dots, T\} \to \mathbb{Z}$$

and a cost (vector) function $\begin{pmatrix} c^1 \\ c^2 \end{pmatrix}$ of two criteria components with

$$c^k : A \times \{0, \dots, T\} \to \mathbb{R}, \quad k = 1, 2$$

where the superscript k corresponds to the cost function of criteria k. The travel time $\lambda_{ij}(t)$ follows the frozen arc model (see Section 2.1).

We denote by $w_i(t)$ the maximum allowable waiting time at node i starting at time t. Here, we assume that the maximum waiting time at every node and at every time has a *memory-less property*, i.e., the maximum waiting time is decided by the current time, not

$$w_i(t) := 3, w_i(t+1) := 3, w_i(t+2) := 0, w_i(t+3) := 1, w_i(t+4) := 1$$
$$w_j(t) := 0, w_j(t+1) := 1, w_j(t+2) := 3, w_j(t+3) := 0, w_j(t+4) := 0$$

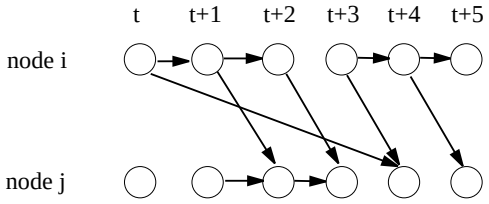


Figure 6.1: Memory-less property of maximum waiting time

by the earlier time or even the arrival time at the node. Figure 6.1 illustrates the memory-less property of a maximum waiting time. This memory-less property implies that it is enough for $w_i(t)$ to have value either zero or one, i.e.,

$$w_i(t) \in \{0, 1\},\ i \in N,\ t \in \{0, \ldots, T\}$$

Furthermore, a time-dependent holding cost attribute $h_i(t)$ can be associated with the node i to define a penalty for waiting one time unit at node i at time t.

We define a dynamic path and a dynamic cycle, respectively, as follows.

Definition 6.1 (Dynamic path) *A dynamic $j_1 - j_l$ path $P_{j_1 j_l}(t_1)$ is a sequence of node-time pairs (NTPs) from the node j_1 to the node j_l that is ready at node j_1 at time $t_1 \in \{0, \ldots, T\}$, as given by*

$$P_{j_1 j_l}(t_1) = \{j_1(t_1, t_1^{'}), \ldots, j_q(t_q, t_q^{'}), \ldots, j_l(t_l, t_l^{'})\}, \quad t_q, t_q^{'} \in \{0, \ldots, T\}, \quad q = 1, \ldots, l \tag{6.1}$$

where $t_{q+1} = t_q^{'} + \lambda_{j_q j_{q+1}}(t_q^{'}),\ q = 1, \ldots, l-1$ and define $t_l = t_l^{'}$.
For any NTP $j_q(t_q, t_q^{'})$, the first time parameter t_q denotes the ready time at node j_q, and the second one $t_q^{'}$ denotes the departure time from node j_q. A ready time t_q also defines the arrival time at node j_q from the previous node j_{q-1}, for $q = 2, \ldots, l$.

Figure 6.1 illustrates Definition 6.1.

If $t \in \{0, \ldots, T\}$ is a ready time of node i and $t^{'}$ is a departing time from node i with $t^{'} \geq t$, then the corresponding total waiting cost at node i, denoted by $H_i(t, t^{'})$, is given by

$$H_i(t, t^{'}) = \sum_{t^{''}=t+1}^{t^{'}} h_i(t^{''})$$

We denote by $\mathbb{P}_{j_1 j_l}(t_1)$ the set of all paths from node j_1 with ready time $t_1 \in \{0, \ldots, T\}$ and reach the node j_l within the time horizon T. We also denote the union of $\mathbb{P}_{j_1 j_l}(t_1)$ for all $t_1 \in \{0, \ldots, T\}$ by $\mathbb{P}_{j_1 j_l}$.

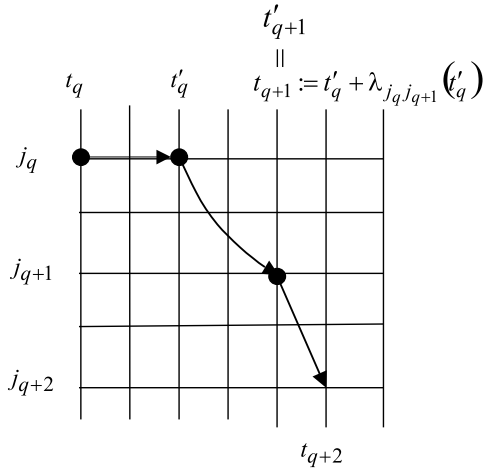


Figure 6.2: Dynamic path on the time-expanded graph

Definition 6.2 *The length of a dynamic path $P_{j_1 j_l}(t_1)$ with respect to the travel times and waiting times is given by*

$$\lambda(P_{j_1 j_l}(t_1)) := t'_l - t_1 \tag{6.2}$$

Definition 6.3 (Dynamic cycle) *A dynamic $j_1 - j_l$ path $P_{j_1 j_l}(t_1)$ defined by (6.1) is a dynamic cycle if $j_1 = j_l$, $t_1 = t_l$ and $t'_1 = t'_l$.*

It should be noted that a dynamic cycle may exist, since negative travel times are allowed. Moreover, a dynamic $j_1 - j_l$ path $P_{j_1 j_l}(t_1)$ is called *simple* if it does not contain any dynamic cycle.

To define the cost of a path, it is useful to define the following operation. For $\begin{pmatrix} g^1 \\ g^2 \end{pmatrix} \in \mathbb{R}^2$ and $e \in \mathbb{R}$ define

$$\begin{pmatrix} g^1 \\ g^2 \end{pmatrix} \oplus e = \begin{pmatrix} g^1 + e \\ g^2 + e \end{pmatrix} \tag{6.3}$$

Let $c : \mathbb{P}_{j_1 j_l} \to \mathbb{R}^2$. The total cost of a $j_1 - j_l$ path $P_{j_1 j_l}(t_1) \in \mathbb{P}_{j_1 j_l}(t_1)$ is defined by

$$c(P_{j_1 j_l}(t_1)) := \begin{pmatrix} c^1(P_{j_1 j_l}(t_1)) \\ c^2(P_{j_1 j_l}(t_1)) \end{pmatrix} = \sum_{q=1}^{l-1} \begin{pmatrix} c^1_{j_q j_{q+1}}(t'_q) \\ c^2_{j_q j_{q+1}}(t'_q) \end{pmatrix} \oplus H_{j_q}(t_q, t'_q) \tag{6.4}$$

Using this cost definition, we give a definition of a negative dynamic cycle as follows.

Definition 6.4 (Negative dynamic cycle) *A dynamic cycle $P_{j_1 j_l}(t_1)$ is a negative dynamic cycle if at least one component of $c(P_{j_1 j_l}(t_1))$ has a negative value.*

For every node $i \in N$ and every ready time $t \in \{0, \ldots, T\}$, we want to find a shortest dynamic path from i to a single sink d with respect to the cost vector $\mathbf{c}$ and the holding cost $\mathbf{h}$. To derive the linear programming formulation of this problem we define the following objective function $\mathbf{c}^k_{\sum^T}$ as

$$\mathbf{c}^k_{\sum^T} := \sum_{t=0}^{T} \sum_{(i,j)\in A} \sum_{\{t':t'+\lambda_{ij}(t')=t\}} c^k_{ij}(t')x_{ij}(t') \quad + \quad \sum_{t=0}^{T} \sum_{i\in N-\{d\}} x_{ii}(t)h_i(t), \; k = 1,2 \tag{6.5}$$

The supply vector $\mathbf{q}$ is defined by

$$q_i(t) := \begin{cases} 1 & , \; i \in N - \{d\}, \; t \in \{0, \ldots, T\} \\ -(n-1)(T+1) & , \; i = d, \; t = T \\ 0 & , \; \text{otherwise} \end{cases}$$

The linear programming formulation of TdBiDSP is given as follows.

$$\text{(TdBiDSP)} \qquad \min \qquad \mathbf{c}^k_{\sum^T}, \; \forall k = 1,2 \tag{6.6}$$

Subject to

$$\sum_{(j,i)\in A} \sum_{\{t':t'+\lambda_{ji}(t')=t\}} x_{ji}(t') - \sum_{\{(i,j) \,:\, (i,j)\in A, \; t+\lambda_{ij}(t)\le T\}} x_{ij}(t) = x_{ii}(t) - x_{ii}(t-1) - q_i(t), \quad i \in N, \; t = 0, \ldots, T \tag{6.7}$$

$$0 \le x_{ii}(t) \le w_i(t)M, \; i \in N - \{d\}, \; t = 0, \ldots, T \tag{6.8}$$

In (6.8), it is assumed that $a_i(t) = M$ for every node $i \in N - \{d\}$ and for every time $t \in \{0, \ldots, T\}$ in which M is a very big number. Therefore if $w_i(t) = 1$, then the holdover flow $x_{ii}(t)$ is unbounded. Moreover, it is assumed that the arc capacity $u_{ij}(t) = \infty$ for every arc $(i,j) \in A$ and for every time $t \in \{0, \ldots, T\}$.

In general, there exists no dynamic flow that simultaneously minimizes $\mathbf{c}^k_{\sum^T}$ for both $k = 1,2$, i.e., no dynamic path $P_{id}(t)$ that simultaneously minimizes $c^k(P_{id}(t))$ for both $k = 1,2$. Nevertheless a privileged set of paths called the set of *Pareto optimal* (*efficient* or *nondominated*) paths can be determined. To describe a Pareto optimal path, we need the notion of a vector ordering.

Definition 6.5 (Vector ordering) *Let* $z_1 = \begin{pmatrix} z^1_1 \\ z^2_1 \end{pmatrix}$ *and* $z_2 = \begin{pmatrix} z^1_2 \\ z^2_2 \end{pmatrix}$ *be two vectors in* $\mathbb{R}^2$.

- $z_1 = z_2$ *if and only if* $z_1^k = z_2^k$ *for all* $k = 1, 2$.
- $z_1 \leq z_2$ *if and only if* $z_1^k \leq z_2^k$ *for all* $k = 1, 2$.
- $z_1 < z_2$ *if and only if* $z_1^k \leq z_2^k$ *for all* $k = 1, 2$ *and there exists* $k \in \{1, 2\}$ *such that* $z_1^k < z_2^k$.

Definition 6.6 (Pareto optimal path and minimal complete set)

(a) *Let* $P_{id}(t)$ *and* $P'_{id}(t)$ *be two paths in* $\mathbb{P}_{id}(t)$. *We say* $P_{id}(t)$ *dominates* $P'_{id}(t)$ *if and only if*

$$c(P_{id}(t)) < c(P'_{id}(t))$$

(b) *A path* $P_{id}(t) \in \mathbb{P}_{id}(t)$ *is called Pareto optimal (PO) or nondominated if there is no other path* $P'_{id}(t) \in \mathbb{P}_{id}(t)$ *dominating* $P_{id}(t)$.

(c) *Two PO-paths* $P_{id}(t)$ *and* $P'_{id}(t)$ *in* $\mathbb{P}_{id}(t)$ *are said equivalent if and only if* $c(P'_{id}(t)) = c(P_{id}(t))$. *Otherwise, they are said nonequivalent.*

(d) *A complete set* $\overline{PO}(\mathbb{P}_{id}(t)) \subset \mathbb{P}_{id}(t)$ *of PO-paths is a set such that any path* $P_{id}(t) \notin \overline{PO}(\mathbb{P}_{id}(t))$ *is either dominated or equivalent to at least one PO-path* $P'_{id}(t) \in \overline{PO}(\mathbb{P}_{id}(t))$.

(e) *A complete set is minimal if and only if it does not have two equivalent PO-paths. We then denote by* $PO(\mathbb{P}_{id}(t))$ *the minimal complete set of all Pareto optimal paths from node* i *to the sink node* d *with ready time* $t \in \{0, \ldots, T\}$ *at node* i.

We denote the problem of finding $PO(\mathbb{P}_{id}(t))$ for every node $i \in N$ (*all-to-one*) and for every ready time $t \in \{0, \ldots, T\}$ by

$$(i, d)/(\lambda(t), w(t), c(t), h(t))/PO(\mathbb{P})$$

Our objective is to develop some algorithms to solve this problem.

6.2 Problem Characteristics

6.2.1 Complexity

Looking at the problem complexity, as in the case of bicriterion static path problem (see Hansen [Han80]), Example 6.1 shows a network with time dependent attributes in which the number of PO-paths grows exponentially with the number of nodes in the network. Therefore, in the worst case, this problem is intractable and any algorithm for generating all Pareto dynamic paths is thus doomed to an exponential computation time.

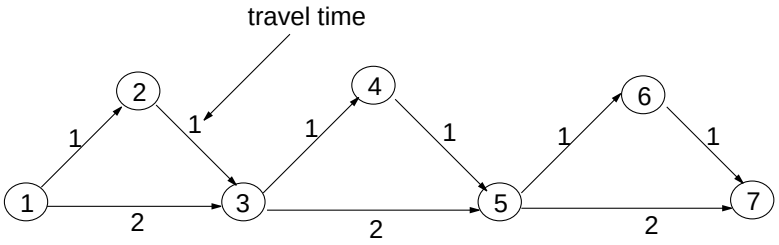


Figure 6.3: Example network with exponential number of Pareto optimal paths

Example 6.1
Consider a network shown in Figure 6.3.
We assign the following two cost criteria to each arc of the network.

$$c_{ij}(t) = \begin{cases} \begin{pmatrix} 2^i(t+1) \\ 2^{i-1}(t+1) \end{pmatrix} & \text{, if } j = i+1 \text{ and } i \text{ is odd} \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{, if } j = i+1 \text{ and } i \text{ is even} \\ \begin{pmatrix} 2^{i-1}(t+1) \\ 2^i(t+1) \end{pmatrix} & \text{, if } j = i+2 \text{ and } i \text{ is odd} \end{cases} \quad ; \; t \in \{0, \ldots, T\}$$

Let us assume that $w_i(t) = 0$ for every node $i \in N$ and every time $t \in \{0, \ldots, T\}$, i.e., no waiting is allowed for every node. By this cost definition, there are $2^{\frac{n-1}{2}} = 8$ PO-paths from node 1 to node 7 for every ready time $0 \leq t_0 \leq T - 6$, as shown in Table 6.1. □

6.2.2 Principle of Optimality

In the case of static shortest path problems, the principle of optimality says that the subsets of an optimal solution are also optimal (see e.g., Bellman [Bel58]). Concerning the multicriteria dynamic shortest path problem, we formalize the principle optimality as follows.

Definition 6.7 (Principle of optimality in a forward direction)
Let $P_{sj}(t) \in PO(\mathbb{P}_{sj}(t))$. The forward principle of optimality holds for $P_{sj}(t)$ if and only if for each intermediate node i of $P_{sj}(t)$, the sub-path $P_{si}(t)$ is also in $PO(\mathbb{P}_{si}(t))$.

Definition 6.8 (Principle of optimality in a backward direction)
Let $P_{id}(t) \in PO(\mathbb{P}_{id}(t))$. The backward principle of optimality holds for $P_{id}(t)$ if and only if for each intermediate NTP $j(t_j, t_j')$ of $P_{id}(t)$, the sub-path $P_{jd}(t_j)$ is also in $PO(\mathbb{P}_{jd}(t_j))$.

When waiting at the nodes is allowed, a similar FIFO property can be imposed on arc costs $\mathbf{c}$.

$P_{17}(t_0),\ 0 \le t_0 \le T-6$	$c(P_{17}(t_0))$
$\{1(t_0,t_0), 2(t_0+1,t_0+1), 3(t_0+2,t_0+2), 4(t_0+3,t_0+3),$ $5(t_0+4,t_0+4), 6(t_0+5,t_0+5)\}, 7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 186+42t_0 \\ 93+21t_0 \end{pmatrix}$
$\{1(t_0,t_0), 2(t_0+1,t_0+1), 3(t_0+2,t_0+2),$ $4(t_0+3,t_0+3), 5(t_0+4,t_0+4), 7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 106+26t_0 \\ 173+37t_0 \end{pmatrix}$
$\{1(t_0,t_0), 2(t_0+1,t_0+1), 3(t_0+2,t_0+2),$ $5(t_0+4,t_0+4), 7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 94+22t_0 \\ 185+41t_0 \end{pmatrix}$
$\{1(t_0,t_0), 2(t_0+1,t_0+1), 3(t_0+2,t_0+2),$ $5(t_0+4,t_0+4), 6(t_0+5,t_0+5), 7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 174+38t_0 \\ 105+25t_0 \end{pmatrix}$
$\{1(t_0,t_0), 3(t_0+2,t_0+2), 5(t_0+4,t_0+4),$ $6(t_0+5,t_0+5), 7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 173+37t_0 \\ 106+26t_0 \end{pmatrix}$
$\{1(t_0,t_0), 3(t_0+2,t_0+2), 5(t_0+4,t_0+4),$ $7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 93+21t_0 \\ 186+42t_0 \end{pmatrix}$
$\{1(t_0,t_0), 3(t_0+2,t_0+2), 4(t_0+3,t_0+3),$ $5(t_0+4,t_0+4), 7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 105+25t_0 \\ 174+38t_0 \end{pmatrix}$
$\{1(t_0,t_0), 3(t_0+2,t_0+2), 4(t_0+3,t_0+3),$ $5(t_0+4,t_0+4), 6(t_0+5,t_0+5), 7(t_0+6,t_0+6)\}$	$\begin{pmatrix} 185+41t_0 \\ 94+22t_0 \end{pmatrix}$

Table 6.1: Pareto optimal paths and their associated values of Example 6.1

Definition 6.9 (e.g., Pallottino and Scutella [PS97]) *An arc (i,j) is said to have a Cost Consistency (CC) property if leaving node i earlier along (i,j) does not cost more than leaving i later, i.e.,*

$$c_{ij}^k(t) \le c_{ij}^k(t') + \sum_{t''=t+1}^{t'} h_i(t''-1),\ \forall t < t',\ k = 1,2$$

A dynamic network $G=(N,A,T)$ is said to be a CC network when all of its arcs have CC property.

The following theorem shows that the backward principle of optimality holds, even if FIFO or cost consistency assumptions do not hold.

Theorem 6.1 (Kostreva and Wiecek [KW93]) *Let $G=(N,A,T)$ be a dynamic network with nonnegative costs and travel times. The backward principle of optimality holds for any Pareto optimal path in G.*

Proof:
Let $P_{id}(t) \in PO(\mathbb{P}_{id}(t))$ with $j(t_j, t_j^{'}) \in P_{id}(t)$. Assume that $P_{jd}(t_j) \subset P_{id}(t)$ is not a Pareto optimal, i.e., there exists some path $P_{jd}^{'}(t_j)$ such that

$$c(P_{jd}^{'}(t_j)) < c(P_{jd}(t_j)) \tag{6.9}$$

Let $P_{ij}(t)$ be the subpath of $P_{id}(t)$. By concatenating $P_{ij}(t)$ with $P_{jd}(t_j)$ and $P_{jd}^{'}(t_j)$, we obtain two paths $P_{id}(t)$ and $P_{id}^{'}(t)$ such that their total cost are

$$c(P_{id}(t)) = c(P_{ij}(t)) + c(P_{jd}(t_j))$$

and

$$c(P_{id}^{'}(t)) = c(P_{ij}(t)) + c(P_{jd}^{'}(t_j))$$

, respectively. Condition (6.9) implies

$$c(P_{id}^{'}(t)) < c(P_{id}(t))$$

contradicting the fact that $P_{id}(t)$ is a Pareto optimal path. Hence, $P_{jd}(t_j)$ must be a Pareto optimal path. ■

In contrast, one can build a dynamic network, even if FIFO and CC properties are satisfied, such that the forward principle of optimality does not hold.

Example 6.2
Consider a dynamic network $G = (N, A, T = 8)$ shown in Figure 6.4. Node 0 is the source node and node 3 is the sink node. Clearly each arc in G has a FIFO property and every cost component has a CC property. The path $P_{03}(0) = \{0(0,0), 2(1,1), 3(2,2)\}$ is a Pareto optimal path from node 0 to node 3 with ready time $t = 0$. Its total cost is

$$c(P_{03}(0)) = \begin{pmatrix} 9 \\ 11 \end{pmatrix}$$

This path dominates $P_{03}^{'}(0) = \{0(0,0), 1(3,3), 2(4,4), 3(5,5)\}$ which has total cost

$$c(P_{03}^{'}(0)) = \begin{pmatrix} 12 \\ 15 \end{pmatrix}$$

The path $P_{02}^{'}(0) = \{0(0,0), 1(3,3), 2(4,4)\}$ is a Pareto optimal path from node 0 to node 2 with ready time $t = 0$. Its total cost is

$$c(P_{02}^{'}(0)) = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

This path dominates $P_{02}(0) = \{0(0,0), 2(1,1)\}$ which has total cost

$$c(P_{02}(0)) = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

We see that $P_{02}(0)$ is a subset of a Pareto optimal path $P_{03}(0)$, but it is dominated. Hence, the forward principle of optimality does not hold. □

$$\lambda_{01}(t) = 3,\ t \geq 0\ ;\ \lambda_{02}(t) = \lambda_{12}(t) = \lambda_{23}(t) = 1,\ t \geq 0$$
$$w_i(t) = 0,\ i \in N,\ t \geq 0$$
$$c_{01}(t) = c_{12}(t) = \begin{pmatrix} 1 \\ 2 \end{pmatrix},\ t \geq 0$$
$$c_{02}(t) = \begin{pmatrix} 5 \\ 6 \end{pmatrix},\ t \geq 0$$
$$c_{23}(t) = \begin{pmatrix} 2t+2 \\ 2t+3 \end{pmatrix},\ t \geq 0$$

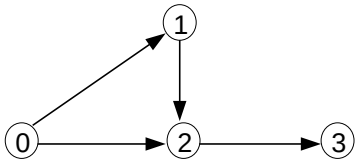


Figure 6.4: A network for Example 6.2

If the principle of optimality is not valid, optimal subpaths as partial results can not be used to obtain the optimal paths. Therefore, one cannot use the forward label setting-type algorithm to find Pareto optimal paths. However, Theorem 6.1 guarantees that we can use the label setting principles in a backward direction to find the *all-to-one* Pareto optimal paths. This will be worked out in more detail in Section 6.4.

6.2.3 Negative Dynamic Cycle

We allow in this subsection positive as well as negative travel times. In case of positive travel times, a dynamic path may pass through the same node several times but with increasing arrival times (i.e., the path must pass through different node-time pairs). Therefore, there will be no negative dynamic cycle. Furthermore, since T is finite, the cost of this path is also finite, disregarding the fact that the cost may also be negative. But, if the network has negative travel times, then there may exist some negative dynamic cycles within T. The following example illustrates these phenomena.

