摘要

在我们的日常生活中，网络无处不在，它们以各种各样应用为背景的形式出现。通常，当网络中所有的参数，如点的容量和弧的容量都是常数，也不考虑流在运输过程中所需的时间，这样的网络就被看作静态的。但是这些假设在现实生活中未必成立。例如，一支舰队，从一个港口出发到达另一个港口，途中所需时间是需要考虑的，还有航道的容量和泊位的数量也都可能随着时间的变化。若一个网络的参数会随着时间而变化，则称这样的网络为时变网络。与静态网络相反，目前只有少数几种解决时变网络最优化问题的方法。

在这篇文章中，我们考虑时变网络的最大流问题。$G(N,A,b,l)$ 表示一个时变网络, N 是点集，A 是弧集，$b(i,j,t)$ 是流经弧 $(i,j)((i,j)\in A)$ 所需的传输时间，$l(i,j,t)$ 是弧 (i,j) 的容量。$b(i,j,t)$ 和 $l(i,j,t)$ 都是弧 (i,j) 第一个点上流的出发时间 t 的函数，$t=0,1,2,...,T$，T $(T>0)$ 是一个给定的数。时变网络最优化问题就是在时间限制 T 内，从发点送尽可能多的流到达收点，这个问题已被证明是 NP- 完备问题。

本文的文章结构如下：首先，基于伪流我们提出一个解决时变最大流问题的算法，剩余容量收缩算法。然后我们给出时间标号算法使得所求最大流的每个子流到达终点的时间都是最早（或最晚）的。我们用同样的算法解决了最短间隔路问题，即所求解中每个子流出发时间和到达时间之间的间隔是最短的。在这篇文章中，我们考虑两种等待条件，它们分别是零等待时间和任意等待时间，我们证明算法正确性的同时也列举算例阐明算法如何进行运算，每一种算法都能在伪多项式时间内找到最优解。

关键词: 最大流，时变网络，剩余容量收缩算法，时间标号算法，最优化，伪流

Abstract

Networks exist everywhere in our daily lives. There are a lot of applications which can be modeled as a network optimization problem. Usually, a network is treated as a static model where all attributes, such as the node capacity and the arc capacity, are constants, and the flow transshipped in it takes zero time. However, these assumptions may not hold in our real world. A fleet, for example, should take time from one harbor to another, and all situations, such the capacity of the fairway and the number of berths, may vary over time. A network with time-varying attributes is called the time-varying network. In contrast with the static network, there are a few approaches for solving the time-varying network optimization problem.

In this thesis, we consider the maximum flow problem in a time-varying network. A time-varying network is denoted by $G(N, A, b, l)$, where N is the node set, A is the arc set, $b(i, j, t)$ is the transit time needed to traverse arc $(i, j) \in A$ and $l(i, j, t)$ is the capacity of arc (i, j). Both $b(i, j, t)$ and $l(i, j, t)$ are functions of the departure time t at the beginning node of arc (i, j), where $t = 0, 1, 2, ..., T$ and $T > 0$ is a given number. The problem is to send the flow as much as possible from the source node to the sink node no later than the time limit T. The problem is known to be NP-complete.

First, we propose an algorithm to solve the time-varying maximum flow problem while adopting preflow-push strategy. Then, we present an algorithm to find such a maximum flow solution that has the earliest (or the latest) arrival time at the sink node. Throughout this thesis, two kinds of waiting constraints are considered, they are zero waiting and arbitrary waiting respectively. Each algorithm can find the optimal solution in pseudopolynomial time.

Keywords: maximum flow, time-varying network, excess scaling algorithm, time labeling algorithm, optimization, preflow

Main Notations

G	network
N	the set of nodes
A	the set of arcs
$b(i,j,t)$	transit time of arc (i,j) at time t
$l(i,j,t)$	capacity of arc (i,j) at time t
$x(i,j,t)$	the flow transshiped on arc (i,j) starting at time t
λ	specify when and how to send flows from the source s to the sink ρ
$f(\lambda,T)$	the total value of flows under the solution λ with the deadline T
$P(s,x)$	a direct path from s to x
$\tau(x_i)$	the departure time of the flow at node x_i
$\alpha(x_i)$	the arrival time of the flow at node x_i
$\omega(x_i)$	the waiting time of the flow at node x_i
$Cap(P)$	the capacity of path P
$B(i)$	the set of arcs emanating to node i
$d(i,t)$	the distance label of node i at time t
$e(i,t)$	the excess of node i at time t
$M(i,t)$	the time label of node i at time t

Acknowledgements

The accomplishment of this thesis benefits from the enlightenment of my supervisor, associate professor Sha Dan, whose inspiring insights, generous encouragements, and enthusiastic instructions have facilitated me much throughout my writing process. His penetrating and insightful comments afford me with inspiring source. He has been in constant concern about my paper, spared no pains to entertain my thesis draft. So my deepest gratitude goes first and foremost to him.