Example 6.3
Consider a network shown in Figure 6.5.
Let $T = 17$ and assume that no waiting is allowed. Consider a dynamic path $P_{17}(0)$ from node 1 to node 7 with ready time $t_0 = 0$.

$$\begin{aligned} P_{17}(0) \ = \ & \{1(0,0), 2(1,1), 4(4,4), 6(5,5), 5(6,6), 2(7,7), 4(10,10), \\ & 6(11,11), 5(12,12), 2(13,13), 4(16,16), 7(17,17)\} \end{aligned}$$

The total cost of path $P_{17}(0)$ is

$$c(P_{17}(0)) = \begin{pmatrix} -15 \\ -5 \end{pmatrix}$$

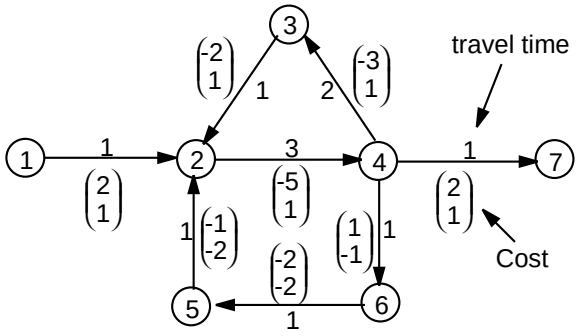


Figure 6.5: A network for Example 6.3

This path is a PO-path and passes through the same nodes 2 and 4 three times, nodes 5 and 6 two times, with different departure and arrival times. Hence, $P_{17}(0)$ does not contain any dynamic cycle. Other Pareto optimal paths in $\mathbb{P}_{17}(0)$ are

$$\begin{aligned} P'_{17}(0) &= \{1(0,0), 2(1,1), 4(4,4), 3(6,6), 2(7,7), 4(10,10), 3(12,12), \\ &\quad 2(13,13), 4(16,16), 7(17,17)\} \end{aligned}$$

with value $c(P'_{17}(0)) = \begin{pmatrix} -21 \\ 9 \end{pmatrix}$ and

$$\begin{aligned} P''_{17}(0) &= \{1(0,0), 2(1,1), 4(4,4), 3(6,6), 2(7,7), 4(10,10), 6(11,11), \\ &\quad 5(12,12), 2(13,13), 4(16,16), 7(17,17)\} \end{aligned}$$

with value $c(P''_{17}(0)) = \begin{pmatrix} -18 \\ 2 \end{pmatrix}$.

Now, if we change the travel time of arcs $(4,6), (6,5), (5,2)$ to -1, then for any t with $t, t+3 \in \{0, \ldots, T\}$ we will obtain the following negative dynamic cycle

$$\{2(t,t), 4(t+3,t+3), 6(t+2,t+2), 5(t+1,t+1), 2(t,t)\}$$

with cost $\begin{pmatrix} -7 \\ -4 \end{pmatrix} < \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $\square$

6.3 Label Structure

We denote by $\pi_i(t)$ a label associated with node i at time t with the following components (see Figure 6.6) :

- a tuple of reals π^1, π^2, corresponding to the total cost with respect to the two cost criteria,

- a tuple of integers t_{ready}, t_{depart}, corresponding to the ready time and departure time, respectively,
- a pointer *succ_ptr* points to the Pareto optimal label of the successor node.

These tuple of reals, tuple of integers, and pointer, are then called the *cost component*, *time component*, and *successor pointer*, respectively, of the label $\pi_i(t)$. Hence, $\pi_i(t)$ is represented by

$$\pi_i(t) := \left(\begin{pmatrix} \pi^1 \\ \pi^2 \end{pmatrix}; (t_{ready}, t_{depart}); succ\_ptr \right) \tag{6.10}$$

We denote the cost component of label $\pi_i(t)$ by $cost(\pi_i(t))$. The need to store the time components makes the structure of a label $\pi_i(t)$ more complex than that of the static path problem.

We denote the set of labels of node i at time t by $\Pi_i(t)$. This set is represented as a singly linked list having $\mid \Pi_i(t) \mid$ cells and each cell corresponds to the label $\pi_i(t) \in \Pi_i(t)$. Each cell will have six fields as the amount of information we wish to store. Five data fields will store the five components of $\pi_i(t)$ and the sixth field will store a pointer to the next cell in the list. We denote by *next_ptr* the pointer to the next cell in the list. If a cell happens to be the last cell in the list, by convention we set its link to $\emptyset$.

Example 6.4
Consider Example 6.3. Figure 6.6 shows a representation of $\Pi_6(11)$. □

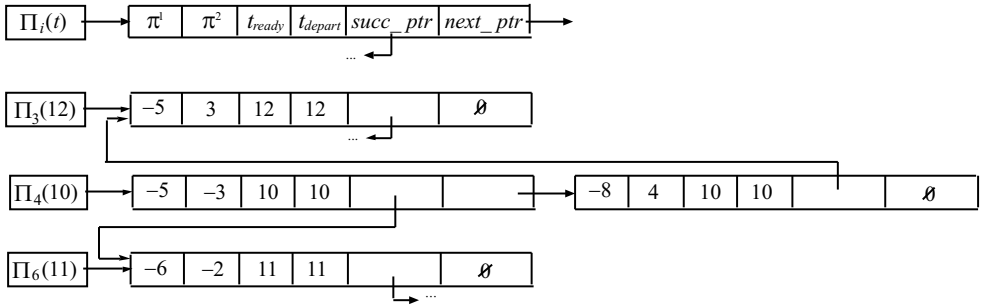


Figure 6.6: Label structure

6.4 Algorithms for All Ready Times with Nonnegative Attributes

In this section, we first discuss the algorithm of Kostreva and Wiecek [KW93]. Next, we will discuss our proposed backward label setting algorithm. With nonnegative attributes, we mean the cost components may have zero or positive value, but the travel time must be positive.

6.4.1 Backward Dynamic Programming

Kostreva and Wiecek [KW93] developed a backward dynamic programming approach to generate all Pareto optimal paths leading from every node in the network to the sink node. In order to update the set of Pareto labels of node i at time $t_{ready} \in \{0, \ldots, T\}$, the algorithm considers the labels of all node j with $(i,j) \in A$ at every $t_{arrival} \in \{0, \ldots, T\}$ in which $t_{arrival} = t_{depart} + \lambda_{ij}(t_{depart})$. The value of t_{depart} is determined by the value of waiting time at node i starting at t_{ready}. The backward dynamic programming in [KW93] does not consider the waiting time, but it can be easily extended to that case. Here, instead of modeling the waiting by a self-loop as suggested by the authors, we consider it directly in the labeling process. Since it is assumed that the maximum waiting time has a memory-less property, the value of t_{depart} runs over all possible departure times in the set $Depart_i(t_{ready})$ defined by

$$\begin{aligned} Depart_i(t_{ready}) \;:=\; & \{t_{ready}\} \cup \\ & \{t' \;:\; t_{ready}+1 \le t' \le t \le T \text{ with} \\ & \prod_{t''=t_{ready}}^{t-1} w_i(t'') > 0 \text{ and } w_i(t) = 0\} \end{aligned} \tag{6.11}$$

The backward dynamic programming approach is summarized as follows. For every set $\Pi_j(t)$ of labels, we denote by $VMIN(\Pi_j(t))$ the vector minimization over the associated cost components of all members of the label set $\Pi_j(t)$. The vector minimization process yields the set of labels with associated Pareto optimal costs. Let $\Pi_i(t_{ready})^{(l)}$ be the set of labels of all Pareto optimal $i-d$ paths with ready time t_{ready} of at most l arcs. The set $\Pi_i(t_{ready})^{(l+1)}$ for any $t_{ready} \in \{0, \ldots, T\}$ is obtained by

$$\Pi_i(t_{ready})^{(l+1)} = \begin{cases} VMIN\big(\Pi_j(t_{arrival})^{(l)} + c_{ij}(t_{depart}) \oplus \\ \qquad H_i(t_{ready}, t_{depart})\big), \; i \in N - \{d\} \\ \left\{\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} \;;\; (t_{ready}, t_{ready}); \emptyset\right)\right\}, \; i = d \end{cases} \tag{6.12}$$

where $t_{arrival} = t_{depart} + \lambda_{ij}(t_{depart})$, for all $t_{depart} \in Depart_i(t_{ready})$.
The operation

$$\Pi_j(t_{arrival})^{(l)} + c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})$$

sums the associated cost component of every label in $\Pi_j(t_{arrival})^{(l)}$ with the vector $c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})$. This operation also sets the associated time component of every label in the set $\Pi_i(t_{ready})^{(l+1)}$ to (t_{ready}, t_{depart}) and the associated successor pointer to node j. The iteration continues until $\Pi_i(t_{ready})^{(l)}$ converges to $\Pi_i(t_{ready})$ for all $i \in N$ and $t_{ready} \in \{0, \ldots, T\}$. The complete algorithm is summarized in Algorithm 6.1.

Algorithm 6.1 (Kostreva and Wiecek [KW93])

INPUT Network $G = (N, A, T)$, $w(t), c(t)$, and $h(t)$.
OUTPUT $PO(\mathbb{P}_{id}(t))$, $\forall i \in N$, $\forall t \in \{0, \ldots, T\}$.

0 Modify the cost vector $c_{ij}(t)$, $\forall t \in \{0, \ldots, T\}$ as follows.

$$c_{ij}(t) = \begin{cases} \begin{pmatrix} c_{ij}^1(t) \\ c_{ij}^2(t) \end{pmatrix} & , t + \lambda_{ij}(t) \leq T \\ \begin{pmatrix} \infty \\ \infty \end{pmatrix} & , t + \lambda_{ij}(t) > T \end{cases} \quad ; \ \forall (i,j) \in A$$

1 Assign the initial label to each node for all $t \in \{0, \ldots, T\}$

$$\Pi_i(t)^{(0)} = \begin{cases} \left\{ \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} \ ; \ (t,t); \emptyset \right) \right\} & , i = d \\ \emptyset & , \text{ otherwise} \end{cases}$$

Set $l := 0$.

2 Calculate the new set $\Pi_i(t_{ready})^{(l+1)}$, $\forall i \in N$, $\forall t_{ready} \in \{0, \ldots, T\}$, as determined by (6.12).

3 If $\Pi_i(t_{ready})^{(l+1)} = \Pi_i(t_{ready})^{(l)}$, $\forall i \in N$, $\forall t_{ready} \in \{0, \ldots, T\}$, then go to Step 4. Otherwise, set $l := l + 1$ and go to Step 2.

4 For any $i \in N$ and any $t \in \{0, \ldots, T\}$, output $PO(\mathbb{P}_{id}(t))$ as obtained by forwardtracking the successor pointers of $\Pi_i(t)$.
Terminate the algorithm.

To solve the vector minimization problem in Step 2 of Algorithm 6.1, we sort the set of labels $\Pi_j(t_{arrival})^{(l)} + c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})$ according to the first criteria. If we denote the cardinality of this set of labels by b, then the sorting process can be done in $\mathcal{O}(b \ log \ b)$ by applying *mergesort technique*. Maintaining the set of labels as a linked list will give an advantage to the implementation of mergesort technique, since there will be no demand for an extra space $\mathcal{O}(b)$ as if this sorting technique is applied on arrays. On the set of sorted labels (with respect to the first criteria), the minimum vectors can be obtained in $\mathcal{O}(b)$. Therefore, the vector minimization problem in Step 2 of Algorithm 6.1 can be solved in $\mathcal{O}(b \ log \ b)$.

6.4.2 Backward Label Setting Algorithm

Since all data are nonnegative, no negative dynamic cycle will exist. We therefore in this subsection, extend the idea of Djikstra's label setting static shortest path. Theorem 6.1 guarantees that the labeling process can be done in a *backward direction from the sink node d to all other nodes in the network.*

We consider, over the set of all labels associated with each node $i \in N$ at each ready time $t_{ready} \in \{0, \ldots, T\}$, the set of *temporary* and *permanent labels* denoted by $\Pi_i(t_{ready})_{tmp}$ and $\Pi_i(t_{ready})_{prm}$, respectively. The set $\Pi_i(t_{ready})_{prm}$ contains only labels having nonequivalent Pareto optimal cost components. The corresponding dynamic paths from this node i with

ready time t_{ready} can be found by tracing its successor pointers (i.e., *succ_ptr*). At any intermediate step of the algorithm, the permanent labels remain unchanged, but the temporary labels can be changed or deleted. Initially, for every node i and every time t_{ready}, we define $\Pi_i(t_{ready})_{prm}$ by an empty set. Since we assume that no arc leaves the sink node d, we assign the sink node d at any time $t_{ready} \in \{0, \ldots, T\}$, a temporary label having cost component zero, and every other node i an empty set, i.e.,

$$\Pi_i(t_{ready})_{tmp} := \begin{cases} \left\{ \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}; (t_{ready}, t_{ready}); \emptyset \right) \right\} & , i = d \\ \emptyset & , \text{otherwise} \end{cases}, \quad t_{ready} \in \{0, \ldots, T\} \tag{6.13}$$

At each iteration, a label is removed from the set of temporary labels and put into the set of permanent labels. This means that a label is set permanently in each iteration. Every label $\pi_i(t_{ready})$ in $\Pi_i(t_{ready})_{prm}$ determines a dynamic path from i with ready time t_{ready} that contains only node-time pairs $j'(t_{ready_{j'}}, t_{depart_{j'}})$ having labels in $\Pi_{j'}(t_{ready_{j'}})_{prm}$ as internal nodes.

Suppose that a label $\pi_j^*(t_{arrival}) \in \Pi_j(t_{arrival})_{tmp}$ is chosen as the *pivot label* to be put in $\Pi_j(t_{arrival})_{prm}$, i.e.,

$$\Pi_j(t_{arrival})_{prm} := \Pi_j(t_{arrival})_{prm} \cup \{\pi_j^*(t_{arrival})\}$$

This pivot label is then used to generate the new labels (as candidate of temporary labels) for all nodes i at time t_{ready} those can reach node j at time $t_{arrival}$. Let $\hat{\pi}_i(t_{ready})$ be the candidate for the temporary label of a node i at ready time t_{ready}. We define

$$\begin{aligned} \hat{\pi}_i(t_{ready}) := \; & \pi_j^*(t_{depart} + \lambda_{ij}(t_{depart})) + (c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})), \\ & (i,j) \in A, \; t_{depart} + \lambda_{ij}(t_{depart}) = t_{arrival}, \\ & t_{depart} \in Depart_i(t_{ready}) \end{aligned} \tag{6.14}$$

This label $\hat{\pi}_i(t_{ready})$ is then merged with $\Pi_i(t_{ready})_{tmp}$. Executing this step means deleting all labels representing dominated dynamic paths from i with ready time t_{ready} to the sink. The algorithm stops when $\Pi_i(t_{ready})_{tmp} = \emptyset$ for all $i \in N$ and all $t_{ready} \in \{0, \ldots, T\}$.

Let us define the following set operation *Merge* to obtain the minimal complete set of $B_1 \cup B_2$ with B_1 and B_2 two set of labels. We denote by $cost(v)$ the cost component of $v \in B_1 \cup B_2$. For any B_1 and B_2 we define

$$\begin{aligned} Merge(B_1, B_2) = \; & B_1 \cup B_2 - \{v \in B_1 \cup B_2 \; : \; cost(u) \leq cost(v) \text{ for some} \\ & u \neq v \text{ with } u, v \in B_1 \cup B_2\} \end{aligned} \tag{6.15}$$

where the relation $\leq$ is given by Definition 6.5. This merge operation includes the process of deleting the corresponding time and pointer components of the dominated cost components.

By (6.15), $Merge(B_1, B_2)$ contains only nonequivalent Pareto optimal labels in $B_1 \cup B_2$. Since the set $\{\hat{\pi}_i(t_{ready})\}$ is a singleton, it is obvious that $Merge(\{\hat{\pi}_i(t_{ready})\}, \Pi_i(t_{ready})_{tmp})$ can be done in $\mathcal{O}(\mid \Pi_i(t_{ready})_{tmp} \mid)$ time.

The following theorem implies that the labeling process of $\pi_i(t)$ can be done in a decreasing order of time.

Theorem 6.2 *The labeling process of $\pi_i(t_{ready})$ depends on the label at time $t' > t_{ready}$ but does not depend on the labels at the lower time period.*

Proof:
Since $\lambda_{ij}(t_{depart}) > 0$, (6.14) includes only $\pi_j(t_{arrival})$ for $t_{arrival} > t_{depart}$. ■

Corollary 6.1 *The labeling process can be done in a decreasing order of time, i.e., starting from time $t_{ready} = T$ down to 0.*

This corollary implies that the algorithm is started by processing the set $\Pi_i(t_{ready} - 1)_{tmp}$ for any $i \in N - \{d\}$ only when $\Pi_j(t_{ready})_{tmp} = \emptyset$ for all $j \in N$ and $t_{ready} \in \{1, \ldots, T\}$. Consequently, the selection of a pivot label $\pi_j^*(t_{arrival}) \in \Pi_j(t_{arrival})_{tmp}$ can be done arbitrarily, as stated by the following proposition.

Proposition 6.1 *An arbitrary selection rule can be applied to choose a pivot label $\pi_j^*(t_{arrival})$ from the set $\Pi_j(t_{arrival})_{tmp}$. In effect, each member of $\Pi_j(t)_{prm}$ for any $t \in \{0, \ldots, T\}$ defines a Pareto optimal $j - d$ dynamic path with ready time t.*

Proof:
The merging process (6.15) implies that $\Pi_j(t)_{tmp}$ contains only nonequivalent Pareto optimal labels. Furthermore, Corollary 6.1 implies that during the labeling process at time t, no additional label is inserted into or deleted from $\Pi_j(t)_{tmp}$. The members of $\Pi_j(t)_{tmp}$ are moved to $\Pi_j(t)_{prm}$ one by one and the labeling process at time t finishes when $\Pi_j(t)_{tmp}$ is empty. Therefore, an arbitrary selection rule for the pivot label can be applied and guarantees that $\Pi_j(t)_{prm}$ contains only nonequivalent Pareto optimal labels. ■

This proposition implies that $\pi_j^*(t_{arrival})$ can be chosen in a constant time $\mathcal{O}(1)$ by e.g., maintaining the set $\Pi_j(t_{arrival})_{tmp}$ as a linked list and take $\pi_j^*(t_{arrival})$ from the head of this list.

In order to do labeling in a decreasing order of time, we need to determine

- $Arrival_{ij}^{-1}(t_{arrival})$, the set of all possible departure times in $\{0, \ldots, T\}$ along arc $(i, j) \in A$ arriving at node j at time $t_{arrival}$, i.e.,

$$Arrival_{ij}^{-1}(t_{arrival}) \quad := \quad \{t' \ : \ t' + \lambda_{ij}(t') = t_{arrival}\} \tag{6.16}$$

- $Depart_i^{-1}(t_{depart})$, the set of all possible ready times in $\{0,\ldots,T\}$ at node $i \in N-\{d\}$ departing from node i at time t_{depart}, i.e.,

$$Depart_i^{-1}(t_{depart}) := \{t' : 0 \le t \le t' \le t_{depart} \text{ with } \prod_{t''=t}^{t_{depart}-1} w_i(t'') > 0 \text{ and } w_i(t-1) = 0\} \quad (6.17)$$

The set $Arrival_{ij}^{-1}(t_{arrival})$ may be empty. But, the set $Depart_i^{-1}(t_{depart})$ has cardinality at least one, since $t_{depart} \in \{0,\ldots,T\}$ itself must be in this set. These two inverse sets can be determined in a preprocessing step before the labeling process. The alternative is to compute these inverse sets as needed while employing the labeling algorithm.

The proposed algorithm to find the minimal complete set of all PO-paths from every node $i \in N - \{d\}$ in the network to the sink d and for all ready times $t \in \{0,\ldots,T\}$ is summarized in Algorithm 6.2. At the end of algorithm there are as many permanent labels associated with node $i \in N$ and ready time $t \in \{0,\ldots,T\}$ as the number of nonequivalent Pareto optimal dynamic paths from i with ready time t to the sink node. The correctness of Algorithm 6.2 is shown by the following proposition.

Proposition 6.2 *Algorithm* 6.2 *generates* $PO(\mathbb{P}_{id}(t))$ *for any node* $i \in N$ *and ready time* $t \in \{0,\ldots,T\}$.

Proof:
By (6.15), $\Pi_i(t)_{tmp}$ contains only nonequivalent Pareto optimal labels. The set $\Pi_i(t)_{tmp}$ will be emptied and its content is moved into $\Pi_i(t)_{prm}$ during the labeling process at time t. Theorem 6.2 and Proposition 6.1 imply that $\Pi_i(t)_{prm}$ is a minimal complete set. Furthermore, in step 2, all possible candidates for the Pareto optimal labels of any predecessor nodes $i \in N$ are tested from all permanent Pareto optimal labels of the successor node $j \in N$ with $(i,j) \in A$. This is correct since by Theorem 6.1, no Pareto optimal path can be constructed from a dominated sub-path. ■

Later in Section 6.6 we compare the performance of Algorithms 6.1 and 6.2. The following example illustrates Algorithm 6.2.

Example 6.5
Consider a network in Figure 6.7. Node 5 is the sink node. We define the time horizon T equals to eight time units. The maximum waiting w, cost data corresponding to the travel cost c of two criteria and waiting cost h for every arc and for every time t in $\{0,\ldots,T\}$ are given below.

$$c_{01}(t) = \begin{pmatrix} 1 \\ 1 \end{pmatrix},\ t \le 8 \ ; \quad c_{02}(t) = \begin{pmatrix} 2 \\ 1 \end{pmatrix},\ t \le 8 \ ; \quad c_{12}(t) = \begin{pmatrix} 7 \\ 8 \end{pmatrix},\ t \le 8$$

Algorithm 6.2

INPUT Network $G = (N, A, T)$, $w(t), c(t)$, and $h(t)$.
OUTPUT $PO(\mathbb{P}_{id}(t))$, $\forall i \in N$, $\forall t \in \{0, \ldots, T\}$.

0 Modify the cost vector $c_{ij}(t)$, $\forall t \in \{0, \ldots, T\}$ as follows.

$$c_{ij}(t) = \begin{cases} \begin{pmatrix} c_{ij}^1(t) \\ c_{ij}^2(t) \end{pmatrix} & , t + \lambda_{ij}(t) \leq T \\ \begin{pmatrix} \infty \\ \infty \end{pmatrix} & , t + \lambda_{ij}(t) > T \end{cases} \quad ; \ \forall (i,j) \in A$$

1 Define the initial set of temporary labels to every node $i \in N$ and for all $t \in \{0, \ldots, T\}$ as

$$\Pi_i(t)_{tmp} := \begin{cases} \left\{ \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} \ ; \ (t,t) \ ; \ \emptyset \right) \right\} & , i = d \\ \emptyset & , \text{ otherwise} \end{cases}$$

Assign the empty set as the initial set of permanent labels to every node $i \in N$ and for all $t \in \{0, \ldots, T\}$, i.e., $\Pi_i(t)_{prm} := \emptyset$.

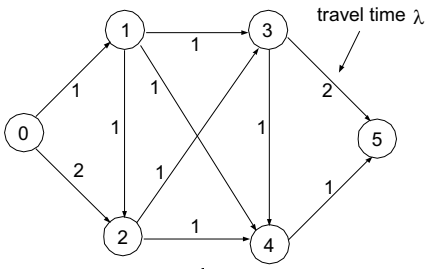
2 For $t_{arrival} = T$ down to 0 do
{
  While $\bigcup_{i \in N} \Pi_i(t_{arrival})_{tmp} \neq \emptyset$ do
  { Select a pivot label $\pi_j^*(t_{arrival})$ from $\bigcup_{i \in N} \Pi_i(t_{arrival})_{tmp}$
    Remove $\pi_j^*(t_{arrival})$ from $\Pi_j(t_{arrival})_{tmp}$ to $\Pi_j(t_{arrival})_{prm}$.
    For all $(i,j) \in A$ do
    { For each $t_{depart} \in Arrival_{ij}^{-1}(t_{arrival})$ do
      {For each $t_{ready} \in Depart_i^{-1}(t_{depart})$ do
        { $\hat{\pi}_i(t_{ready}) := \pi_j^*(t_{arrival}) + \big(c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})\big)$
          $\Pi_i(t_{ready})_{tmp} := Merge\big(\Pi_i(t_{ready})_{tmp}, \{\hat{\pi}_i(t_{ready})\}\big)$
        }
      }
    }
  }
}

3 For any $i \in N$ and any $t \in \{0, \ldots, T\}$, the set $PO(\mathbb{P}_{id}(t))$ is obtained by forwardtracking the successor pointers of $\Pi_i(t)_{prm}$.
Terminate the algorithm.

$$c_{13}(t) = \begin{cases} \begin{pmatrix} 2 \\ 6 \end{pmatrix} & , t \leq 3 \\ \begin{pmatrix} 4 \\ 4 \end{pmatrix} & , t > 3 \end{cases} \ ; \ c_{14}(t) = \begin{cases} \begin{pmatrix} 3 \\ 1 \end{pmatrix} & , t \leq 3 \\ \begin{pmatrix} 5 \\ 3 \end{pmatrix} & , t > 3 \end{cases} \ ;$$

$$c_{23}(t) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \ t \leq 8 \ ; \ c_{24}(t) = \begin{pmatrix} 1 \\ 5 \end{pmatrix}, \ t \leq 8 \ ; \ c_{34}(t) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \ \forall t \leq 8 \ ;$$

$$c_{35}(t) = \begin{cases} \begin{pmatrix} 2 \\ 2 \end{pmatrix} & , t \leq 2 \\ \begin{pmatrix} 4 \\ 3 \end{pmatrix} & , t > 2 \end{cases} \ ; \ c_{45}(t) = \begin{cases} \begin{pmatrix} 4 \\ 3 \end{pmatrix} & , t \leq 3 \\ \begin{pmatrix} 0 \\ 2 \end{pmatrix} & , t > 3 \end{cases}$$



$$\lambda_{23}(t) = \begin{cases} 2 & ,t \leq 3 \\ 1 & ,t > 3 \end{cases}$$

Figure 6.7: Network for Example 6.5

$$H_i(t,t') = (t'-t),\ \forall t' \geq t \ \ ;\ \ w_i(t) = \begin{cases} 1 & ,i = 2,3,4 \text{ and } t \geq 4 \\ 0 & ,\text{otherwise} \end{cases}$$

The final label $\Pi_i(t)_{prm}$ for every node $i \in N$ and for every time $t \in \{0,\ldots,T\}$ are reported in Table 6.2 and Table 6.3.

The Pareto optimal dynamic paths from node 0 to node 5 that are ready to leave node 0 at time $t = 0$ are

$$\{0(0,0), 2(2,2), 3(4,4), 5(6,6)\} \quad \text{with value} \quad \begin{pmatrix} 7 \\ 5 \end{pmatrix}$$

$$\{0(0,0), 2(2,2), 3(3,3), 4(4,4), 5(5,5)\} \quad \text{with value} \quad \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

Hence,

$$PO(\mathbb{P}_{05}(0)) = \big\{\{0(0,0), 2(2,2), 3(4,4), 5(6,6)\}, \{0(0,0), 2(2,2), 3(3,3), 4(4,4), 5(5,5)\}\big\}$$

We see from Table 6.3 that there is no path which can leave node 0 after time $t = 5$ and arrive at node 5 in time horizon $T = 8$ or earlier. □

6.5 Algorithm for All Ready Times with Arbitrary Attributes

When negative travel times and costs are allowed, Theorem 6.2 is no longer valid. Moreover, as shown in Example 6.3, there may exist some negative dynamic cycles within T. Consequently, dynamic shortest path algorithms must be able to detect negative dynamic cycles.

We describe here an algorithm based on the label correcting technique. Inside we use

Node i	$\Pi_i(8)_{prm}$	$\Pi_i(7)_{prm}$	$\Pi_i(6)_{prm}$
5	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(8,8);\emptyset\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(7,7);\emptyset\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(6,6);\emptyset\right)\right\}$
4	$\emptyset$	$\left\{\left(\begin{pmatrix}0\\2\end{pmatrix};(7,7);5^1(8)\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\2\end{pmatrix};(6,6);5^1(7)\right)\right\}$
3	$\emptyset$	$\emptyset$	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(6,6);5^1(8)\right),\left(\begin{pmatrix}2\\4\end{pmatrix};(6,6);4^1(7)\right)\right\}$
2	$\emptyset$	$\emptyset$	$\left\{\left(\begin{pmatrix}1\\7\end{pmatrix};(6,6);4^1(7)\right)\right\}$
1	$\emptyset$	$\emptyset$	$\left\{\left(\begin{pmatrix}5\\5\end{pmatrix};(6,6);4^1(7)\right)\right\}$
0	$\emptyset$	$\emptyset$	$\emptyset$

Node i	$\Pi_i(5)_{prm}$	$\Pi_i(4)_{prm}$	$\Pi_i(3)_{prm}$
5	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(5,5);\emptyset\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(4,4);\emptyset\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(3,3);\emptyset\right)\right\}$
4	$\left\{\left(\begin{pmatrix}0\\2\end{pmatrix};(5,5);5^1(6)\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\2\end{pmatrix};(4,4);5^1(5)\right)\right\}$	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(3,3);5^1(4)\right)\right\}$
3	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(5,5);5^1(6)\right),\left(\begin{pmatrix}2\\4\end{pmatrix};(5,5);4^1(6)\right)\right\}$	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(4,4);5^1(5)\right),\left(\begin{pmatrix}2\\4\end{pmatrix};(4,4);4^1(5)\right)\right\}$	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(3,3);5^1(4)\right),\left(\begin{pmatrix}2\\4\end{pmatrix};(3,3);4^1(4)\right)\right\}$
2	$\left\{\left(\begin{pmatrix}5\\4\end{pmatrix};(5,5);3^1(6)\right),\left(\begin{pmatrix}3\\5\end{pmatrix};(5,5);3^2(6)\right),\left(\begin{pmatrix}1\\7\end{pmatrix};(5,5);4^1(6)\right)\right\}$	$\left\{\left(\begin{pmatrix}5\\4\end{pmatrix};(4,4);3^1(5)\right),\left(\begin{pmatrix}3\\5\end{pmatrix};(4,4);3^2(5)\right),\left(\begin{pmatrix}1\\7\end{pmatrix};(4,4);4^1(5)\right)\right\}$	$\left\{\left(\begin{pmatrix}5\\4\end{pmatrix};(3,3);3^1(5)\right),\left(\begin{pmatrix}3\\5\end{pmatrix};(3,3);3^2(4)\right),\left(\begin{pmatrix}1\\7\end{pmatrix};(3,3);4^1(4)\right)\right\}$
1	$\left\{\left(\begin{pmatrix}5\\5\end{pmatrix};(5,5);4^1(6)\right)\right\}$	$\left\{\left(\begin{pmatrix}5\\5\end{pmatrix};(4,4);4^1(5)\right)\right\}$	$\left\{\left(\begin{pmatrix}3\\3\end{pmatrix};(3,3);4^1(4)\right)\right\}$
0	$\left\{\left(\begin{pmatrix}6\\6\end{pmatrix};(5,5);1^1(6)\right)\right\}$	$\left\{\left(\begin{pmatrix}6\\6\end{pmatrix};(4,4);1^1(5)\right),\left(\begin{pmatrix}3\\8\end{pmatrix};(4,4);2^1(6)\right)\right\}$	$\left\{\left(\begin{pmatrix}7\\5\end{pmatrix};(3,3);2^1(5)\right),\left(\begin{pmatrix}5\\6\end{pmatrix};(3,3);2^2(5)\right),\left(\begin{pmatrix}3\\8\end{pmatrix};(3,3);2^3(5)\right)\right\}$

Table 6.2: The label set $\Pi_i(t)_{prm}$ for every node $i \in N$ and for every time $t \in \{0, \ldots, T\}$ of Example 6.5

a scan eligible (SE) list to store the nodes which have potential of improving the labels of at least one other node. During the initialization step of the algorithm, only the sink node is in the SE list. The labeling process moves backward from the sink to all other nodes.

Node i	$\Pi_i(2)_{prm}$	$\Pi_i(1)_{prm}$	$\Pi_i(0)_{prm}$
5	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(2,2);\emptyset\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(1,1);\emptyset\right)\right\}$	$\left\{\left(\begin{pmatrix}0\\0\end{pmatrix};(0,0);\emptyset\right)\right\}$
4	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(2,2);5^1(3)\right)\right\}$	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(1,1);5^1(2)\right)\right\}$	$\left\{\left(\begin{pmatrix}4\\3\end{pmatrix};(0,0);5^1(1)\right)\right\}$
3	$\left\{\left(\begin{pmatrix}2\\2\end{pmatrix};(2,2);5^1(4)\right)\right\}$	$\left\{\left(\begin{pmatrix}2\\2\end{pmatrix};(1,1);5^1(3)\right)\right\}$	$\left\{\left(\begin{pmatrix}2\\2\end{pmatrix};(0,0);5^1(2)\right)\right\}$
2	$\left\{\left(\begin{pmatrix}5\\4\end{pmatrix};(2,2);3^1(4)\right),\left(\begin{pmatrix}3\\5\end{pmatrix};(2,2);3^2(4)\right)\right\}$	$\left\{\left(\begin{pmatrix}5\\4\end{pmatrix};(1,1);3^1(3)\right),\left(\begin{pmatrix}3\\5\end{pmatrix};(1,1);3^2(3)\right)\right\}$	$\left\{\left(\begin{pmatrix}3\\3\end{pmatrix};(0,0);3^1(2)\right)\right\}$
1	$\left\{\left(\begin{pmatrix}7\\4\end{pmatrix};(2,2);4^1(3)\right),\left(\begin{pmatrix}6\\9\end{pmatrix};(2,2);3^1(3)\right),\left(\begin{pmatrix}4\\10\end{pmatrix};(2,2);3^2(3)\right)\right\}$	$\left\{\left(\begin{pmatrix}7\\4\end{pmatrix};(1,1);4^1(2)\right),\left(\begin{pmatrix}4\\8\end{pmatrix};(1,1);3^1(2)\right)\right\}$	$\left\{\left(\begin{pmatrix}7\\4\end{pmatrix};(0,0);4^1(1)\right),\left(\begin{pmatrix}4\\8\end{pmatrix};(0,0);3^1(1)\right)\right\}$
0	$\left\{\left(\begin{pmatrix}4\\4\end{pmatrix};(2,2);1^1(3)\right),\left(\begin{pmatrix}3\\8\end{pmatrix};(2,2);2^3(4)\right)\right\}$	$\left\{\left(\begin{pmatrix}7\\5\end{pmatrix};(1,1);2^1(3)\right),\left(\begin{pmatrix}5\\6\end{pmatrix};(1,1);2^2(3)\right),\left(\begin{pmatrix}3\\8\end{pmatrix};(1,1);2^3(3)\right)\right\}$	$\left\{\left(\begin{pmatrix}7\\5\end{pmatrix};(0,0);2^1(2)\right),\left(\begin{pmatrix}5\\6\end{pmatrix};(0,0);2^2(2)\right)\right\}$

Table 6.3: (Cont.) The label set $\Pi_i(t)_{prm}$ for every node $i \in N$ and for every time $t \in \{0,\ldots,T\}$ of Example 6.5

The label structure of node i at time t is given by (6.10) and $\Pi_i(t)$ denotes the set of all cost labels associated with node i of ready time t. At each iteration of the algorithm, a node, called the *current node*, is removed from the SE list. In the first iteration, the sink node is defined as the current node. During the scanning process of the current node, the temporary labels of all predecessor nodes of the current node are calculated for each time step $t \in \{0,\ldots,T\}$. As the algorithm progresses, the label $\Pi_i(t)$ is updated in such a way that $\Pi_i(t)$ is always a Pareto optimal set. The cardinality of $\Pi_i(t)$ may increase or decrease as the algorithm proceeds. Upon termination, this set contains the labels denoting all Pareto optimal dynamic paths from node i to the sink node d with ready time t. The current node is selected from the SE list by following the FIFO rule.

Suppose that node i is a predecessor node of the current node j. We denote the set of temporary labels of node i at time $t_{ready} \in \{0,\ldots,T\}$ generated by j by $\{tmpLabel\}$. Recall the definition of $Depart_i(t_{ready})$, the set of all possible departure times in $\{0,\ldots,T\}$

of the ready time t_{ready}, given by (6.11). Using $Depart_i(t_{ready})$, we define

$$\begin{aligned}\{tmpLabel\} \;:=\; & \Pi_j^*(t_{arrival}) + (c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})), \\ & \text{for all } t_{depart} \in Depart_i(t_{ready}) \\ & \text{with } t_{arrival} = t_{depart} + \lambda_{ij}(t_{depart}) \end{aligned} \tag{6.18}$$

The set $\{tmpLabel\}$ is then merged with the current label of node i at time t_{ready}, $\Pi_i(t_{ready})$, yielding $\Pi_i(t_{ready})_{new}$, i.e.,

$$\Pi_i(t_{ready})_{new} = Merge(\Pi_i(t_{ready}), \{tmpLabel\}) \tag{6.19}$$

The definition of *Merge* function is given by (6.15). If $\Pi_i(t_{ready})_{new}$ is not equal to $\Pi_i(t_{ready})$, then we replace $\Pi_i(t_{ready})$ by $\Pi_i(t_{ready})_{new}$ and send the node i to the tail of SE list, i.e., $SE := SE + \{i\}$. We then select again a node from the head of the SE list and repeat the same process until the list SE becomes empty (i.e., no current node can be selected), or the algorithm detects the existence of a negative dynamic cycle.

Lemma 6.1 *If at any time during the labeling algorithm there is a dynamic cycle, then this dynamic cycle is negative.*

Proof:
Suppose the labeling process produces a dynamic path P given by

$$P := \{j_1(t_1, t_1'), j_2(t_2, t_2'), \ldots, j_{k-1}(t_{k-1}, t_{k-1}')\}$$

with $j_1 = i$ as shown in Figure 6.8 (take $k = 4$).

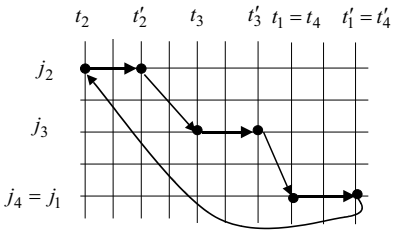


Figure 6.8: The dynamic cycle for the proof of Lemma 6.1

Since we do cost labeling in a backward direction, the cost label of any node $j_l \in P$ satisfies

$$cost(\pi_{j_{l+1}}(t_{l+1})) = cost(\pi_{j_l}(t_l)) + (c_{j_{l+1},j_l}(t_{l+1}') \oplus H_{j_{l+1}}(t_{l+1}, t_{l+1}'))$$

By taking the sum over $l = 1$ to $k - 2$, we obtain

$$cost(\pi_{j_{k-1}}(t_{k-1})) = cost(\pi_{j_1}(t_1)) + \sum_{l=1}^{k-2}(c_{j_{l+1},j_l}(t_{l+1}') \oplus H_{j_{l+1}}(t_{l+1}, t_{l+1}')) \tag{6.20}$$

Now, suppose that the scanning process of the current node j_{k-1} on arc (i, j_{k-1}) creates a dynamic cycle C, i.e., $j_k = j_1 = i$, $t_k = t_1$, $t_k^{'} = t_1^{'}$, and

$$C := \{j_1(t_1, t_1^{'}), j_2(t_2, t_2^{'}), \ldots, j_{k-1}(t_{k-1}, t_{k-1}^{'}), j_k(t_k, t_k^{'}) = j_1(t_1, t_1^{'})\}$$

This cycle is created only if either $cost(\pi_{j_{k-1}}(t_{k-1})) + (c_{j_1,j_{k-1}}(t_1^{'}) \oplus H_{j_1}(t_1, t_1^{'}))$ dominates $cost(\pi_{j_1}(t_1))$ or both $cost(\pi_{j_{k-1}}(t_{k-1}))+(c_{j_1,j_{k-1}}(t_1^{'})\oplus H_{j_1}(t_1, t_1^{'}))$ and $cost(\pi_{j_1}(t_1))$ are Pareto optimal. If this scanning process replaces the current label of $\pi_{j_1}(t_1)$ (i.e., $cost(\pi_{j_{k-1}}(t_{k-1}))+ (c_{j_1,j_{k-1}}(t_1^{'}) \oplus H_{j_1}(t_1, t_1^{'}))$ dominates $cost(\pi_{j_1}(t_1))$), then just before the scanning process we must have

$$cost(\pi_{j_1}(t_1)) > cost(\pi_{j_{k-1}}(t_{k-1})) + (c_{j_1,j_{k-1}}(t_1^{'}) \oplus H_{j_1}(t_1, t_1^{'})) \tag{6.21}$$

The sum of (6.20) and (6.21) yields

$$\begin{aligned} cost(\pi_{j_{k-1}}(t_{k-1})) + cost(\pi_{j_1}(t_1)) &> cost(\pi_{j_1}(t_1)) + cost(\pi_{j_{k-1}}(t_{k-1})) + \\ &\quad \sum_{l=1}^{k-1}(c_{j_{l+1},j_l}(t_{l+1}^{'}) \oplus H_{j_{l+1}}(t_{l+1}, t_{l+1}^{'})) \\ 0 &> \mathbf{c}(C) \end{aligned} \tag{6.22}$$

On the other hand, if both $cost(\pi_{j_{k-1}}(t_{k-1})) + (c_{j_1,j_{k-1}}(t_1^{'}) \oplus H_{j_1}(t_1, t_1^{'}))$ and $cost(\pi_{j_1}(t_l))$ are Pareto optimal, then one component of

$$cost(\pi_{j_{k-1}}(t_{k-1})) + (c_{j_1,j_{k-1}}(t_1^{'}) \oplus H_{j_1}(t_1, t_1^{'}))$$

must be smaller than the corresponding component of $cost(\pi_{j_1}(t_l))$ and the other component is bigger than that of $cost(\pi_{j_1}(t_l))$. Without loss of generality, we assume that the first component of $cost(\pi_{j_{k-1}}(t_{k-1})) + (c_{j_1,j_{k-1}}(t_1^{'}) \oplus H_{j_1}(t_1, t_1^{'}))$ is smaller than the first component of $cost(\pi_{j_1}(t_l))$, i.e.,

$$cost^1(\pi_{j_1}(t_l)) > cost^1(\pi_{j_{k-1}}(t_{k-1})) + (c^1_{j_1,j_{k-1}}(t_1^{'}) + H_{j_1}(t_1, t_1^{'})) \tag{6.23}$$

The sum of the first component of (6.20) and (6.23) yields

$$0 > \mathbf{c}^1(C) \tag{6.24}$$

Hence, C is a negative dynamic cycle. ■

Proposition 6.3 *Suppose that for some node $i \in N - \{d\}$ and some time $t \in \{0, \ldots, T\}$, there is some dynamic $i - d$ path with ready time t containing a negative dynamic cycle.*

(a) *If both components of the cost of the negative dynamic cycle are negative, then $PO(\mathbb{P}_{id}(t)) = \emptyset$.*

(b) *If one component of the cost of the negative dynamic cycle is negative and the other is zero, then $PO(\mathbb{P}_{id}(t)) = \emptyset$.*

(c) *If one component of the negative dynamic cycle is negative and the other is positive, then there will be an infinite number of Pareto optimal dynamic paths.*

Proof:
Let C be a negative dynamic cycle associated with the label $\pi_i(t)$.

(a) Suppose that $cost^1(\pi_i(t)) < 0$ and $cost^2(\pi_i(t)) < 0$. Let's construct a new label of node i at time t, $\pi_i'(t)$, obtained by repeating the cycle C two times. We obtain

$$cost^1(\pi_i'(t)) = 2cost^1(\pi_i(t)) < 0 \text{ and } cost^2(\pi_i'(t)) = 2cost^2(\pi_i(t)) < 0$$

Obviously, $\pi_i'(t)$ dominates $\pi_i(t)$. Since we can repeat the cycle infinitely, the labeling process will always be able to produce a new $\pi_i(t)$ that dominates the current label $\pi_i(t)$. Hence, there will be no Pareto optimal $\pi_i(t)$.

(b) Without loss of generality, assume that $cost^1(\pi_i(t)) < 0$ and $cost^2(\pi_i(t)) = 0$. As in the proof of (a), we can have label $\pi_i'(t)$ such that

$$cost^1(\pi_i'(t)) = 2cost^1(\pi_i(t)) < 0 \text{ and } cost^2(\pi_i'(t)) = 2cost^2(\pi_i(t)) = 0$$

Therefore, $\pi_i'(t)$ dominates $\pi_i(t)$. Since we can repeat the cycle infinitely, there will be no Pareto optimal $\pi_i(t)$.

(c) Without loss of generality, assume that $cost^1(\pi_i(t)) > 0$ and $cost^2(\pi_i(t)) < 0$. As in the proof of (a), let's construct a new label $\pi_i'(t)$ by repeating C two times. We obtain

$$cost^1(\pi_i'(t)) = 2cost^1(\pi_i(t)) > 0 \text{ and } cost^2(\pi_i'(t)) = 2cost^2(\pi_i(t)) < 0$$

where

$$cost^1(\pi_i'(t)) > 2cost^1(\pi_i(t)) \quad \text{and} \quad cost^2(\pi_i'(t)) < 2cost^2(\pi_i(t))$$

Therefore, both $\pi_i'(t)$ and $\pi_i(t)$ are Pareto optimal paths. Since we can repeat this cycle infinitely, there will be an infinite number of Pareto optimal dynamic paths. ■

To detect the existence of a negative dynamic cycle, we divide the execution of the algorithm into *passes*. The definition of a pass is given as folows.

Definition 6.10

- *Pass* 0 *ends after node* d *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 i is removed from the list before the end of pass k, then there must be a node j with $(i,j) \in A$ removed from the list before the end of pass $(k-1)$ and there are $t', t \in \{0,\ldots,T\}$ such that $\Pi_i(t)$ is improved from $\Pi_j(t')$. We denote by ps the index of a pass of the execution of the algorithm. From the definition of a pass, we increase ps by one if at the end of pass ps, the list SE is not empty. We denote by ls the node in the tail of SE list at the end of pass $k-1$, i.e.,

$$ls := tail(SE)$$

If the current node is equal to ls at the end of pass k and $SE \neq \emptyset$, then

$$ps := ps + 1$$

Lemma 6.2 *If the SE list is not empty at the end of pass $(n-1)(T+1)$, then the dynamic network must contain a negative dynamic cycle.*

Proof:
Define ps_i to be the greatest k such that node i is selected as the current node during pass k. If ps_i is positive, then there must be a node j with $(i,j) \in A$ and some t, t' such that $\Pi_i(t)$ is updated from $\Pi_j(t')$ during pass $(ps_i - 1)$, i.e., node j at time t' becomes a successor component of some member of $\Pi_i(t)$. Since the label of node j may be updated again at a pass greater than ps_i, we obtain the following interrelation

$$ps_i \leq ps_j + 1$$

The value of ps_j is strictly greater than $(ps_i - 1)$ only if the label of node j is updated again before the end of pass greater or equal than ps_i. Therefore, the ps values of any two consecutive nodes in a dynamic path differs by one. Since it is assumed that no arc leaves node d, we obtain $ps_d = 0$. Furthermore, we know that the maximum number of nodes but the sink node in the time-expanded network for $\{0,\ldots,T\}$ is $(n-1)(T+1)$. Now, suppose the algorithm runs until a node i is selected as the current node in pass $(n-1)(T+1)+1$. Since $ps_i = (n-1)(T+1)+1$, then we must eventually repeat some nodes j at time $t' \in \{0,\ldots,T\}$ during the forwardtracking process to find an $i-d$ Pareto optimal dynamic path that is ready at node i at time $t \in \{0,\ldots,T\}$. Hence, this path contains a dynamic cycle. By Lemma 6.1, this cycle must be a negative dynamic cycle. ■

Hence, the existence of a negative cycle can be detected by either finding a dynamic cycle (see Lemma 6.1) or checking the number of passes.

Since $\{tmpLabel\}$ is in general not a singleton, the merging process in (6.19) is not as simply as that in Section 6.4. Brumbaugh-Smith and Shier [BSS89] proposed an algorithm for a merge operation of two Pareto optimal sets, B_1 and B_2, which has a linear worst-case time complexity of $\mathcal{O}(\mid B_1 \mid + \mid B_2 \mid)$. This method works on assumption that B_1 and B_2 are ordered sets with respect to their cost components, arranged by increasing value of

the first criteria of the cost component. Hence, if the cardinality of B_1 is k_1 and $cost(B_1)$ defines a set of cost components of all members of B_1, then

$$cost(B_1) = \left\{ \begin{pmatrix} u_1^1 \\ u_1^2 \end{pmatrix}, \begin{pmatrix} u_2^1 \\ u_2^2 \end{pmatrix}, \ldots, \begin{pmatrix} u_{k_1}^1 \\ u_{k_1}^2 \end{pmatrix} : l' < l'' \Rightarrow u_{l'}^1 < u_{l''}^1, \right.$$
$$\left. l', l'' \in \{1, \ldots, k_1\} \right\}$$

Since $cost(B_1)$ contains only different Pareto optimal values, it follows that the values corresponding to the second criteria form a strictly decreasing sequence, i.e., $l' < l'' \Rightarrow u_{l'}^2 > u_{l''}^2$.

Let us denote the time component and the pointer component of $u \in B_1$ by $time(u)$ and $point(u)$, respectively. Then B_1 can be represented as

$$B_1 = \left\{ \Big(cost(u_1); time(u_1); point(u_1) \Big), \ldots, \right.$$
$$\Big(cost(u_{k_1}); time(u_{k_1}); point(u_{k_1}) \Big) : l' < l'' \Rightarrow u_{l'}^1 < u_{l''}^1;$$
$$\left. l', l'' \in \{1, \ldots, k_1\} \right\}$$

The same arrangement is applied to B_2, i.e., if k_2 is the cardinality of B_2, then

$$B_2 = \left\{ \Big(cost(v_1); time(v_1); point(v_1) \Big), \ldots, \right.$$
$$\Big(cost(v_{k_2}); time(v_{k_2}); point(v_{k_2}) \Big) : l' < l'' \Rightarrow v_{l'}^1 < v_{l''}^1;$$
$$\left. l', l'' \in \{1, \ldots, k_2\} \right\}$$

We define *the set concatenation operation* "&" as

$$\{u_1, \ldots, u_{k_1}\} \& \{v_1, \ldots, v_{k_2}\} := \{u_1, \ldots, u_{k_1}, v_1, \ldots, v_{k_2}\}$$

Details of the modified Merge procedure are given in Algorithm 6.3.

Proposition 6.4 *Algorithm* 6.3 *produces a minimal complete set* M *of* $B_1 \cup B_2$ *in* $\mathcal{O}(\mid B_1 \mid + \mid B_2 \mid)$.

Proof:
The proof focusses on step 2 of the algorithm. For any $i \leq k_1$ and $j \leq k_2$, if $u_i^1 < v_j^1$ then we check the value of u_i^2 compared with v_j^2. If $u_i^2 \leq v_j^2$, then u_i dominates v_j. Therefore, we skip v_j and remove it from further consideration as a member of M by increasing the counter j. Since v_j^1 is increasing and v_j^2 is decreasing, we can continue removing v_j by

Algorithm 6.3 (adapted from Brumbaugh-Smith and Shier [BSS89])

INPUT B_1 and B_2 are ordered sets arranged by increasing value of the first criteria of the cost component.

OUTPUT $Merge(B_1, B_2)$

0 Initiate counter $i = 1$ and $j = 1$.
Initiate Merge set $M = \emptyset$.

1 If $i > k_1$ then
$M = M\&\{v_j, \ldots, v_{k_2}\}$
Go to Step 3.
If $j > k_2$ then
$M = M\&\{u_i, \ldots, u_{k_1}\}$
Go to Step 3.

2 If $u_i^1 < v_j^1$ then
while $u_i^2 \leq v_j^2$
$j = j + 1$
$M = M\&\{u_i\}$
$i = i + 1$ and Go to Step 1.
Else if $v_j^1 < u_i^1$ then
while $v_j^2 \leq u_i^2$
$i = i + 1$
$M = M\&\{v_j\}$
$j = j + 1$ and Go to Step 1.
Else if $u_i^2 < v_j^2$ then
repeat
$j = j + 1$
until $v_j^2 < u_i^2$
$M = M\&\{u_i\}$
$i = i + 1$ and Go to Step 1.
Else
repeat
$i = i + 1$
until $u_i^2 < v_j^2$
$M = M\&\{v_j\}$
$j = j + 1$ and Go to Step 1.

3 $Merge(B_1, B_2) := M$
Terminate the algorithm.

increasing the counter j as long as $u_i^2 \leq v_j^2$. Furthermore, since u_i^1 and v_j^1 are increasing, u_i is Pareto optimal and placed into the set M. We then increase the counter i and check again the possibility to include the new u_i into M by going back to step 1. We do similarly in the case of $v_j^1 < u_i^1$, but for possibly removing u_i from further consideration as a member of M. Moreover, if $u_i^1 = v_j^1$, then we check the value of u_i^2 compared to v_j^2. If $u_i^2 < v_j^2$, then u_i dominates v_j. Since v_i^1 is increasing and v_j^2 is decreasing in B_2, we skip v_j sequentially until we find $v_j^2 < u_i^2$. At this time we can check again the possibility to include v in M. But, if $u_i^2 > v_j^2$, then similarly we skip u_i sequentially until we find $u_i^2 < v_j^2$. In case the two

cost values u_i and v_j are identical, i.e., $u_i^1 = v_j^1$ and $u_i^2 = v_j^2$, we disregard u_i keeping the minimal property of the set M. The complexity follows directly by computing the steps of the algorithm. ■

The proposed label correcting algorithm to generate all Pareto optimal $i-d$ paths for any $i \in N$ and for any ready time $t \in \{0, \ldots, T\}$ is shown in Algorithm 6.4.

Algorithm 6.4

INPUT Network $G = (N, A, T)$, $w(t), c(t)$, and $h(t)$.
OUTPUT $PO(\mathbb{P}_{id}(t))$, $\forall i \in N$, $\forall t \in \{0, \ldots, T\}$.

0 Modify the cost vector $c_{ij}(t)$, $\forall t \in \{0, \ldots, T\}$ as follows.

$$c_{ij}(t) = \begin{cases} \begin{pmatrix} c_{ij}^1(t) \\ c_{ij}^2(t) \end{pmatrix} & , t + \lambda_{ij}(t) \leq T \\ \begin{pmatrix} \infty \\ \infty \end{pmatrix} & , t + \lambda_{ij}(t) > T \end{cases} \quad ; \ \forall (i,j) \in A$$

1 Assign the label to each node for all $t \in \{0, \ldots, T\}$

$$\Pi_i(t) = \begin{cases} \left\{ \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} ; (t,t); \emptyset \right) \right\} & , i = d \\ \emptyset & , \text{ otherwise} \end{cases}$$

Set $SE := \{d\}$.
Set $ps := 0$; $ls := d$.

2 Take the *current node* j from the head of SE and
set $SE := SE - \{j\}$.

3 Scan the current node
For all $(i,j) \in A$ do
{ For $t_{ready} = 0$ to T do
{ For all $t_{depart} \in Depart_i(t_{ready})$ do
{ If $\Pi_j(t_{depart} + \lambda_{ij}(t_{depart})) \neq \emptyset$, then calculate
$\{tmpLabel\} := \Pi_j(t_{depart} + \lambda_{ij}(t_{depart}) + \big(c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})\big)$
$\Pi_i(t_{ready})_{new} = Merge\big(\Pi_i(t_{ready}), \{tmpLabel\}\big)$
If $\Pi_i(t_{ready})_{new} \neq \Pi_i(t_{ready})$, then
Set $\Pi_i(t_{ready}) := \Pi_i(t_{ready})_{new}$
Set $SE := SE + \{i\}$
}
}
}

4 If $j = ls$ and $SE \neq \emptyset$, then
$ps := ps + 1$; $ls := tail(SE)$
If $ps = (n-1)(T+1) + 1$ or a dynamic cycle is found,
then there is a negative dynamic cycle. Terminate the algorithm.
If $SE = \emptyset$, then go to Step 5.
Return to Step 2.

5 For any $i \in N$ and any $t \in \{0, \ldots, T\}$, the set $PO(\mathbb{P}_{id}(t))$ is obtained by forwardtracking the successor pointers of $\Pi_i(t)$.
Terminate the algorithm.

Theorem 6.3 (Correctness of the Algorithm 6.4) *Algorithm* 6.4 *generates all Pareto optimal paths from every node in the network to the sink and for all ready times* $t \in \{0, \ldots, T\}$ *in a finite number of steps or it detects a negative dynamic cycle.*

Proof:
If a dynamic cycle is found or $ps = (n-1)(T+1)+1$, then by Lemmas 6.1 and 6.2, the algorithm detects the existence of a negative dynamic cycle. On the other hand, if this is not the case, then the contrapositive of Lemma 6.2 guarantees that Algorithm 6.4 terminates (because the SE list is empty) at most $(n-1)(T+1)$ passes. Upon termination, for all $i \in N$ and $t \in \{0, \ldots, T\}$, every $\pi_i(t) \in \Pi_i(t))$ defines a Pareto optimal path. Suppose $\exists\ \hat{\pi}_i(t_{ready}) \in \Pi_i(t_{ready})$ with

$$\big(c_{ij}(t_{depart}) \oplus H_i(t_{ready}, t_{depart})\big) + \hat{\pi}_j(t_{arrival}) < \hat{\pi}_i(t_{ready})$$

for some $\hat{\pi}_j(t_{arrival}) \in \big\{\pi_j(t_{arrival}\big\}$.
Since the scanning process in step 3 is done for all $(i,j) \in A$, for all $t_{ready} \in \{0, \ldots, T\}$, and for all $t_{ready} \in Depart_i(t_{ready})$, the condition above implies that node j was not completely scanned yet and must still be in the SE list, contradicting the termination of the algorithm. ■

The following example illustrates Algorithm 6.2.

Example 6.6
Consider the problem in Example 6.5 (Figure 6.7), where we only change the data of arc $(3,4)$ to $\lambda_{34} := -1$ and $c_{34}(t) = \begin{pmatrix} 2 \\ -2 \end{pmatrix}$, $t \leq 8$, respectively.
We obtain

$$\begin{aligned} PO(\mathbb{P}_{05}(0)) \;=\; & \big\{\{0(0,0), 2(2,2), 3(4,5), 4(4,4), 5(5,5)\}, \{0(0,0), 1(1,1), \\ & 3(2,2), 5(4,4)\}\big\} \end{aligned}$$

with the corresponding optimal values $\begin{pmatrix} 6 \\ 3 \end{pmatrix}$ and $\begin{pmatrix} 5 \\ 9 \end{pmatrix}$. □

6.6 Computational Results

A set of experiments are conducted to compare the performance of the three algorithms discussed in the previous sections. In computational testing, again we rely on the representative operation counts (see Appendix C).

- to identify the asymptotic bottleneck operations,
- to estimate the running time for different problems sizes, and
- to obtain a fair comparison

of the three Algorithms 6.1, 6.2, and 6.4. To reach these goals, a series of experiments is run on the basis of randomly generated dynamic networks. All travel times and costs are nonnegative since only one of the three algorithms can deal with negative attributes. However, separate experiments with negative attributes are also run to analyze the performance of Algorithm 6.4 when it deals with negative attributes. For the experiments, Algorithms 6.1, 6.2, and 6.4 are implemented in C++ and run on a PC Pentium III, 500 MHz, and RAM of 256 MB.

To conduct these experiments, a dynamic random network generator is developed by extending the idea of NETGEN (Klingman, Napier, and Stutz [KNS74]) to include the time-dependent attributes (see Appendix B). 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 have zero indegree and zero outdegree, respectively. This degree setting implies that the generated networks have $2n$, $4n$, $6n$, and $8n$ arcs. We denote by $\delta = m/n$ the density of the network. The minimum travel time is always set to one. The maximum travel time and costs of both criteria are set to 10 units. The maximum holding cost is set to 10. For each specific setting of n and m, we test five random dynamic networks. Therefore, the total number of observations is 80.

Algorithm 6.1

Here we identify the following set of representative operations:

(a) sorting process

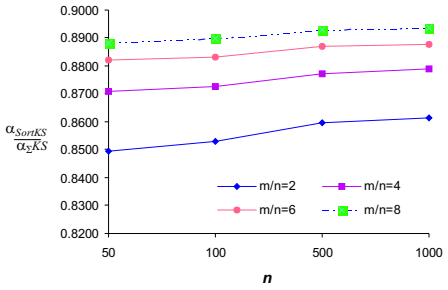
(b) finding the nondominated vectors among the sorted labels vectors

(c) convergence testing

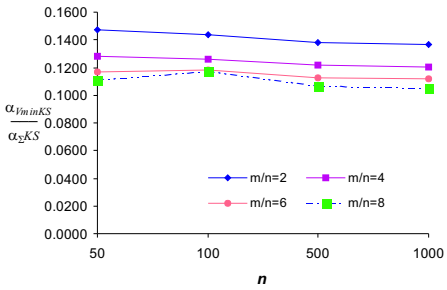
We denote by $\alpha_{sortKS}(I)$, $\alpha_{vminKS}(I)$, and $\alpha_{convKS}(I)$, the number of comparisons done in representative operations (a), (b), and (c), respectively, of a problem instance I. Let

$$\alpha_{\sum KS}(I) = \alpha_{sortKS}(I) + \alpha_{vminKS}(I) + \alpha_{convKS}(I)$$

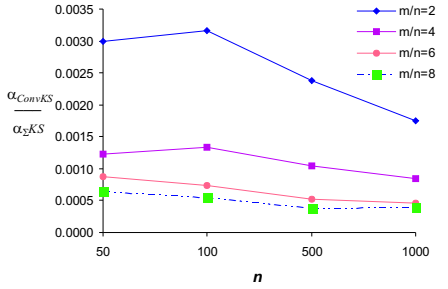
denote the sum of the representative operation counts. Figure 6.9(a) - (c) give the plots of $\alpha_{sortKS}(I)/\alpha_{\sum KS}(I)$, $\alpha_{vminKS}(I)/\alpha_{\sum KS}(I)$, and $\alpha_{convKS}(I)/\alpha_{\sum KS}(I)$ for increasingly larger problem instances and look for a trend. While, the trend on CPU time is shown in Figure 6.9(d). A plot for each different network density δ is given to help us in visualizing the effects of n and m on the growth of either representative operation counts or CPU time. The plot in Figure 6.9(a) suggests that *the sorting process is an asymptotic bottleneck operation* in Algorithm 6.1, and the plots in Figure 6.9(b) - (c) suggest that the other two representative operations are *asymptotic nonbottleneck* ones. Moreover, Figure 6.9(d) shows that the CPU time increases from linear to exponential growth in denser networks.



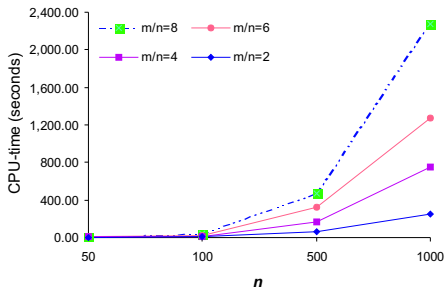
(a) A plot of the ratio of label comparisons in sorting operation to the total number of operations



(b) A plot of the ratio of label comparisons in finding the nondominated vector operations to the total number of operations



(c) A plot the ratio of label comparisons in convergence testing operation to the total number of operations



(d) CPU time (in seconds)

Figure 6.9: Identifying the asymptotic bottleneck operations and CPU time of Algorithm 6.1 for various n and densities m/n

To estimate the growth rate of the count of asymptotic bottleneck operation α_{sortKS}, we define an estimator function $\alpha'_{sortKS}(I) = c_{KS} n^{e_{1KS}} \delta^{e_{2KS}}$ for some choices of constants c_{KS}, e_{1KS}, and e_{2KS}. The regression analysis yields

$$\alpha'_{sortKS}(I) = 13,174.435\ n^{1.218} \delta^{1.418}$$

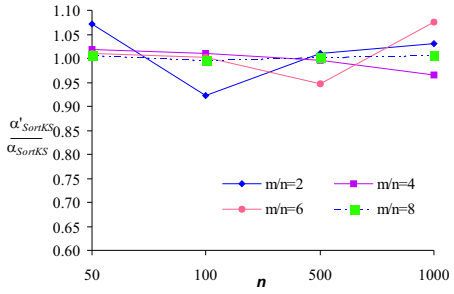
as a best fit for $\alpha_{sortKS}(I)$ with adjusted R^2 value 0.997 and standard error 0.139. The value of adjusted R^2 which close to 1 and a fairly small standard error indicate that the fit is indeed very good. Figure 6.10(a) shows a plot of the ratio between the estimated and the true values which also support the quality of the fit obtained by the regression analysis. Moreover, we obtain the virtual running time of an instance I, $V_{KSlinear}(I)$, as a linear estimate function of CPU time, as follows:

$$V_{KSlinear}(I) = 4.3710^{-6}\ \alpha_{sortKS}(I) - 2.210^{-5}\ \alpha_{vminKS}(I) + 1.3210^{-4}\ \alpha_{convKS}(I)$$

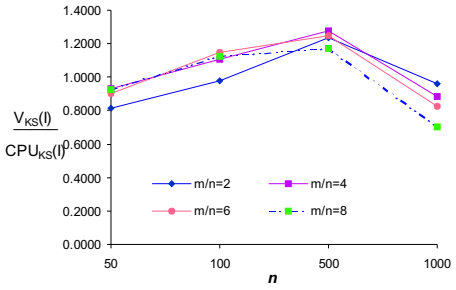
with adjusted R^2 value 0.927 and standard error 148.407. A better estimator of the CPU time is given by the following exponential function

$$V_{KS}(I) = 7.12510^{-7}\ \alpha_{sortKS}(I)^{-2.276}\ \alpha_{vminKS}(I)^{3.851}\ \alpha_{convKS}(I)^{-0.235}$$

with adjusted R^2 value 0.992 and standard error 0.260. The plot of the ratio $V_{KS}(I)/CPU_{KS}(I)$ is shown in Figure 6.10(b), where $CPU_{KS}(I)$ denotes the CPU time (in seconds) obtained by running Algorithm 6.1 for a problem instance I.

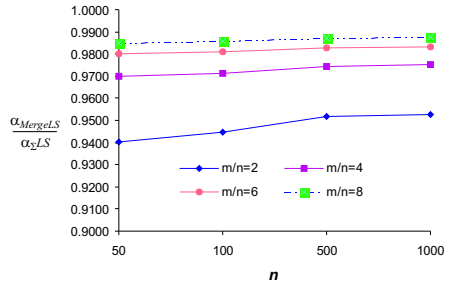


(a) The ratio of $13,174.435\ n^{1.218}\delta^{1.418}$ to α_{sortKS}

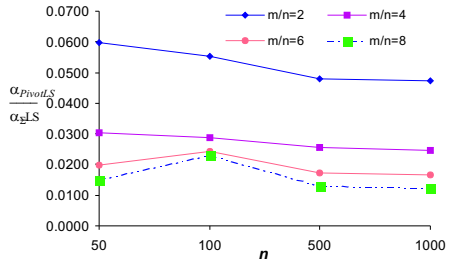


(b) The ratio of $7.12510^{-7}\ \alpha_{sortKS}(I)^{-2.276}\times$ $\alpha_{vminKS}(I)^{3.851}\ \alpha_{convKS}(I)^{-0.235}$ to the CPU time

Figure 6.10: Determine the quality of the estimators of performance functions of Algorithm 6.1



(a) A plot of the ratio of label comparisons in merging operation to the total number of operations



(b) A plot of the ratio of labels scanned in selecting the pivot label operation to the total number of operations

Figure 6.11: Identifying asymptotic bottleneck operations in Algorithm 6.2

Algorithm 6.2

We identify the following set of representative operations:

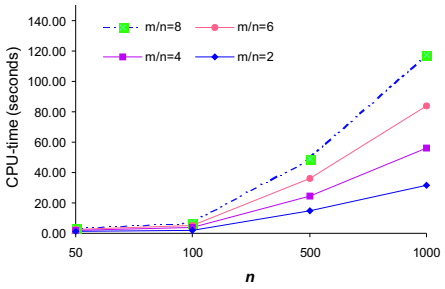


Figure 6.12: CPU time (in seconds) of Algorithm 6.2 for various n and densities m/n

(a) merging process

(b) selecting the pivot label

We denote by $\alpha_{mergeLS}$ and $\alpha_{pivotLS}$, the number of comparisons done in the merging process and the number of labels scanned in selecting the pivot label, respectively. Since the pivot label can be chosen from the head of the corresponding list of temporary labels in $\mathcal{O}(1)$, Theorem 6.2 implies that $\alpha_{pivotLS}$ determines the number of nondominated labels over all nodes in N and all ready times in $\{0, \ldots, T\}$. Figure 6.11(a) - (b) give the plots of $\alpha_{mergeLS}/\alpha_{\sum LS}(I)$ and $\alpha_{pivotLS}/\alpha_{\sum LS}(I)$ where $\alpha_{\sum LS}(I)$ denotes the sum of $\alpha_{mergeLS}$ and $\alpha_{pivotLS}$. These plots suggest that the *merging process is an asymptotic bottleneck operation* in Algorithm 6.2, and that selecting the pivot label is an asymptotic nonbottleneck operation. Figure 6.12 shows that CPU time increases faster in the denser networks.

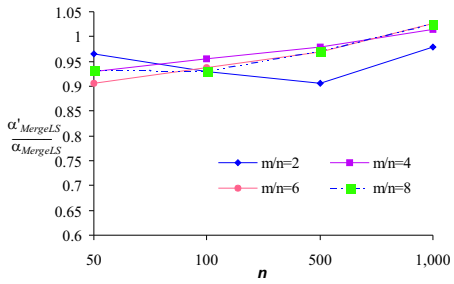
As in the case of Algorithm 6.1, we define an estimator function $\alpha'_{mergeLS}(I) = c_{LS} n^{e_{1LS}} \delta^{e_{2LS}}$ for some choices of constants c_{LS}, e_{1LS}, and e_{2LS}, to estimate the growth rate of the count of asymptotic bottleneck operation $\alpha_{mergeLS}$. The regression analysis yields

$$\alpha'_{mergeLS}(I) = 628.988\ n^{1.178} \delta^{1.270}$$

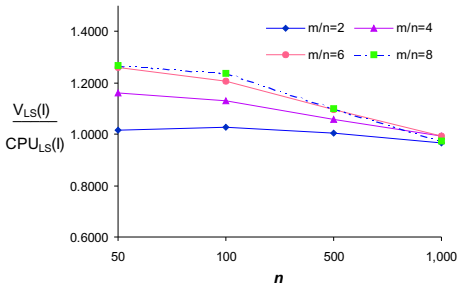
as a best fit for $\alpha_{mergeLS}(I)$ with adjusted R^2 value 0.938 and standard error 0.582. These statistic values and a plot of the ratio between the estimated and the true values in Figure 6.13(a) indicate that the fit is indeed very good. The virtual running time of an instance I, $V_{LS}(I)$, as a linear estimate function of CPU time, as follows:

$$V_{LS}(I) = 3.2310^{-6}\ \alpha_{mergeLS}(I) + 5.0310^{-5}\ \alpha_{pivotLS}(I)$$

with adjusted R^2 value 0.984 and standard error 1.923. The plot of ratio $V_{LS}(I)/CPU_{LS}(I)$ is shown in Figure 6.13(b), where $CPU_{LS}(I)$ denotes the CPU time (in seconds) obtained by running Algorithm 6.2 for a problem instance I. The estimates seems to be moderately good for small values of n and excellent for larger values of n.



(a) The ratio of $628.988\, n^{1.178}\delta^{1.270}$ to $\alpha_{mergeLS}$



(b) The ratio of $3.2310^{-6}\ \alpha_{mergeLS}(I) + 5.0310^{-5}\ \alpha_{pivotLS}(I)$ to the CPU time

Figure 6.13: Determine the quality of the estimators of performance functions of Algorithm 6.2

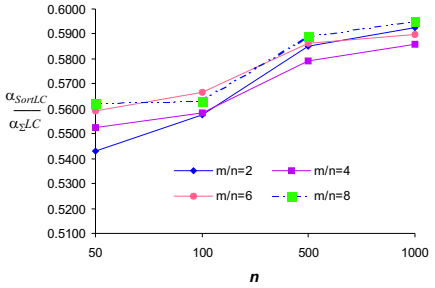
Algorithm 6.4

Here we identify the following set of representative operations:

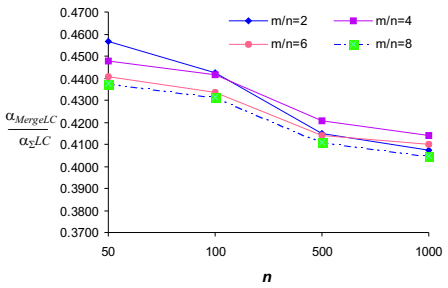
(a) sorting process

(b) merging process of two sorted set of labels

We denote by α_{sortLC} and $\alpha_{mergeLC}$, the number of comparisons done in the sorting process and merging process, respectively. Figure 6.14(a) - (b) give the plots of $\alpha_{sortLC}/\alpha_{\sum LC}(I)$ and $\alpha_{mergeLC}/\alpha_{\sum LC}(I)$ where $\alpha_{\sum LC}(I)$ denotes the sum of α_{sortLC} and $\alpha_{mergeLC}$. These plots suggest that the *sorting process is an asymptotic bottleneck operation* in Algorithm 6.4, and that the merging process of two sorted set of labels is an asymptotic nonbottleneck operation. Figure 6.15 shows that CPU time increases faster in the denser networks.



(a) A plot of the ratio of label comparisons in sorting operation to the total number of operations



(b) A plot of the ratio of label comparisons in merging operation to the total number of operations

Figure 6.14: Identifying asymptotic bottleneck operations in Algorithm 6.4

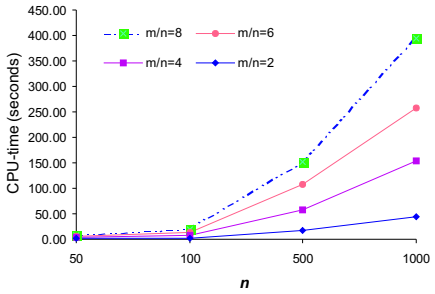


Figure 6.15: CPU time (in seconds) of Algorithm 6.4 for various n and densities m/n

Here we define an estimator function $\alpha'_{sortLC}(I) = c_{LC} n^{e_{1LC}} \delta^{e_{2LC}}$ for some choices of constants c_{LC}, e_{1LC}, and e_{2LC}, to estimate the growth rate of the count of asymptotic bottleneck operation α_{sortLC}. The regression analysis yields

$$\alpha'_{sortLC}(I) = 1,380.481\ n^{1.221} \delta^{1.562}$$

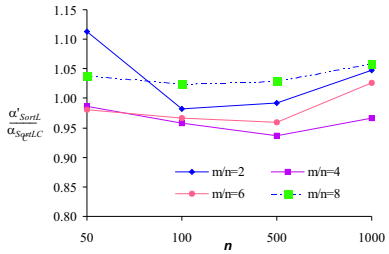
as a best fit for $\alpha_{sortLC}(I)$ with adjusted R^2 value 0.997 and standard error 0.134. These statistic values and a plot of the ratio between the estimated and the true values in Figure 6.16(a) indicate that the fit is indeed very good. The virtual running time of an instance I, $V_{LC}(I)$, as a linear estimate function of CPU time, as follows:

$$V_{LC}(I) = 4.410^{-6}\ \alpha_{sortLC}(I) - 2.710^{-6}\ \alpha_{mergeLC}(I)$$

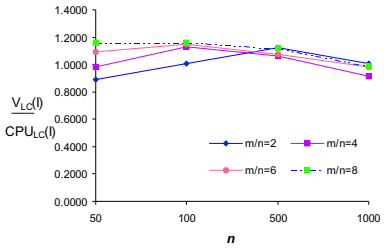
with adjusted R^2 value 0.982 and standard error 8.114. The plot of the ratio $V_{LC}(I)/CPU_{LC}(I)$ is shown in Figure 6.16(b), where $CPU_{LC}(I)$ denotes the CPU time (in seconds) obtained by running Algorithm 6.4 for a problem instance I. The estimates seems to be moderately good for small values of n and excellent for larger values of n.

Comparison

The results of the experiments with respect to the CPU time are given in Table 6.4. Figure 6.17(a) constrasts the performance of Algorithms 6.1, 6.2, and 6.4, with respect to the CPU time. The plots of the ratio $\alpha_{\sum LS}(I)/\alpha_{\sum LC}(I)$, $\alpha_{\sum LC}(I)/\alpha_{\sum KS}(I)$, and $\alpha_{\sum LC}(I)/\alpha_{\sum KS}(I)$ for some instances I are shown in Figure 6.17(b) - (d). From these figures, we can conclude that Algorithm 6.2 is *asymptotically superior to the other two algorithms* when the network attributes are nonegative. However, Algorithm 6.4 is able to deal with negative attributes, an advantage that the other two algorithms do not have. Furthermore, Algorithm 6.4 is also asymptotically superior to Algorithm 6.1.



(a) The ratio of $1,380.481\ n^{1.221}\delta^{1.562}$ to $\alpha_{sortLC}(I)$



(b) The ratio of $4.410^{-6}\ \alpha_{sortLC}(I) - 2.710^{-6}\ \alpha_{mergeLC}(I)$ to the CPU time

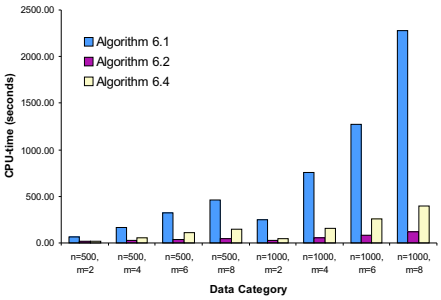
Figure 6.16: Determine the quality of the estimators of performance functions of Algorithm 6.4

n	m/n	Average CPU time (in seconds)		
		Algorithm 6.1	6.2	6.4
50	2	2.66	1.01	1.05
	4	6.00	1.64	3.21
	6	6.24	2.29	5.67
	8	14.47	3.00	7.92
100	2	7.37	2.24	2.57
	4	14.61	3.65	7.41
	6	24.63	5.18	12.96
	8	36.98	6.90	19.25
500	2	62.76	14.61	17.82
	4	164.99	24.42	57.73
	6	321.29	35.83	107.75
	8	464.51	48.69	152.03
1000	2	251.85	31.41	44.81
	4	756.21	55.99	154.17
	6	1,269.37	83.87	257.73
	8	2,275.02	117.09	394.18

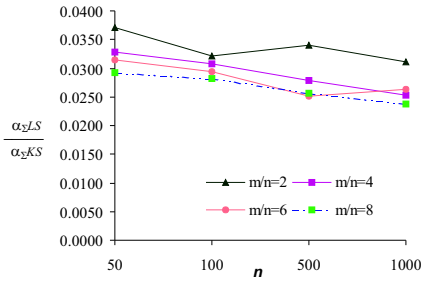
Table 6.4: Computational test results

Allowing negative data

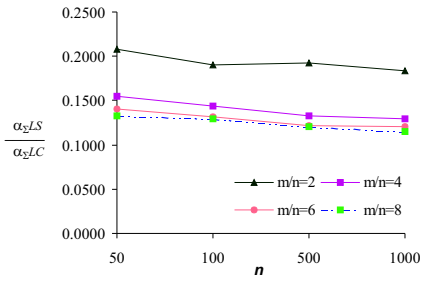
For the case of arbitrary attributes, the minimum travel time and costs of both criteria are always set to the negative value of their corresponding maximum values. The maximum value of the attributes are set as in the case of nonnegative attributes. To avoid the negative dynamic cycle, the network is assumed acyclic. In this case, networks with *a topological order* are generated.



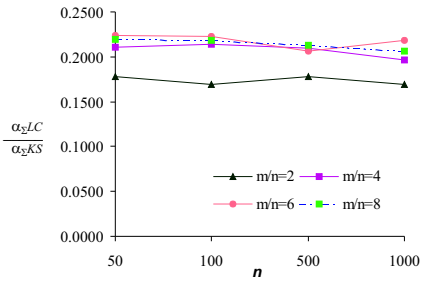
(a) CPU time (in seconds)



(b) A plot of the ratio of total number of representative operations in Algorithm 6.2 to that in Algorithm 6.1



(c) A plot of the ratio of total number of representative operations in Algorithm 6.2 to that in Algorithm 6.4



(d) A plot of the ratio of total number of representative operations in Algorithm 6.4 to that in Algorithm 6.1

Figure 6.17: Comparing Algorithms 6.1, 6.2, and 6.4

To control the distribution of attributes with negative values, we set the percentage of negative data to 0, 5 and 10 percents. The results of experiment with respect to the CPU time from 10 random networks with size $n = 500$ and density $\delta = 8$ is shown in Table 6.5. It is found that the *CPU time increases as the percentage of negative data increases.* Moreover, the *CPU time tends to be unstable as the percentage of negative data increase*, as shown in Figure 6.18.

percentage of negative data	0%	5%	10%
average of CPU time (seconds)	104.451	2,956.273	8,995.864
standard deviation	3.726862	757.5691	3,126.029

Table 6.5: Computational test results of Algorithm 6.4 for three sets of networks with $n = 500$, $m/n = 8$, and three different percentages of negative data

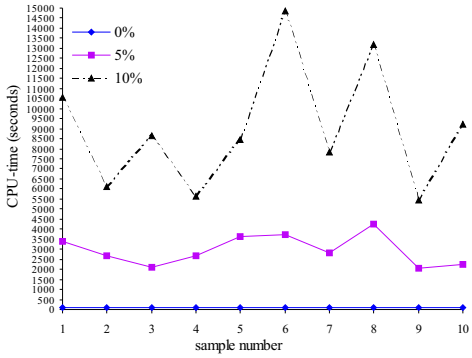


Figure 6.18: The CPU time of Algorithm 6.4 for ten random samples with $n = 500$, $m/n = 8$, and three different percentages of negative data

Chapter 7

Application to Evacuation Problems

7.1 Introduction

Some critical issues may come together with the development of inhabited areas such as, city, office buildings, condominiums, passenger aircrafts, ships, and trains. One such issue is the question of *how to evacuate a large number of threatened people in a minimum time and with utmost safety during an emergency situation.* There are a number of reasons why an emergency evacuation is necessary. Threat of fire or/and smoke, bomb threat, toxic gas leak, and earthquake may trigger an emergency evacuation. This evacuation issue underlines the need for analysis of the design of such areas to obtain an idea of *how and what amount of time an emergency evacuation can be accomplished.* Unfortunately, it is not possible to test the design under some realistic emergency scenario. It is highly costly and unethical to expose people to real emergencies. Therefore, some careful theoritical analysis must be done to estimate the evacuation performances. The *number of fatalities, number of safe evacuees, and evacuation time* are typical and important performance measures of an evacuation process (see e.g., Løvås [Løv95]).

Evacuation planning as a part of the emergency management is a complex problem that must take into account many aspects including

- location planning of emergency facilities such as emergency shelters and ambulances (see e.g., Sundstrom, Blood, and Matheny [SBM95] and Matheny, Keith, Sundstrom, and Blood [MKSB97]),

- analysis of evacuee's behavior in a panic situation (see e.g., Sandberg [San97] and Helbing, Farkas, and Vicsek [HFV00])

- design of evacuation facilities, such as exit or emergency stairs, exit signs, and the design of a directional sound to guide people towards the nearest exit (see e.g., Tanaka, Hagiwara, and Mimura [THM96] and Withington [Wit02])

- analysis of the hazard propagation such as a propagation of smoke and fire in a building (see e.g., Klote [Klo95] and Ebihara, Notake, and Yashiro [ENY96])
- analysis of evacuee's movement distribution to determine the evacuation time

In this thesis *we focus the discussion on the analysis of evacuee's movement distribution to determine the evacuation time.*

In general, there are two approaches used to model the evacuees distribution in order to determine the evacuation time, namely *micro- and macroscopic models.* A survey of both approaches on evacuation problems was reported in Hamacher and Tjandra [HT02]. In microscopic models, each evacuee is considered as a separate flow object and provided with some personal attributes, for example, walking speed, personal memory and psychological condition. These attributes will be used to determine the movement decisions, for example to select the nearest walkway, move on the walkway only when there is no blockage at the end, or change the destination target before reaching it. The psychological responds, such as panic and hysterical during the catastrophe and or life-threatening situations (e.g., fire in a crowded building) may significantly influence the movement decisions and crowd behaviours (see e.g., Helbing, Farkas, and Vicsek [HFV00]). The main tool of this approach is simulation and it has been used by several research groups, for example Ebihara, Ohtsuki, and Iwaki [EOI92], Doheny and Fraser [DF96], Owen, Galea, and Lawrence [OGL96], Gwynne, Galea, Owen, Lawrence, and Filippidis [GGO+99], Helbing, Farkas, and Vicsek [HFV00], and Klüpfel, König, Wahle, and Schreckenberg [KKWS00].

On the other hand, macroscopic models are mainly based on optimization approaches and do not consider individual differences and decisions for selecting egress routes (egress is *means of going out*). Occupants are treated as a homogeneous group where only common characteristics are taken into account. These models are mainly used to produce good lower bounds for the evacuation time. Since the time is a decisive parameter in an evacuation process, most macroscopic approaches are based on dynamic network flow models, see for example, Chalmet, Francis, and Saunders [CFS82], Kisko, and Francis [KF85], Choi, Francis, Hamacher, and Tufekci [CFHT88], Fahy [Fah91], Kostreva, and Wiecek [KW93], and Montes [Mon94]. The common idea of these models is to represent an inhabited area in a static network $G_{stat} = (N, A)$. The modeling of evacuation over time is then done in the network $G = (N, A, T)$ which is a dynamic extension of G_{stat}. In this dynamic network modeling, dynamic network flows correspond to the evacuation flows.

The *evacuation time*, that is a time needed to complete an evacuation process, basically consists of three main time components (Graat, Midden, and Bockholts [GMB99], Løvås [Løv98])

1. The time evacuees need to recognize a dangerous situation. It is called the *awareness time* and influenced mainly by the reliability of the alarm system and the familiarity of evacuees with emergency signals.

2. The time evacuees need to decide which course of action to take. It is called the *preparation time* and influenced by the experience of evacuees in facing an emergency situation. This can, for instance, be generated through some emergency practice and training. Behavioral and organizational factors are the main contributors to the duration of this time component.

3. The time evacuees need to move towards the safety area, which is known as the *egress time*. It is influenced by the availability of emergency exit signs, well planned evacuation procedures, constructional factors (e.g., effective width of walkway, slope of stairs), and human behavior during panic situations.

Since the behavioral and organizational factors are the main contributors to the first two time components, it is hard to predict analytically the duration of those time components. Therefore, most evacuation models *emphasize the calculation of egress time and treat the result as a lower bound of the real evacuation time.*

In this chapter we focus the discussion on the macro modeling of the evacuation problems using a dynamic network. In the next section we describe a network model of evacuation objects. The need of evacuation modeling with time-dependent attributes is described in Section 7.3. In Section 7.4 we discuss how to apply the theoritical results, discussed in the previous chapters, to find the optimal flow distributions and optimal evacuation paths with respect to the egress time. The computational results of dynamic network models applied to the evacuation plan of Building 42 at the University Kaiserslautern, Germany, in Section 7.5 conclude this chapter.

7.2 Network Model of Evacuation Objects

A dynamic network model of evacuation objects has the following components:

a. *Time horizon and basic unit*
The time horizon T is broken up into finite uniform time periods $t = 0, 1, \ldots, T$. As we have discussed in Section 2.1, the time period t depends on a basic unit θ in which travel times are measured. Hence, if we choose 5 seconds as the length of the basic unit (i.e., $\theta = 5$), then time period $t = 3$ associates with a real time 15 seconds. A time horizon $T = 12$ corresponds to one minute of real time in this scenario.

b. *Node*
Nodes are used to model the connected points and/or locations. There are some alternatives to consider in defining a node. For example, a node may represent a single room (or hall, stair, lobby, etc.) in a building or ship evacuation, a seat or a row of seats in a passenger aircraft evacuation, and a train compartment in a train evacuation. It may also represent an intersection point between two crossed walkways. In a detail modeling, a room can be split up into several nodes. Furthermore, each node in the network has a *capacity*, denoted by a, which represents the maximum

number of evacuees can safely occupy a node at one time. Actually many factors play important roles in determining the node capacity, including the type of room or building represented by the node itself. In a swimming pool, for example, the number of people must be limited, even before the consideration of an emergency situation, in order to prevent accidents. Personal comfort and convenience as well as weight limitations should be also taken into account. These considerations are used to determine the so-called *occupant load factor* which represents *the minimum required area per person.* The node capacity can thus be determined, for instance, by

$$\text{node capacity} \quad := \quad \min \left\{ \frac{\text{floor space area}}{\text{occupant load factor}}, \frac{\text{maximum allowable weight}}{\text{average weight per person}} \right\}$$

Some examples of occupant load factors are listed in Table 7.1.

Object type	Occupant load factor in $\frac{square\ meters}{persons}$
University auditoriums	0.63
Classrooms	1.8
Swimming Pools	4.5 for pool area and 1.35 for the deck
Library	4.5 for reading rooms and 9 for stacks
Dormitories (residential halls)	4.5

Table 7.1: Some occupant load factor (see table 10-A of the Uniform Building Code [ICB]).

c. *Arc*

Arcs can be used to model corridors, hallways, stairways, streets, or a connection between two intersection nodes. An arc connects two center points of locations which are considered as nodes. Its direction is determined by the expected or possible direction of evacuees movement. An arc may have several attributes, for example *flow capacity*, *travel distance* and *travel time.* Arc travel time, denoted by λ, is determined by the physical distance and the travel speed. If θ is defined by 5 seconds and the travel time is measured 13 seconds long, then λ is determined as three time units long, since

$$\lambda := \left[\frac{13}{5} \right] = 3$$

The physical distance of an arc object (e.g., corridor, stair, street) can be measured as a distance between the center of the two locations connected by this arc. The speed itself is actually depended on the density. When the density is low, any speed

can be chosen, but when the density is high, individuals will have to conform their speed to the speed of the mass. When a walkway becomes more crowded, people tend to slow down their pace in order to keep a comfortable distance between themselves and others. Fruin [Fru71] gave several levels of crowd ranging from less than 0.39 *persons per meter width-second* (PMS) to more than 1.39 PMS which corresponding to the average travel speed ranging from 1.3 m/s for minimum crowd to less than 0.30 m/s for crushing crowds. These data are given in Table 7.2. A crowd level data can be used to determine the arc flow capacity (denoted by u) which represent the maximum number of evacuees that can traverse the corresponding arc per unit time. For example, in a hallway of 3 meters width, the crowd level D determines that the number of people who can pass any point of interest in the hallway during a second is between 2.49 and 3.33. If θ is defined by 5 seconds, then the arc capacity will be between 12 and 17 person per time unit.

d. *Source and sink nodes*
Some locations which house a significant number of evacuees are considered to be the source nodes in a network. The supply of a source node is given by the number of evacuees in the location associated with the node. The safety locations that might be considered as the final destinations of evacuees movement, are considered as the sink nodes. In the evacuation problem, we have only one sink node by connecting all the exit nodes to one artificial node and assign the total number of evacuees as the demand value of this node. Hence, evacuation problems can be modeled as *multi-sources single-sink dynamic network flow problems.*

Example 7.1
Figure 7.1 is a plan section of the first floor of a building. This floor section has eleven rooms and a long corridor. It is connected to the building's exit door at the second floor by a stairwell exit. Another building's exit door is located at the right end of the corridor (not shown in the picture). Table 7.3 is an example of how to define the nodes for this floor. Node d is added into the network as the sink node which represent *the common safety area.*

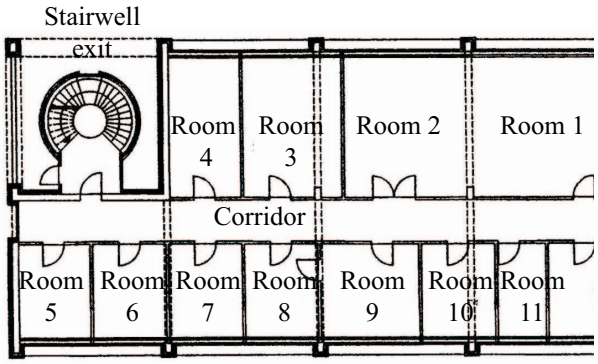
To define arcs of the network, consider a situation in which some persons are located in room 2 just before the announcement of an evacuation. As soon as they find out that there will be an evacuation, they run out from their room and step into the corridor area modeled by node C4. Then, some of them may choose to move to C3 since they want to go out via the building's exit door at the right end of the corridor while the others choose to move to C5 since they want to go out via the building's exit door at the second floor. Similar situations are considered for other rooms. These movement possibilities determine the direction of arcs in the network model. Table 7.4 lists the arcs of the floor plan example. □

Crowd level	Average Speed ($m/seconds$)	Description
A	> 1.3	Level of crowd : 0.39 PMS or less. Virtually unrestricted choice of speed; minimum maneuvering to pass; crossing and reverse movements are unrestricted; flow is approximately 25% of maximum capacity.
B	1.25 - 1.3	Level of crowd : 0.39 - 0.56 PMS. Normal walking speeds only occasionally restricted; some occasional interference in passing; crossing and reverse movements are possible with occasional conflict; flow is approximately 35% of maximum capacity.
C	1.15 - 1.25	Level of crowd : 0.56 - 0.83 PMS. Walking speeds are partially restricted; passing is restricted but possible with maneuvering to avoid conflict; flow is reasonably fluid and is about $40-65\%$ of maximum capacity.
D	1.0 - 1.15	Level of crowd : 0.83 - 1.11 PMS. Walking speeds are restricted and reduced, passing is rarely possible without conflict; crossing and reverse movements are severely restricted with multiple conflicts; some probability of momentary flow stoppages when critical densities might be intermittently reached; flow is approximately $65-80\%$ of maximum capacity.
E	0.55 - 1.0	Level of crowd : 1.11 - 1.39 PMS. Walking speeds are restricted and frequently reduced to shuffling; frequent adjustment of gait required; passing is impossible without conflict; crossing and reverse movements are severely restricted with unavoidable conflicts; flows attain maximum capacity under pressure, but with frequent stoppages and interruptions of flow.
F	0 - 0.55	Level of crowd : 1.39 PMS or more. Walking speed is reduced to shuffling; passing is impossible; crossing and reverse movements are impossible; physical contact is frequent and unavoidable; flow is sporadic and on the verge of complete breakdown and stoppage.

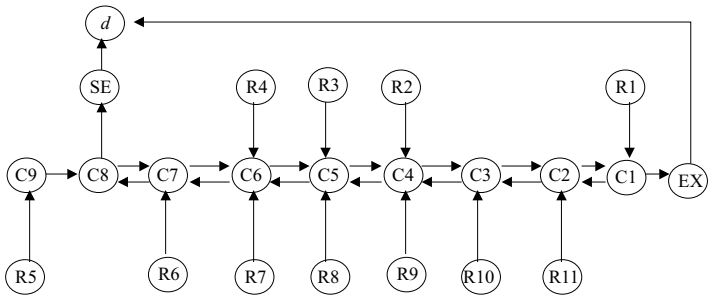
Table 7.2: Travel speed for different crowd levels, Fruin [Fru71]

7.3 The Need of Time-Dependent Modeling

During an evacuation process, a connection between two nodes may only be temporary due to, for instance, blocking by fire or smoke. In this case, an arc that represents a



(a) The floor plan section of the example building



(b) The corresponding network representation

Figure 7.1: The floor plan and its corresponding network representation for Example 7.1

Node specification	Area associated with the node
R1 - R11	Room 1 - Room 11
C1, ..., C9	Areas in the long corridor in front of the room's exit doors or the entrance door to the stairwell exit
SE	The stairwell exit
EX	The building's exit door at the right end of the corridor
d	An additional node as the common safety area

Table 7.3: Node definition of the floor section plan

connection must also be temporary, i.e., the arc capacity decreases over time and becomes zero after some times. Furthermore, increasing crowd and some other physical barriers may also increase the travel times. Therefore, the time needed to travel from one node to another is highly likely to be time-varying. The following *eyewitness report from the 11 September 2001 disaster at the World Trade Centre, New York, USA*, highlights the need

Arc specification	Arc definition
(Ri, Cj)	Passageway from the room i to the corridor area j in front of the exit door of room i
(Ci, Cj)	Passageway in the corridor between area i and j
(C8, SE)	Passageway via stairwell from the left end of corridor to the exit at the second floor
(C1, EX)	Passageway in the corridor to the the exit at the right end
(SE, d), (EX, d)	Passageway from the building exits to the common safety area

Table 7.4: Arc definition of the floor section plan

of evacuation modeling with time-dependent attributes.

> **Joe Crimmins, of Hoboken, N. J.**, was on the 43rd floor in the cafeteria of the World Trade Center tower hit by the first airplane. "There was an explosion" Crimmins said. "The building shook. Within seconds, you could see debris coming towards the window. So we just ran toward the emergency exit. It took about 20 minutes to a half hour to get out. The stairways were crowded and smoky. The lower 10 to 15 floors were filled with water, so we were walking through water as firemen were walking up."
>
> **A survivor security consultant, Joseph Gomez**, says: "Everyone made for the fire exits. There was screaming and panic. People were shoving and pushing."
>
> **Michael Hingson**, the 51-year-old has been blind since birth. Michael was on the seventy-eighth floor of the World Trade Center, the One building, the north tower. He was guided out by his guide dog Roselle and another colleague, Frank. "The crowds weren't huge at first," Hingson said. "But as we started making our way down, they got bigger. It was getting hot, too, with temperatures in the stairwell climbing higher than 90 degrees." Hingson was sweating and Roselle was panting. "We moved fairly swiftly until we hit about the 40th floor. Then things got kind of jammed, a lot of stopping and starting."

On the other hand, it is also possible to build some additional emergency connections at the later time (i.e., not starting from time zero) to expedite the movement of the occupants. For example, emergency exit by using a *ladder fire* or a helicopter.

Consequently, dynamic network flow models with time-dependent attributes are considered more suitable than those with constant attributes.

7.4 Solution Methods

Here we apply the time-dependent dynamic network flow models and their associated solution algorithms which are developed in the previous chapters to determine

1. the lower bound of the evacuation time,

2. the evacuation routes, and

3. the maximum capacity of inhabited area with respect to the safety requirements.

We say "lower bound" because we do not consider other time components like awareness time, preparation time, etc. (see Section 7.1).

7.4.1 Optimal Evacuees Distribution

The application looks at the evacuation of a building with either known or unknown number of occupancies. When the number of residents in a building is difficult to estimate (e.g., a public building), we are interested to find the maximum number of evacuees which can be sent to the safety area within a given time horizon T. Since it is required to save as many evacuees as possible and as quickly as possible, the time-dependent earliest arrival flow model (see Chapter 4) is considered more suitable than the maximum dynamic flow model. The model also assumes that the arc capacities and travel times may vary over time. These assumptions fulfill the requirement of an evacuation modeling as described in Section 7.3. Algorithm 4.4 can be used to obtain the following results.

- The optimal flow distribution $\mathbf{x}$ represents the flow distribution of evacuees within the time horizon T.

- The total flow $V_{\sum^{T'} \leq T}(\mathbf{x})$ bounds the number of people that can be safely evacuated within the T time units. Hence, the results of EAF model can be used to design the capacity of an inhabited area with respect to the safety requirements.

- The set of times Γ_{ij}^{T} (see (3.9) and (4.9)) determines how long a corridor or a street, etc., represented by an arc (i, j), is fully occupied within the time horizon T. If $\psi(K)$ denotes the cardinality of the set K, then an arc (i, j) is considered as the bottleneck if

$$\psi(\Gamma_{ij}^{T}) = \max\{\psi(\Gamma_{kl}^{T}) \ : \ (k, l) \in A\} \tag{7.1}$$

Sensitivity analysis can be conducted to see the effect of additional exit routes and decreasing/increasing the capacities of some routes due to e.g., increasing/decreasing of smoke intensity. The results can then be used to improve the structural design of the system under study.

When the initial occupancy is known, the *quickest flow model with time-dependent attributes* (see Chapter 5) can be used to obtain *the minimum evacuation time*. This model will find an optimal evacuees distribution $\mathbf{x}$ minimizing the total evacuation time T. Since the network evacuation model is a multiple source single sink network model, Algorithm 5.3 can be used to find the optimal T and the optimal evacuee distribution.

7.4.2 Multicriteria Evacuation Routes

During an evacuation process, an evacuee by taking a quick look at a scene in the panic situation, may consider not only the time required to travel along the route, but also whether the route is blocked by the presence of fire or smoke, availability/clarity of exit signs, and familiarity. One may choose a route (or path) with longer distance from his/her position to the safety area when he/she is more familiar to this route than to other routes. Furthermore, the preference value to some criteria on selecting the evacuation routes may also change over time due to some reasons. Some common reasons are psychological conditions (e.g., increasing panic and hysterical), structural changes (structural failure, additional emergency exits), increasing/decreasing of smoke intensity, and fire propagation. These multiple criteria which determine the movement decision are then considered as vector of cost attributes. We denote by $c_{ij}^k(t)$ the k-th component of the cost vector $c_{ij}(t)$ corresponds to the k-th criterion.

In the case of two criteria, Algorithm 6.2 or 6.4 can be applied to determine all possible evacuation routes for each occupant, from his/her initial location, which have the minimum cost (or Pareto optimal cost). However, these two algorithms can be extended to the case of more than two criteria.

7.5 Case Study

In this section we apply our algorithm to find the lower bound of the evacuation time of Building 42 in the University of Kaiserslautern, Germany.

Building 42 at the University of Kaiserslautern is a six stories building composed of lectures rooms, auditoriums, offices, halls, and library. Figure A.1 in Appendix A gives the first floor plan of Building 42. The other floors have similar plans, but without three auditoriums as in the first floor. This building is provided with a lift and two stairwell exits. However, it is assumed that during an emergency situation, the lift is not used. The basement is not considered because there are rarely people present. The network repre-

sentation of all floors are given in Appendix A which consists of 108 nodes (including the super source and super sink) and 198 arcs. Node 0 and node 107 is the super source and the super sink, respectively. Node 0 and 107 might be drawn several times in order to find a representation of the network such that the reader can follow the network connections easier. A node drawn by a circle surrounded by a rectangle describes a building object (e.g. room, stairwell, or exit) located in the floor one level above/below the current floor.

We define one time unit by 5 seconds and the time horizon T is set to 70 time units. The constant travel time and capacity data are obtained from Montes [Mon94] and attached in the network representation. Table 7.5 lists all nodes with positive initial contents of total 687 people. Algorithm 5.3 is used to find the minimum evacuation time. The solution is then compared to the one given by Montes in [Mon94]. Figure 7.2(a) and (b) shows the distribution of the evacuees and the cumulative number of evacuees, respectively.

Nodes	1	2	3	4	6	7-12	25	26-32	44
Initial contents	45	120	40	20	7	10 each	12	10 each	12

Nodes	45-51	64	66-72	85	87	88	89	90	95
Initial contents	10 each	12	10 each	12	10	20	7	30	5

Nodes	98-99	100
Initial contents	10 each	45

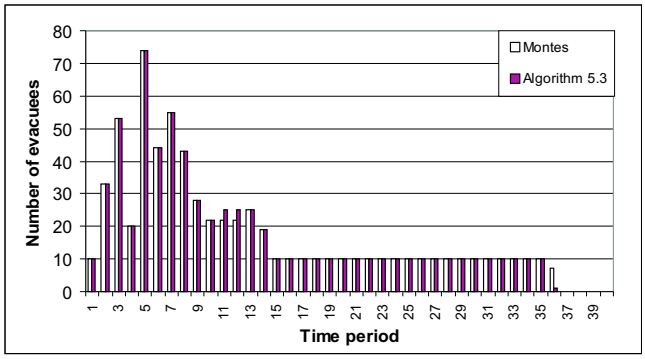
Table 7.5: Distribution of initial occupations of Building 42

Algorithm 5.3 yields 36 time units, which is equal to 180 seconds, as the evacuation time of 687 people. The same evacuation time is obtained by Montes [Mon94] by applying a min cost flow algorithm of Bertsekas [PB91] on the associated time-expanded network. However, Algorithm 5.3 *provides better flow distribution since it sends out more people in a shorter time*, as shown in Figure 7.2(a). The cumulative number of safe evacuees at every time $t \in \{12, \ldots, 35\}$ are 6 units bigger that the one provided by Montes. This is happened since Algorithm 5.2 *always maintains the earliest arrival property*, i.e. it sends flows as many as possible and as quickly as possible. Furthermore, Figure 7.2(c) shows that exit arcs $(25, 107)$ in the first floor and $(25, 107)$ in the second floor are most passed through. These arcs can be considered as the bottleneck arcs.

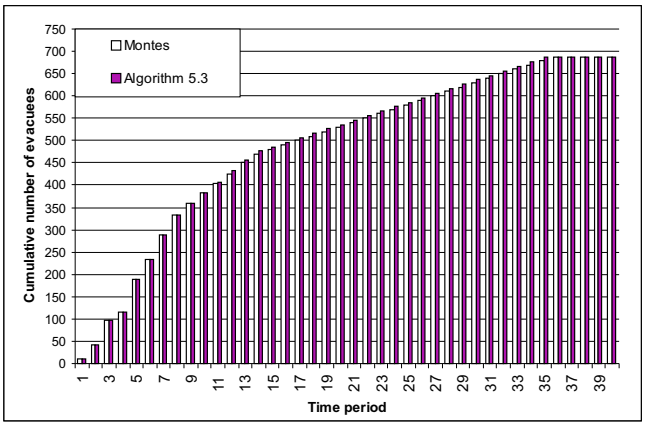
When the exit arc $(25, 107)$ is completely blocked after 2 time units (i.e. arc $(25, 107)$ has zero capacity for $t > 2$), Algorithm 5.3 shows that only 657 people are able to leave the building in 36 time units. The evacuation time of 687 people now increases to 39 time

units. See Figure 7.3(a) and (b).

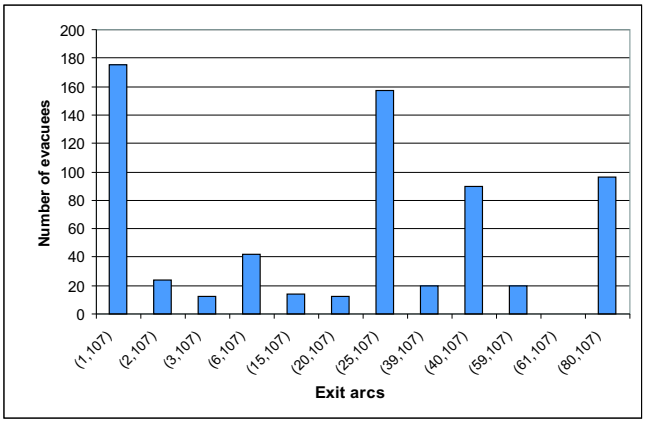
Concerning the running time, the results of this case study are obtained in **51.42 seconds** with a personal computer having 252 megabyte RAM and 500 megahertz.



(a) The distribution of evacuees

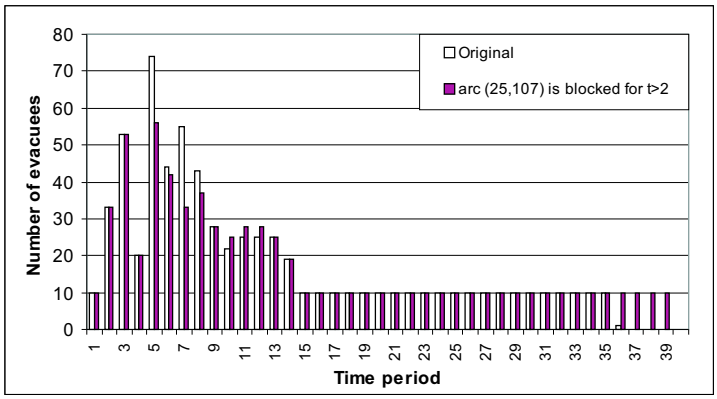


(b) The cumulative number of evacuees

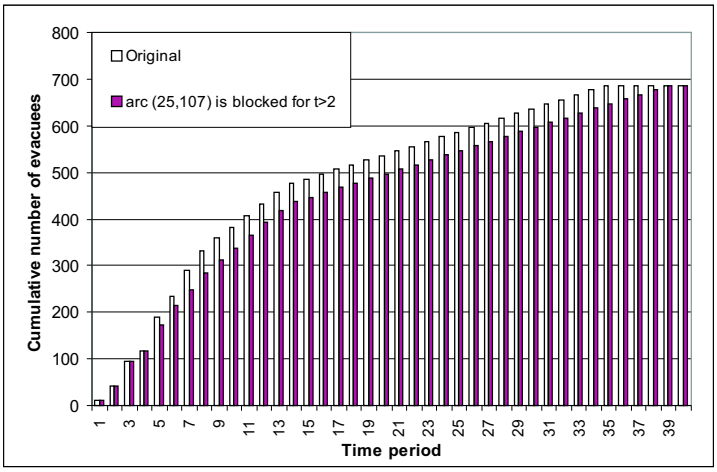


(c) The distribution of evacuees over all exit arcs

Figure 7.2: Comparing the results of Montes [Mon94] and Algorithm 5.3



(a) The distribution of evacuees



(b) The cumulative number of evacuees when the exit arc $(25, 107)$ in the second floor is completely blocked after 2 time units

Figure 7.3: The distribution of evacuees when exit arc $(25, 107)$ in the second floor is completely blocked for time $t > 2$

Chapter 8

Conclusions and Future Research

In this thesis we discussed dynamic network flow problems and their application to the evacuation problems. We reviewed some existing approaches of the dynamic network problems with constant attributes and gave a generalization of the hybrid capacity scaling and shortest augmenting path algorithm to consider the time dependency of the network attributes. We also developed a new, successive earliest arrival augmenting path algorithm, to solve an earliest arrival flow problem with time-dependent attributes. Under an assumption of infinite waiting, this algorithm is more efficient by factor T than implementing the successive static shortest path algorithm on the time-expanded network. The computational analysis showed that the scanning process is the asymptotic bottleneck operation of this algorithm. By modifying the network, we showed that this algorithm can be used to solve a quickest flow problem with time-dependent attributes. Even when the supply to the source vary with time. The modification is done by adding a dummy node as the super source and define the capacity of all arcs going out from this node by the initial contents or supply value.

Dealing with a discrete-time dynamic path, we developed a dynamic label setting and a dynamic label correcting algorithms for the bicriteria dynamic shortest path problems with time-dependent travel times, cost vectors, waiting times, and holding costs. We also compared the performance of these two new algorithms with the existing algorithm of Kostreva and Wiecek [KW93]. The computational analysis showed that in the case of positive valued data, the dynamic label setting algorithm is superior to the other two algorithms. However, the dynamic label correcting algorithm is able to deal with negative attributes, an advantage that the other two algorithms do not have. It is also asymptotically superior to the one of Kostreva and Wiecek. Furthermore, our dynamic label correcting algorithm, to the best of our knowledge, is the first algorithm designed to solve a bicriteria dynamic shortest path problem with unrestricted (in sign) time-dependent attributes.

As an application problem, we discussed the evacuation problems and showed how to use the solution algorithms developed for the dynamic network flow problems, to find optimal evacuee distributions and optimal evacuation paths. In particular, we apply our dynamic network flow algorithm to find a lower bound of the evacuation time of Building 42 in

the University of Kaiserslautern, Germany. The results is compared to the one given in Montes [Mon94]. It is found that our algorithm provides better flow distribution since it sends out more people in a shorter time. Moreover, our algorithm is able to account for the time-dependent nature of the evacuation problems, such as increasing travel times due to, for instance crowd and smoke, and decreasing or increasing the capacity of passageway. The properties which are not considered by Montes [Mon94].

In the rest of this chapter, we discuss some further research topics related to dynamic network flows and evacuation modeling.

Minimum cost dynamic flows. In this thesis we have not yet considered the minimum cost dynamic flow problem. If the supply is equal to the value of maximum dynamic flow for a given T, then the problem is called the minimum cost maximum dynamic flow problem. On the other hand, when T is equal to the quickest time to send the supply from the source to the sink, the problem is called the minimum cost quickest flow problem. One can construct an example showing that the temporally repeated flow (TRF) does not give a minimum cost dynamic flow. Klinz and Woeginger [KW95] showed that this problem is NP-hard, even for the network with constant attributes (also for series-parallel graphs). Consider now when the network attributes are constant. One can show that finding a minimum cost augmenting path in the residual dynamic network is a constrained static shortest path problem. Desrochers and Soumis [DS88] brought this constrained static shortest path problem as a multiobjective shortest path problem. They proposed a label-setting algorithm to solve this problem in which the travel time (or weight) and cost are considered together as a vector. The label setting algorithm, however, will not work in our case since the residual costs and travel times may be negative. Instead, we can use the static version of our label correcting algorithm in Chapter 6 to find the Pareto optimal $s-d$ augmenting paths in the static residual network having distance (with respect to the travel times) less than T. The minimum cost augmenting path is then given by the Pareto optimal path having the minimal residual cost. The interesting questions are when and how long we can repeat this restricted augmenting path within T. We know that TRF does not work, therefore we may start repeating this augmenting path not from time zero. Moreover, this augmenting path may contain some backward arcs. Therefore, to keep the feasibility, repeating the flow on backward arc (j,i) must be stopped at time $t+\lambda_{ij}$ when there is no positive flow on forward arc (i,j) at time t. The answer to these questions will determine the overal complexity of the algorithm.

Earliest arrival flows. In Chapter 4 we have discussed several algorithms for solving the earliest arrival flow problem. However those algorithms are either pseudo-polynomial time or approximation algorithms. There is no known polynomial time algorithm to solve the earliest arrival flow problem. It is an open problem to prove if the earliest arrival flow problem is an NP-hard problem.

Multicommodity-like dynamic flows. Consider an evacuation problem in which young, old, and handicapped people occupy the same area. In the dynamic network terminology, these differences lead to different commodities with different travel times. As in the classical multicommodity flow problem, these commodities will also share the arc and node capacities. But in an evacuation problem, different commodities may come from the same source and certainly have the same sink. The difference on the travel time leads to a different time-expanded structure of the dynamic network for each commodity. It is an interesting open problem to model this problem as a dynamic network flow problem.

Bibliography

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[APP82] E.J. Anderson, Nash P., and A.B. Philpott. A class of continuous network flow problems. *Mathematics of Operation Research*, 7:501–514, 1982.

[Aro89] Jay E. Aronson. A survey of dynamic network flows. *Annals of Operation Research*, 20:1–66, 1989.

[BDK93] R.E. Burkard, K. Dlaska, and B. Klinz. The quikest flow problem. *ZOR-Methods and Models of Operations Research*, 37:31–58, 1993.

[Bea96] K. A. Beall, editor. *Proceedings of 13th meeting of the UJNR Panel on Fire Research and Safety*, Gaithersburg, MD, 1996.

[Bel58] R. E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[BSS89] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operation Research*, 43:216–224, 1989.

[CC90] Y.L. Chen and Y.H. Chin. The quickest path problem. *Computers and Operations Research*, 17:153–161, 1990.

[CFHT88] W. Choi, R.L. Francis, H.W. Hamacher, and S. Tufekci. Modelling of building evacuation problems with side constraints. *European Journal of Operation Research*, 35:98–110, 1988.

[CFS82] L.G. Chalmet, R.L. Francis, and P.B. Saunders. Network models for building evacuation. *Management Science*, 28:86–105, 1982.

[CH66] K.L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis Applications*, 14:493–498, 1966.

[Cha98] I. Chabini. Discrete dynamic shortest path problems in transportation applications. *Transportation Research Record*, 1645:8170–175, 1998.

[CM85] H.W. Corley and I.D. Moon. Shortest paths in networks with vector weights. *Journal of Optimization Theory and Applications*, 46:79–86, 1985.

[DF96] J.G. Doheny and J.L. Fraser. MOBEDIC - A decision modelling tool for emergency situations. *Expert Systems With Applications*, 10(1):17–27, 1996.

[Dre69] S.E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17:395–412, 1969.

[DS88] M. Desrochers and F. Soumis. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR*, 26:191–212, 1988.

[ENY96] M. Ebihara, H. Notake, and Y. Yashiro. Assessment of clarity of egress route in buildings. In Beall [Bea96], pages 43–51.

[EOI92] M. Ebihara, A. Ohtsuki, and H. Iwaki. Model for simulating human behavior during emergency evacuation based on classificatory reasoning and certainty value handling. *Shimizu Technical Research Bulletin*, 11:27–33, 1992.

[Fah91] R.F. Fahy. An evacuation model for high rise buildings. In *Proceedings of the Third International Symposium on Fire Safety Science*, pages 815–823, Elsevier, London, 1991.

[FF58] L.R. Ford and D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operation Research*, 6:419–433, 1958.

[FF62] L.R. Ford and D.R. Fulkerson. *Flows in Network*. Princeton University Press, Princeton, New Jersey, 1962.

[Fle01] Lisa Fleischer. Universally maximum flows with piecewise-constant capacities. *Networks*, 38(3):115–125, 2001.

[Fru71] John J. Fruin. Pedestrian Planning and Design, Metropolitan Association of Urban Designers and Environmental Planners, 1971. out of print.

[FS02] L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In W.J. Cook and A.S. Schulz, editors, *Integer Programming and Combinatorial Optimization,* volume 2337 of *Lecture Notes in Computer Science*, pages 36–53, Springer, Berlin, 2002.

[FT98] L. Fleischer and E. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operation Research Letters*, 23:71–80, 1998.

[Gal59] David Gale. Transient flows in networks. *The Michigan Mathematical Journal*, 6:59–63, 1959.

[GGO+99] S. Gwynne, E.R. Galea, M. Owen, P.J. Lawrence, and L. Filippidis. A review of the methodologies used in the computer simulation of evacuation from the built environment. *Building and Environment*, 34:741–749, 1999.

[GGT89] G. Gallo, M.D. Grigoriadis, and R.E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. COMPUT*, 18(1):30–55, 1989.

[GKL00] T. Getachew, M. Kostreva, and L. Lancaster. A generalization of dynamic programming for Pareto optimization in dynamic networks. *RAIRO Operation Research*, 34:27–47, 2000.

[GMB99] E. Graat, C. Midden, and P. Bockholts. Complex evacuation; effects of motivation level and slope of stairs on emergency egress time in a sports stadium. *Safety Science*, 31:127–141, 1999.

[Han80] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making : Theory and Applications, Lecture Notes in Economics and Mathematical Systems*, volume 177, pages 109–127. Springer-Verlag, Berlin, 1980.

[HFV00] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamic features of escape panic. *Nature*, 407:487–490, 2000.

[HK00] H.W. Hamacher and Kathrin Klamroth. *Linear and Network Optimization - A Bilingual Textbook*. Friedr. Vieweg & Sohn Verlagsgesellschaft GmbH, Braunschweig/Wiesbaden, Germany, 2000.

[HT94] 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.

[HT02] H.W. Hamacher and S.A. Tjandra. Mathematical modeling of evacuation problems: a state of the art. In Schreckenberg and Sharma [SS02], pages 227–266.

[ICB] International Conference of Building Officials (ICBO). http://www.icbo.org/.

[JR82] J.J. Jarvis and H.D. Ratliff. Some equivalent objectives for dynamic network flow problems. *Management Science*, 28:106–108, 1982.

[KF85] T.M. Kisko and R.L. Francis. EVACNET+ : A computer program to determine optimal evacuation plans. *Fire Safety Journal*, 9:211–220, 1985.

[KKWS00] H. Klüpfel, T.M. König, J. Wahle, and M. Schreckenberg. Microscopic simulation of evacuation processes on passenger ships. In *Fourth International Conference on Cellular Automata for Research and Industry*, Karlsruhe, Germany, 2000.

[Klo95] John H. Klote. Design of smoke control systems for elevator fire evacuation including wind effects. In *Proceedings of the 2-nd Symposium American Society of Mechanical Engineers*, pages 59–77, Baltimore, MD, 1995.

[KNS74] 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(5):814–821, 1974.

[KW93] M.M. Kostreva and M.M. Wiecek. Time dependency in multiple objective dynamic programming. *Journal of Mathematical Analysis and Application*, 173(1):289–307, 1993.

[KW95] B. Klinz and G.J. Woeginger. Minimum cost dynamic flows : the series-parallel case. In *Proceedings of the 4-th Conference on Integer Programming and Combinatorial Optimization*, pages 329–343, Copenhagen, Denmark, 1995.

[Løv95] G.G. Løvås. On performance measures for evacuation systems. *European Journal of Operation Research*, 85:352–367, 1995.

[Løv98] G.G. Løvås. Models of wayfinding in emergency evacuations. *European Journal of Operation Research*, 105:371–389, 1998.

[Mar84] E.Q.V. Martins. On a multicriteria shortest path problem. *European Journal of Operation Research*, 16:236–245, 1984.

[McG97] C.C. McGeoch. Priority queue core test. *The Fifth DIMACS Challenge*, pages 1–8, 1997.

[Min73] E. Minieka. Maximal, Lexicographic, and dynamic network flows. *Operations Research*, 21:517–527, 1973.

[Min74] E. Minieka. Dynamic network flows with arc changes. *Networks*, 4:255–265, 1974.

[MKSB97] S.A. Matheny, D.C. Keith, S.C. Sundstrom, and C.G. Blood. A medical planning tool for projecting the required casualty evacuation assets in a military theater of operations. Technical Report 97-7G, Naval Health Research Center, San Diego, California, 1997.

[Mon94] Christian Montes. Evacuation of buildings. Master's thesis, Department of Mathematics, Universität Kaiserslautern, Kaiserslautern, Germany, 1994.

[Ogi88] R.G. Ogier. Minimum delay routing in continuous-time dynamic networks with piecewise constant capacities. *Networks*, 18:303–318, 1988.

[OGL96] M. Owen, E.R. Galea, and P.J. Lawrence. The exodus evacuation model applied to building evacuation scenarios. *Journal of Fire Protection Engineering*, 8(2):65–86, 1996.

[OR90] 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.

[OR91] A. Orda and R. Rom. Minimum weight paths in time-dependent network. *Networks*, 21(3):295–320, 1991.

[Orl88] J.B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Proc. 20-th Annual Symp. Theory of Computing*, pages 377–387, 1988.

[PB91] Dimitri P. Bertsekas. *Linear Network Optimization : Algorithms and Codes.* The MIT Press, Cambridge, Massachusetts, 1991.

[Phi90] A.B. Philpott. Continuous-time flows in networks. *Mathematics of Operation Research*, 15(4):640–661, 1990.

[Phi94] A.B. Philpott. Continuous-time shortest path problems and linear programming. *SIAM Journal Control and Optimization*, 32(2):538–552, 1994.

[PM93] A.B. Philpott and A.I. Mees. A finite algorithm for shortest path problems with time-varying costs. *Applied Mathematics Letters*, 6(2):91–94, 1993.

[PS97] S. Pallottino and M.G. Scutella. Shortest path algorithms in transportation models: classical and innovative aspects. Technical Report 97-06, University of Pisa, Pisa, Italy, 1997.

[RSX91] J.B. Rosen, S.Z. Sun, and G.L. Xue. Algorithms for the quickest path problem and the enumeration of quickest paths. *Computers and Operations Research*, 18:579–584, 1991.

[San97] Anders Sandberg. Unannounded evacuation of large retail-stores: an evaluation of human behavior and the computer model simulex. Technical report, Department of Fire Safety Engineering, Lund University, Sweden, 1997.

[SBM95] S.C. Sundstrom, C.G. Blood, and S.A. Matheny. The optimal placement of casualty evacuation assets : a linear programming model. Technical Report 95-39, Naval Health Research Center, San Diego, California, 1995.

[SS02] M. Schreckenberg and S.D. Sharma, editors. *Pedestrian and Evacuation Dynamics*, Springer-Verlag, Berlin, 2002.

[THM96] T. Tanaka, I. Hagiwara, and Y. Mimura. A consideration on required number of exits in a room. In Beall [Bea96], pages 53–63.

[Wil71] W.L. Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operation Research*, 19:1602–1612, 1971.

[Wit02] Deborah Withington. Life saving applications of directional sound. In Schreckenberg and Sharma [SS02], pages 277–296.

[WZ00] W.W. Wardell and A.K. Ziliaskopoulos. A intermodal optimum path algorithm for dynamic multimodal networks. *European Journal of Operation Research*, 125:486–502, 2000.

[Yam96] Takeo Yamada. A network approach to a city emergency evacuation planning. *International Journal of Systems Science*, 27(10):931–936, 1996.

[ZM93] A.K. Ziliaskopoulos and H.S. Mahmassani. Time-dependent shortest path algorithm for real-time intelligent vehicle/highway system. *Transportation Research Record*, 1408:94–104, 1993.

Appendix A

Building 42 and Its Network Representations

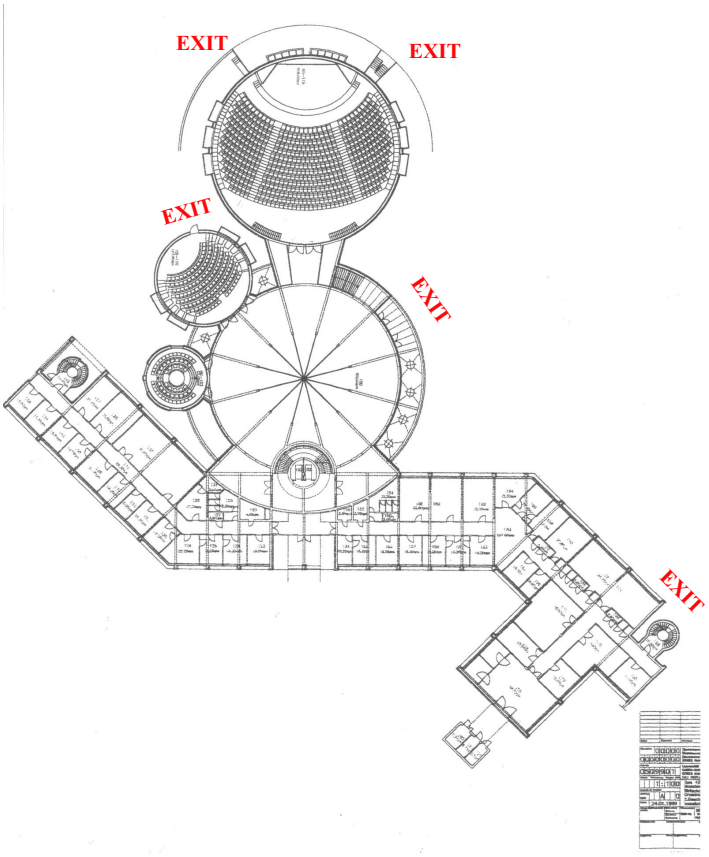


Figure A.1: Building 42 first floor

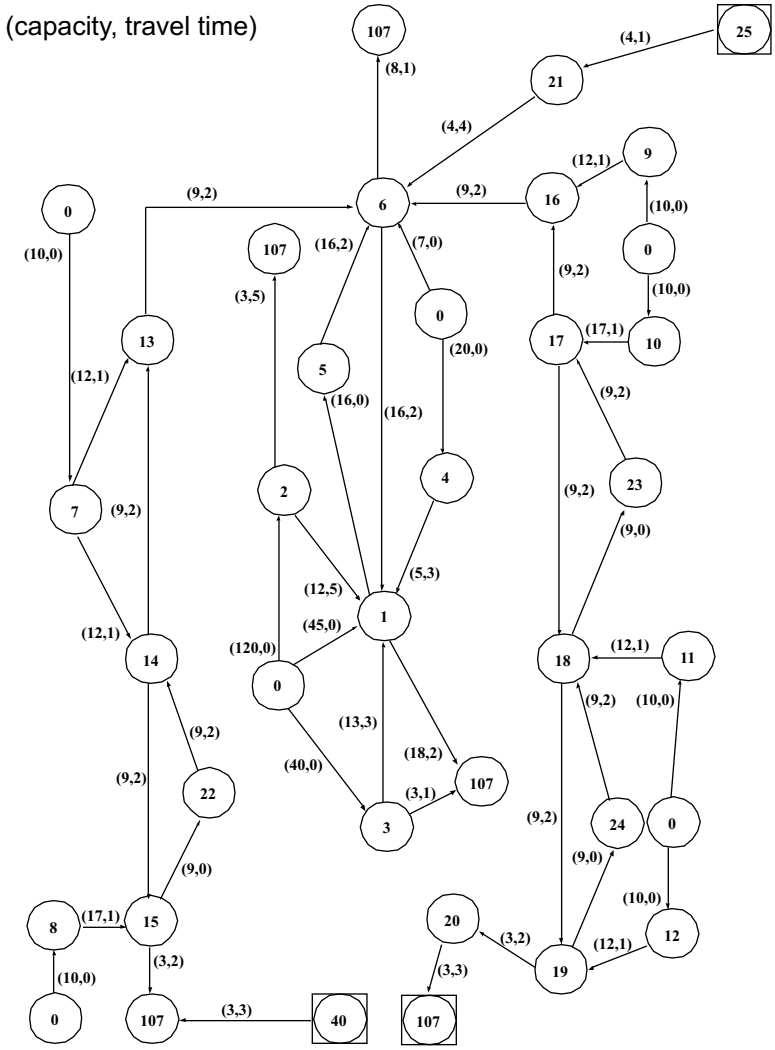


Figure A.2: Network representation of the 1st-floor of Building 42

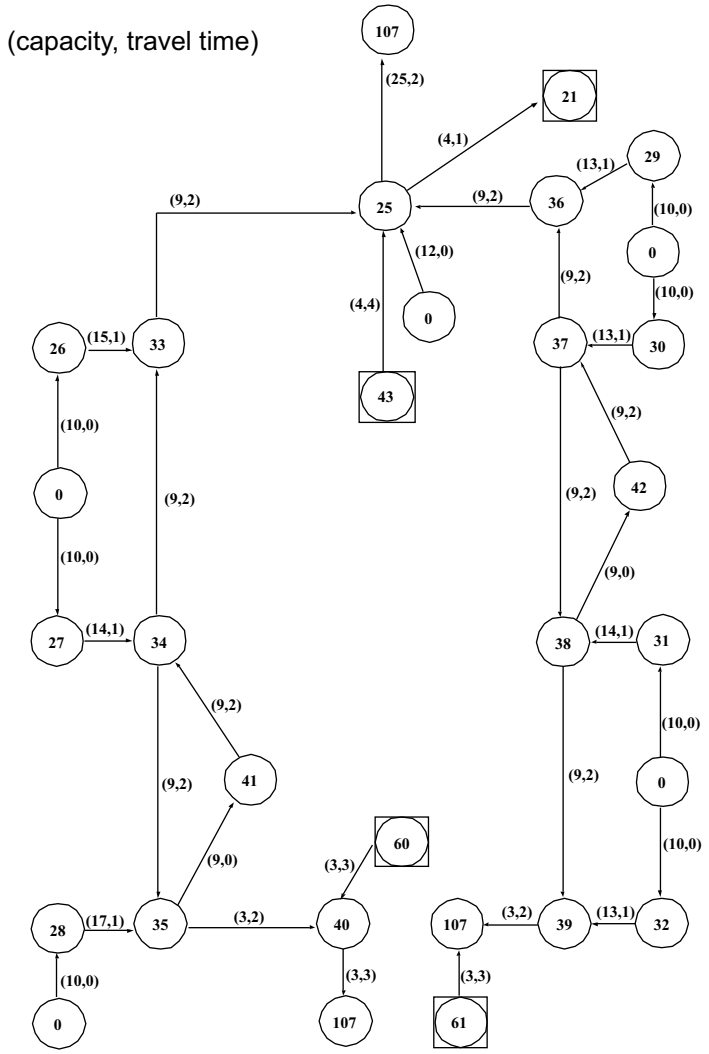


Figure A.3: Network representation of the 2nd- floor of Building 42

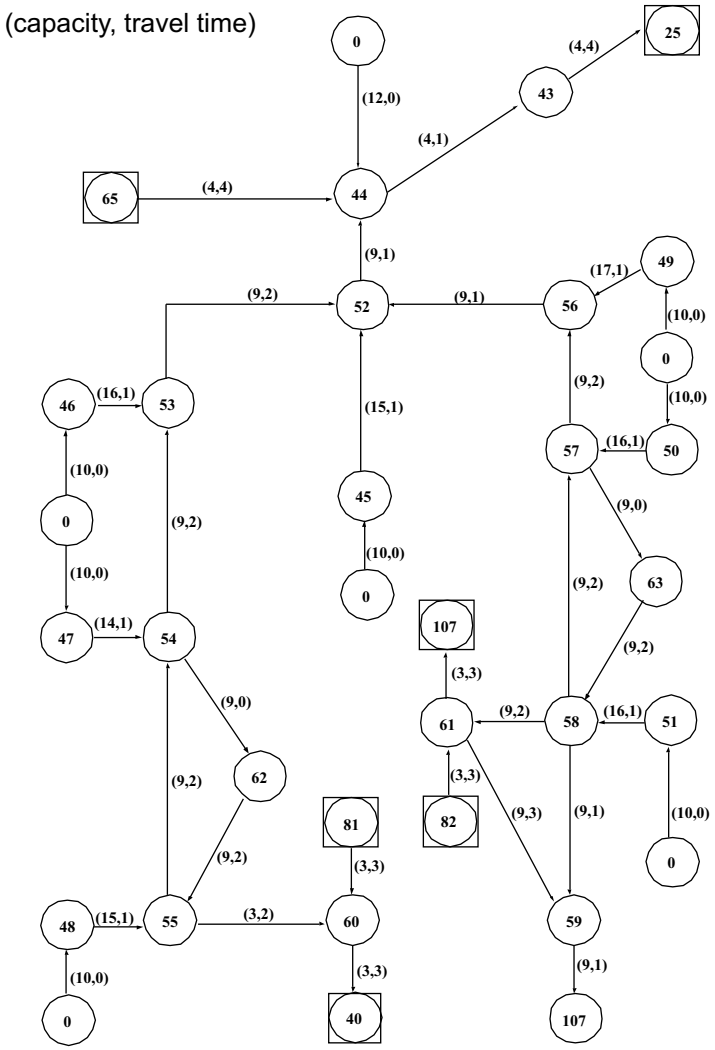


Figure A.4: Network representation of the 3rd- floor of Building 42

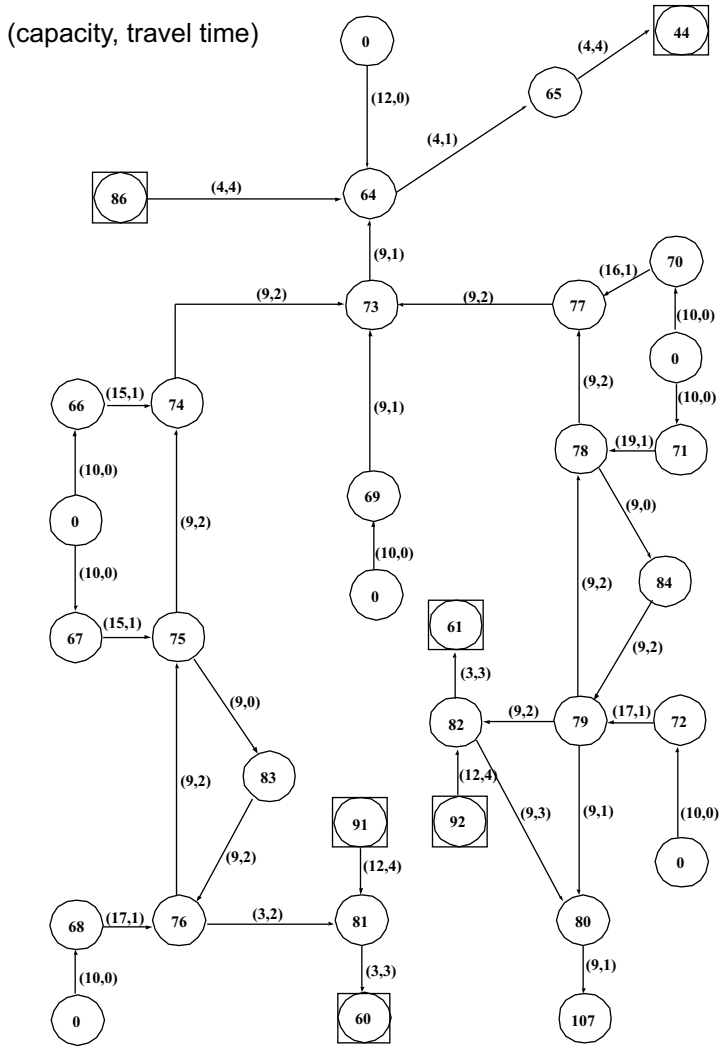


Figure A.5: Network representation of the 4th- floor of Building 42

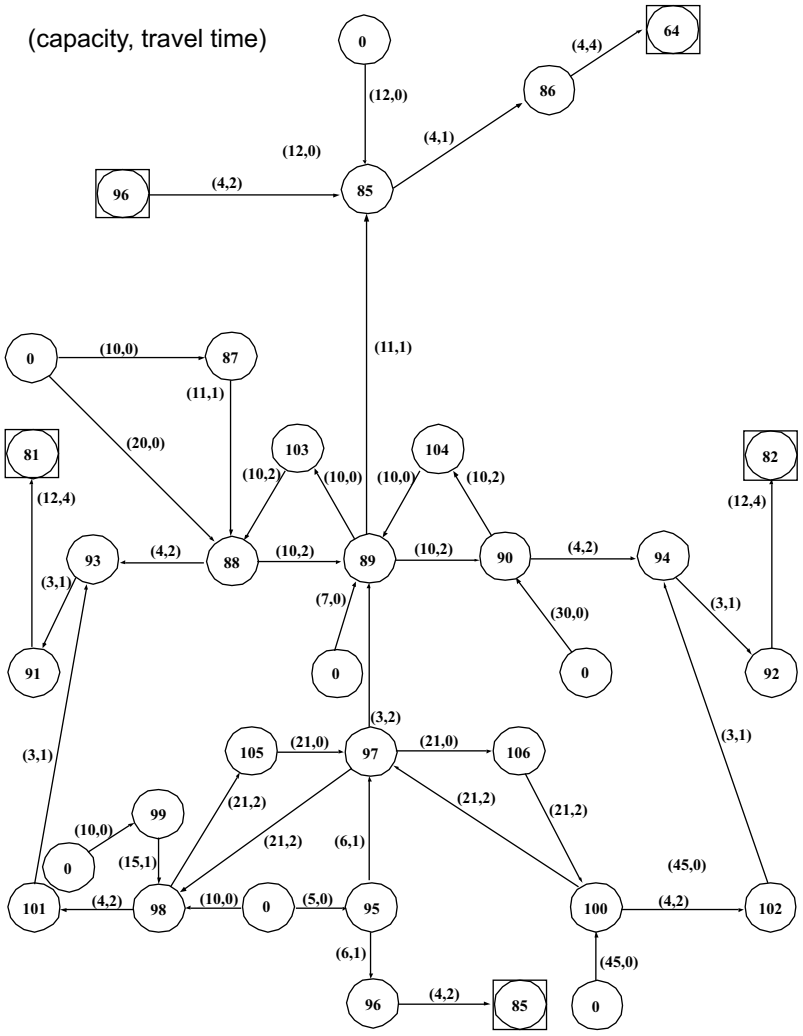


Figure A.6: Network representation of the 5th- and 6th-floor of Building 42

Appendix B

Dynamic Network Generator

Here we describe a generator for dynamic networks with time-dependent attributes. The generator can generate networks associated with two classes of problems maximum dynamic flow (or earliest arrival flow) and bicriterion dynamic shortest path problems. The user has control of the network structure through the use of the following parameters

- random seed,
- time horizon T,
- number of nodes,
- indegree and outdegree of each node,
- minimum and maximum value of allowable travel times
- allowable waiting time at any node (zero or one)

We make the distinction between words "parameters" and "attributes". Parameters are associated with the inputs which must be given by the user to generate the network and attributes are associated with the network attributes such as travel times, capacities, etc.. The generator asks the user to supply a seed number for the random number generator. This feature implies that the generator will regenerate the same problem if every input parameter is the same. Furthermore, the users are allowed to control some parameters to vary the structural characteristics within a class of problem. For the maximum dynamic network flow problems, these parameters are

- minimum and maximum value of arc capacities (must be nonnegative) and
- maximum value of waiting capacities (must be nonnegative).

and for the bicriterion dynamic shortest path problems, these parameters are

- minimum and maximum value of allowable costs on both the first and second criteria and

- maximum value of holding cost.

The creation process of a dynamic network problem can be divided into two main parts. The first part creates a network which is concerned with obtaining the proper number of nodes and guaranteeing that the resulting network is connected. The second part completes the generation while insuring that the remaining specifications are fulfilled, such as indegree and outdegree requirements of each node, range of travel times, etc..

The first part of the network generation is to give all nodes an integer number between 0 and the number of nodes minus one. The source node and sink node is numbered by zero and number of nodes minus one, respectively. Next, some paths from source to sink are generated in such a way that they are *pairwise disjoint and mutually exhaustive* of all nodes except source and sink (following the idea of Klingman, Napier, and Stutz in developing NETGEN [KNS74]). To generate such paths, we create a list L containing all nodes but the source node. From the source node, we select one node i from L randomly using random numbers from *a uniform probability distribution* and connect the source node to i (creating an arc). If i is the sink node, then we obtain a source-sink path and repeat the process from the source node. Otherwise, remove i from L, repeat the process of selecting a node from L, and connect the new selected node to the last selected one to create a new arc until L becomes empty. If L is empty but i is not the sink node, then just connect i to the sink node and terminate the first part of the creation process. Note that a condition for selecting a node from L must be given in such a way that an arc connecting source and sink nodes are not twice generated. During the creation of an arc, the associated arc attributes, such as travel times, capacities (for maximum dynamic network flow problems), and cost vectors (for bicriteria dynamic shortest path problems), are also generated for each time period $t = 0, \ldots, T$. The attribute values are generated randomly while insuring that the specifications given by the user are met, such as travel times range, cost range, and capacity range. The first part of the creation process is ended by generating all node attributes, such as waiting times, waiting capacities (for maximum dynamic network flow problems), and holding cost (for bicriteria dynamic shortest path problems), for each time period $t = 0, \ldots, T$. At this point, the network has the correct number of nodes, nodes attributes, and is guaranteed to be connected.

The second part of the network generation process begins by creating the lists of nodes having less indegree, I, and less outdegree, O, compare to the indegree and outdegree specifications given by the user. Next, a new arc (i, j) is created by selecting a node i and j randomly from O and I, respectively. A condition must be given in such a way that no parallel arc is created and the antisymmetric property of the graph is maintained. This process is continued until at least one of I and O is empty. If I is empty but not O, then one can enforce the indegree requirement by connecting the source node to nodes with deficient indegree if it is possible (avoiding parallel arcs and keeping antisymmetric). Similarly, if it is possible, one can enforce the outdegree requirement by connecting nodes

with deficient outdegree to the sink node. The final network will approximate the required indegree and outdegree of each node. The creation of an arc is followed by generating the associated arc attributes as described in the first part.

The dynamic network generator also provides a tool to generate acyclic network with negative values of attributes. In this case, networks having *a topological order* are generated. To control the distribution of negative travel times and capacities, one additional specification can be given by the user, namely the percentage of negative data $\bar{v}$. During the generation process of attribute values for an arc, a random integer number v in the closed interval $[0, 100]$ is generated. If $v \geq \bar{v}$, then no negative value will be generated. Otherwise, only negative values are generated. This percentage of negative data will not give any effect when the minimum value of an attribute is nonnegative.

Appendix C

Representative Operation Counts

Ahuja, Magnanti, and Orlin [AMO93] noted that CPU time as a measure of the performance of an algorithm has the following drawbacks:

- it is implementation dependent, i.e. the CPU time strongly depends, for example, on the chosen programming language, compiler and computer, the implementation style and skill of the programmer;
- because of multiple sources of variablity, CPU times are often hard to replicate;
- since CPU time is an aggregate measure of empirical performance, it is difficult to obtain detailed insight into an algorithm's behavior using CPU time measurements only. As an example, a typical CPU time analysis does not identify the bottleneck operations of an algorithm.

Therefore, they propose to use the *representative operation counts* to overcome those drawbacks. The background of their idea is that instead of counting the number of times the algorithm executes each line of code, one can focus on *a relatively small number of lines that represent the empirical behavior of the algorithm.* These small number of lines of code are called as the *representative operation counts.*

In order to describe the idea of representative operation counts, we need the knowledge of what a computer does in executing a program.

Definition C.1 (e.g. Ahuja, Magnanti, and Orlin [AMO93]) *Suppose that z denotes the size of a problem. An algorithm is said to be $\Theta(g(z))$ if for some constants $k_1 > 0, k_2 > 0$, and z_0, the algorithm takes at least $k_1 g(z)$ and at most $k_2 g(z)$ time for all $z \geq z_0$.*

Let assume that the computer's program is written so that each line of the code gives $\mathcal{O}(1)$ instructions to the computer and that each instruction requires $\mathcal{O}(1)$ time units, and at least one time unit. By this assumption, the execution time of each line of code is bounded from both above and below by a constant number of units. Therefore, each line of code requires $\Theta(1)$ time units. Suppose that the computer program we are investigating

consists of a finite number of lines of computer code $l_1, \ldots, l_K$. Let M be a subset of $\{1, \ldots, K\}$ and l_M denote the set $\{l_i : i \in M\}$. For a given instance I of the problem, $\alpha_i(I)$, $i = 1, \ldots, K$, denote the number of times that the computer executes line i of this computer program.

Definition C.2 (Ahuja, Magnanti, and Orlin [AMO93]) *The set l_M is a representative set of lines of code of a program if for some constant k,*

$$\alpha_i(I) \leq k(\sum_{j \in M} \alpha_j(I))$$

for every instance I of the problem and for every line l_i of code.

This means that the time spent in executing any line l_i is dominated (up to a constant) by the time spent in executing the lines of code in a representative set. Therefore, the CPU time of the computer program on instance I denoting by CPU(I), is determined by that of the representative set, as described by the following property.

Property C.1 ([AMO93]) *Let M be a representative set of lines of code. Then*

$$CPU(I) = \Theta(\sum_{j \in M} \alpha_j(I))$$

To determine the representative operation counts of an algorithm, the following criteria can be used (McGeoch [McG97]):

- a most frequently executed line of code, such as one appearing inside several nested loops
- a count of loop iterations that are sensitive to some property of input, and
- a frequently executed data structure operation, such as a comparison between two labels

By identifying the representative operation counts, we can use them

- to estimate the CPU time,
- to identify some *asymptotic bottleneck operations*, and
- to compare two algorithms

as described in the following paragraphs.

Ahuja, Magnanti, and Orlin [AMO93] proposed to use the representative operation counts as a linear estimation of CPU time. This linear estimation is called the virtual running

time of an algorithm. If we denote the virtual running time of an algorithm for an instance I by $V(I)$, then

$$V(I) = \sum_{i=1}^{|M|} k_i \alpha_i(I)$$

where $k_i,\ i = 1, \ldots, \mid M \mid$ are constants selected so that $V(I)$ is the best possible estimate of $CPU(I)$. It is suggested to use *multiple regression* to obtain those constants $k_1, \ldots, k_{|M|}$. Using virtual running time is particularly well suited for situations in which the testing is carried out on more than one computer with different capabilities. When the experiment changes from one computer system to another, the representative operation counts remain unchanged. Therefore, using the representative operation counts from the previous study and the constants (of the estimated function) obtained for the new computer system (by doing regression analysis on the new computer), one can obtain the virtual running time for all problems of the previous study measured in terms of the new computer system. This is one of the several advantages of using virtual running time instead of CPU time. More advantages are described in [AMO93].

Definition C.3 (asymptotic bottleneck/nonbottleneck operations) *An operation is considered to be an asymptotic bottleneck operation for an algorithm if the operation consumes a significant percentage of the sum over all operations counts as the problem size increases. In contrast, an operation is considered to be an asymptotic nonbottleneck operation if its share in the the sum over all operations counts becomes smaller and approaches zero as the problem size increases.*

[AMO93] suggested the graphical approaches for indentifying an asymptotic bottleneck operation of an algorithm. This is done by making plots of $\alpha_i(I)/(\sum_{j \in M} \alpha_j(I)),\ \forall i = 1, \ldots, \mid M \mid$ for increasingly larger problem instances I and look for a trend. To estimate the growth rate of this bottleneck operation, [AMO93] suggested to use an estimated function of n and δ, such as $c_M n^{e_{1M}} \delta^{e_{2M}}$ for some choices of constants c_M, e_{1M}, and e_{2M}. Multiple regression technique is then used to estimate the values of these constants and obtain the statistics (such as adjusted R^2 and standard errors) to evaluate the quality of the fit.

The representative operation counts can also be used to compare two different algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$. Let $\alpha_{\mathcal{A}_1}(k)$ and $\alpha_{\mathcal{A}_2}(k)$ be the total expected number of representative operations performed by the algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ on instances of size k. Algorithm $\mathcal{A}_1$ is said asymptotically superior to algorithm $\mathcal{A}_2$ if

$$\lim_{k \to \infty} \frac{\alpha_{\mathcal{A}_1}(k)}{\alpha_{\mathcal{A}_2}(k)} = 0$$

Appendix D

Classification of Dynamic Network Problems

Here, we describe a classification scheme either for dynamic network path problems or dynamic network flow problems. We use this classification scheme throughout this dissertation. This classification scheme is based on three positions

$$\textbf{Pos1} \ / \ \textbf{Pos2} \ / \ \textbf{Pos3} \tag{D.1}$$

with the following meaning.

Pos1 : Number of source nodes and sink nodes, e.g.
- (s, d) : single source s and single sink d
- (S, D) : multi sources - multi sinks with S the set of source nodes and D the set of sink nodes with cardinality greater than one

Pos2 : Assumption on the network parameters which may include, for example, the travel time λ, arc capacity u , node (or holdover) capacity a, maximum waiting time at a node w, flow cost c and holding cost h.
Some parameters may be constant or time-dependent.
If the parameter is time-dependent, then we end the notation with (t), e.g. $(\lambda(t), u(t), a, w(t), c(t), h(t))$ denotes the network with time-dependent travel time, time-dependent arc capacity, constant node capacity, time dependent maximum waiting time, and time-dependent flow and holding costs.
We write the parameter only if it is necessary.

Pos3 : Type of the problem, where notation $\sum$ and $\int$ are used for the discrete-time and continuous-time representations, respectively. The notation T denotes the time horizon T.

$\mathbf{c}_{\sum^T}$	: discrete-time minimum cost problem
$\mathbf{c}_{\sum^T par}$	: discrete-time minimum cost problem with Pareto relation
$\mathbf{c}_{\sum^T lex}$	: discrete time minimum cost problem with lexicographic relation
$T_{\sum}(y(0))$	: discrete-time quickest flow problem
$V_{\sum^T}$	: discrete-time maximum flow problem
$V_{\sum^{T' \leq T}}$	: discrete-time earliest arrival flow problem
$PO(\mathbb{P})$	: Pareto optimum dynamic paths problem
$\mathbf{c}_{\int^T}$	: continuous-time minimum cost problem
$V_{\int}^{T' \leq T}$	: continuous-time earliest arrival flow problem (CTEAF)

The dynamic properties of the system is recognized from the notation T used in the third position of the classification scheme. As an example, single source - single sink discrete-time maximum dynamic network flow problem with constant travel times and constant arc capacities (see Section 3.3) is classified as

$$(s, d)/(\lambda, u, a)/V_{\sum^T}$$

while, the corresponding minimum cost static circulation problem with cost defined by (3.38) can be classified as

$$(s, d)/(u, c)/\mathbf{c}_{\sum circ}$$

List of Tables

3.1 Minimum cost circulation flow for static network in Example 3.2 41
3.2 Time-dependent travel times and capacities for the network in Example 3.3 50
3.3 Time-dependent waiting capacities for the network in Example 3.3 50
3.4 Initial label for the residual network $G_x(2)$ in Example 3.3 51
3.5 The node labels after the first shortest dynamic augmenting path in $G_x(2)$ is found 51
3.6 Maximum dynamic flow of Example 3.3. 52

4.1 Modified step 2 of Algorithm 4.3 when infinite waiting is allowed 70
4.2 Labels on nodes after completing Algorithm 4.3 71
4.3 Time-dependent travel times and capacities for the residual network after the first pass of Algorithm 4.3 72
4.4 Labels on nodes after completing Algorithm 4.3 for the fifth times 72
4.5 Earliest arrival flow of Example 4.3. The upper position of each row $x_{ij}(t)$ contains an optimal flow when the waiting at a node is limited, while the lower one deals with the case when infinite waiting is allowed for every node 73
4.6 The sets Γ_{ij}^T for every arc $(i,j) \in A$ and Γ_{ii}^T for every node $i \in N - \{0,5\}$ correspond to the earliest arrival flow of Example 4.3 74

5.1 The optimal dynamic flow of $(s^*, d)/(\lambda(t), u(t), a(t))/V_{\sum^{T'} \leq T}$ of Example 5.1 86

6.1 Pareto optimal paths and their associated values of Example 6.1 97
6.2 The label set $\Pi_i(t)_{prm}$ for every node $i \in N$ and for every time $t \in \{0, \ldots, T\}$ of Example 6.5 109
6.3 (Cont.) The label set $\Pi_i(t)_{prm}$ for every node $i \in N$ and for every time $t \in \{0, \ldots, T\}$ of Example 6.5 110
6.4 Computational test results 125
6.5 Computational test results of Algorithm 6.4 for three sets of networks with $n = 500$, $m/n = 8$, and three different percentages of negative data 127

7.1 Some occupant load factor (see table 10-A of the Uniform Building Code [ICB]). 132
7.2 Travel speed for different crowd levels, Fruin [Fru71] 134
7.3 Node definition of the floor section plan 135

7.4 Arc definition of the floor section plan 136
7.5 Distribution of initial occupations of Building 42 139

List of Figures

1.1 Walking through disaster 4

2.1 Conservation flows 9
2.2 The network $G = (N, A, T)$ and its associated time-expanded network G_T . 13
2.3 Compact version of G_T for $T = 4$ in Figure 2.2 under an assumption that $\lambda_{12}(t) := 1,\ t \leq 4$ and $u_{12}(t) := 2,\ t \leq 4$ 16
2.4 Time-expanded network G_T with $T = 4$ for the discrete-time maximum dynamic network flow problem 16
2.5 Time-expanded network G_T with $T = 4$ for the discrete-time minimum cost dynamic network flow problem 17
2.6 Non-FIFO property 18
2.7 A single arrival time associates with several departure times. The dashed lines correspond to the backward arcs 20

3.1 Eight possible cases of i and j that may be in C_T or $\overline{C_T}$ for every $(i, j) \in A - \{(s, d)\}$ 28
3.2 Network G for Example 3.1 32
3.3 Discrete-time dynamic network flow for Example 3.1. The solid lines and dashed lines correspond to TRF and non-TRF, respectively 32
3.4 Travel time distance $\lambda_i(P_l)$ from the source node s to the node i along the path P_l 34
3.5 A circulation network with circulation arc $(6, 1)$ and cost of each arc defined by (3.38) 35
3.6 Dynamic augmenting path 42
3.7 A network for Example 3.3 50
3.8 Identifying asymptotic bottleneck operations in Algorithm 3.2 53
3.9 CPU time and the number of shortest dynamic augmenting paths for various n and densities $\delta = m/n$ 54
3.10 Determine the quality of the estimators of performance functions of Algorithm 3.2 54

4.1 Discrete-time maximum dynamic flow vs discrete-time earliest arrival flow. The solid lines indicate a maximum dynamic flow without having the earliest arrival property, and the dashed lines indicate an earliest arrival flow . . . 59

4.2 A chain decomposition and its associated induced dynamic flow 62
4.3 A network for Example 4.2 62
4.4 The minimum dynamic cut of Example 4.3 The shaded circles are in the source side of the cut. The bold arrows represent the optimal flows of the arcs in the cut. The numbers beside the arrows define the values of the flows 75
4.5 Identifying asymptotic bottleneck operations in Algorithm 4.4 76
4.6 CPU time and the number of earliest arrival augmenting paths for various n and densities $\delta = m/n$ 77
4.7 Determining the quality of the estimators of performance functions of Algorithm 4.4 77

5.1 A Network for Example 5.1 86

6.1 Memory-less property of maximum waiting time 92
6.2 Dynamic path on the time-expanded graph 93
6.3 Example network with exponential number of Pareto optimal paths 96
6.4 A network for Example 6.2 99
6.5 A network for Example 6.3 100
6.6 Label structure 101
6.7 Network for Example 6.5 108
6.8 The dynamic cycle for the proof of Lemma 6.1 111
6.9 Identifying the asymptotic bottleneck operations and CPU time of Algorithm 6.1 for various n and densities m/n 120
6.10 Determine the quality of the estimators of performance functions of Algorithm 6.1 121
6.11 Identifying asymptotic bottleneck operations in Algorithm 6.2 121
6.12 CPU time (in seconds) of Algorithm 6.2 for various n and densities m/n . 122
6.13 Determine the quality of the estimators of performance functions of Algorithm 6.2 123
6.14 Identifying asymptotic bottleneck operations in Algorithm 6.4 123
6.15 CPU time (in seconds) of Algorithm 6.4 for various n and densities m/n . 124
6.16 Determine the quality of the estimators of performance functions of Algorithm 6.4 125
6.17 Comparing Algorithms 6.1, 6.2, and 6.4 126
6.18 The CPU time of Algorithm 6.4 for ten random samples with $n = 500$, $m/n = 8$, and three different percentages of negative data 127

7.1 The floor plan and its corresponding network representation for Example 7.1 135
7.2 Comparing the results of Montes [Mon94] and Algorithm 5.3 141
7.3 The distribution of evacuees when exit arc $(25, 107)$ in the second floor is completely blocked for time $t > 2$ 142

A.1 Building 42 first floor 153
A.2 Network representation of the 1st-floor of Building 42 154

A.3 Network representation of the 2nd- floor of Building 42 155
A.4 Network representation of the 3rd- floor of Building 42 156
A.5 Network representation of the 4th- floor of Building 42 157
A.6 Network representation of the 5th- and 6th-floor of Building 42 158

Index

admissible arc, 45
admissible dynamic path, 45
antisymmetric, 7
approximation algorithm, 61
asymptotic bottleneck operations, 52, 118
attributes, 7
    constant, 79
    constant attributes, 30
    maximum waiting time, 91
      memory-less property, 92
    time-dependent attributes, 63, 83, 138
    time-dependent capacity, 7
    time-dependent cost, 7
    time-dependent holding cost, 92
    time-dependent node capacity, 8
    time-dependent travel time, 7
      elastic arc model, 8
      frozen arc model, 8, 91
augmenting path
    earliest arrival augmenting path, 63

bicriteria dynamic shortest path, 89
binary search, 81
breadth-first search, 46

chain decomposition, 61
    chain flow, 61
complete set, 95
    minimal complete set, 95, 115
Cost Consistency, 97
cost vector, 91
CPU time, 54, 119
crowd, 130, 133
    persons per meter width-second, 133

Djikstra's label setting algorithm, 15, 103
DTDNFP, *see* Dynamic network, discrete-time dynamic network flow problem
DTEAFP, *see* Dynamic network, earliest arrival flow
DTMDNFP, *see* Dynamic network, maximum dynamic flow
DTQFP, *see* Dynamic network, quickest flow
dynamic augmenting path, 42, 63
dynamic cut, 23, 68
    $s-d$ dynamic cut, 24
    cut value, 24
    dual of the maximum dynamic flow problem, 26
    Maximum dynamic flow - minimum dynamic cut, 29
dynamic cycle, 92
Dynamic network, 7
    continuous-time dynamic network flow problem, 7
    discrete-time dynamic network flow problem, 7, 9
    Earliest arrival flow, 58, 83, 137
    maximum dynamic flow, 31, 80
    quickest flow, 79, 138
    quickest path, 79
dynamic network flow conservation constraints, 9
dynamic path, 92
dynamic programming, 102
dynamic random network generator, 52, 119

EAAP, *see* earliest arrival augmenting paths
earliest arrival time, 63

efficient, *see* Pareto optimal
egress time, *see* evacuation problems, evacuation time
emergency exit, 136
evacuation problems, 130
  evacuation routes, 137
  evacuation time, 130, 139
    awareness time, 130
    egress time, 131
    lower bound, 137, 138
    preparation time, 131
  macroscopic models, 130
  microscopic models, 130

fair comparison, 118
First-In First-Out, 17
  FIFO network, 17
  non-overtaking, 17

holdover flow, 8

inhabited areas, 129
integrality property, 9

Kaiserslautern, 131, 138

label, 64, 100
  predecessor node, 64
  successor, 101
label correcting algorithm, 63
  current node, 64
  pass, 67, 113
  scan eligible list, 64, 109
label setting algorithm, 101
linked list, 101

MCCP, *see* temporally repeated flows, minimum cost circulation problem
mergesort, 103
minimum cost dynamic flow, 144
  minimum cost maximum dynamic flow, 144
  minimum cost quickest flow, 144
movement flow, 8

network modification, 88
  dummy arc, 88
  dummy node, 88
nondominated, *see* Pareto optimal

Pareto optimal, 94, 102, 138
pivot label, 104
PMS, *see* crowd, persons per meter width-second

regression, 55, 120
  adjusted R^2, 55, 120, 122, 124
  standard error, 55, 120, 122, 124
representative operation counts, 52, 118
residual dynamic network, 19
  backward arc, 20
  residual arc capacities, 19
  residual arc costs, 19
  residual travel times, 19
  residual waiting capacities, 20

scaling
  capacity scaling, 42
SE, *see* label correcting algorithm, scan eligible list
set of permanent labels, 104
set of temporary labels, 104
shortest dynamic augmenting path, 43
strong duality theorem, 29
super sink, 15
super source, 15

TdBiDSP, *see* bicriteria dynamic shortest path
temporally repeated flows, 31
  cut definition, 37
  minimum cost circulation problem, 35
    static network, 35
  path decomposition, 31
  path flow, 31, 61
time horizon, 7
time-dependent supplies, 87
time-expanded network, 12, 59, 139
  compact time-expanded network, 14

topological order, 125
totally unimodular, 10
TRF, *see* temporally repeated flows

unimodularity, 10

validity conditions, 44
virtual running time, 55

walkways, 131
weak duality theorem, 29

The Author's Scientific Career

- May 2003
  Ph.D Disputation

- Since September 2001
  Scientific employee in Mathematics Department at the University of Kaiserslautern, Germany (funded by the *Deutsche Forschungsgemeinschaft* (DFG), Germany for the project "Algorithmen für dynamische Netzwerkflüsse mit Anwendungen in der Evakuierungsplanung" in *schwerpunkt* "Algorithmik grosser und komplexer Netzwerke")

- October 1998 - August 2001
  Ph.D study in Mathematics at the University of Kaiserslautern, Germany (funded by the Fraunhofer Institut Techno- und Wirtschaftsmathematik (ITWM), Kaiserslautern, Germany)

- April 1998 - September 1998
  Researcher at the Fraunhofer ITWM, Kaiserslautern, Germany

- August 1990 - April 1998
  Lecturer in Industrial Engineering Department at Petra Christian University, Surabaya, Indonesia

- May 1994 - December 1995
  Master study in Industrial Engineering and Management at the Asian Institute of Technology, Bangkok, Thailand (funded by Petra Christian University, Surabaya, Indonesia)
  Thesis title: *Stochastic Modelling of Production Systems with Multiple Parts*
  Thesis supervisor: Prof. Dr. Nagendra N. Nagarur

- August 1986 - March 1990
  Bachelor study in Mathematics at Sepuluh Nopember Institute of Technology (ITS), Surabaya, Indonesia.
  Thesis title: *Analysis of Elementary Catastrophe Theory*
  Thesis supervisor: Drs. Achmad Subjanto