Next, I would like to express my heartfelt gratitude to all teachers in Mathematics and Sciences College, Shanghai Normal University. Without their consistent and illuminating instructions, I can not finish my education for Master degree successively. I am also greatly indebted to the professors and teachers at the Department of Applied Mathematics: Professor Zhang Jizhou, Professor Zhu Detong , Professor Shi Yongbing, Professor Yue Rongxian et al. Thanks should go to their instructive guidance and comprehensive education.

My thanks should also go to my beloved family for their loving and confidence through these years. I also owe my sincere gratitude to my friends and my classmates who gave me their helps for overcoming difficulties in each step of my thesis writing process.

Last but not least, I thank all professors and experts who carefully read this thesis and kindly present their valuable comments. All of these will lead me go ahead with much more self-confidence.

Chapter 1

INTRODUCTION

The maximum flow problem is, in a given network, to send the flow from the source node to the sink node as much as possible without exceeding the capacity constraint. Usually, the problem is described to be static, where the flow takes zero time to travel arcs and both the arc capacity and the node capacity are constants. The problem is in a sense a complementary model to the shortest path problem. The shortest path problem models situation in which flow incurs a cost but is not restricted by any capacities, in contrast, in the maximum flow problem a flow incurs no costs but is restricted by capacity limitations. The maximum flow problem, together with the shortest path problem, arises as a subproblem in algorithms for solving the minimum cost flow problem. The maximum flow problem and the shortest path problem combine all the basic ingredients of network flows. As such, they have become the nuclei of network optimization.

Consider the network $G = (N, A)$ with a nonnegative capacity u_{ij} for each arc $(i, j) \in A$, where A is the set of arcs, N is the set of nodes. There are two special nodes in the network G, *a source node* s and *a sink node* ρ. We wish to find the maximum flow from s to ρ that satisfies the arc capacities and mass balance constraints at all nodes. The maximum flow problem can be formulated as follows.

Maximize $\quad v$

s.t.

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v, & \text{for } i = s \\ 0, & \text{for all } i \in N\backslash\{s,\rho\} \\ -v, & \text{for } i = \rho \end{cases} \tag{1.1}$$

$$0 \leq x_{ij} \leq u_{ij}, \text{ for each } (i,j) \in A$$

We refer to a vector $x = \{x_{ij}\}$ satisfying all the constraints as a *flow* and the corresponding value of the scalar variable v as the *value* of the flow.

We have noticed that in our real lives, most maximum flow models are time-varying. Consider a transportation network, for example, in which several cargo-transportation services are available among a number of cities. Each of them may take a certain time to travel from one city to another, with a limited cargo-transporting capacity. Moreover, the travel time and the capacity for each transportation service are season dependent. A question often asked is: what is the maximum possible cargoes that can be sent between two specific cities within a certain time duration T? This is a time-varying maximum flow problem. Its solution is important, particularly to the planning of the network.

An extension of the classical static problem is the *maximal dynamic flow* problem which is formulated and solved by Ford et al (1962), where the transit time to traverse an arc is taken into consideration. Although their model still assumes that attributes in the problem, including arc capacities and transit times, are time independent, it is widely regarded as the fundamental work on the time-varying maximum flow problem. A further extension is studied by Halpern(1979), where arc capacities vary over time and storages at intermediate nodes may be prohibited at some times. Orlin (1983) considers the problem with an infinite time horizon and the flow is to be sent through the network in each period of time so as to satisfy the upper and lower bounds. He formulates the problem as an infinite integer program.

Cai, Sha & Wong (2007) consider a new extension of the maximum flow problem in which all transit times, arc capacities and node capacities vary over time. The problem is to send as much flow as possible from the source to the sink within a given time limit T. Also, the solution has the property which is called universality. This kind of problem is called *time-varying universal maximum flow* problem which was introduced by Gale (1959). Note that the solution derived by Ford and Fulkerson's algorithm does not necessarily give the universal maximum flow. Minieka (1973) and Wilkinson (1971) have independently modified Ford and Fulkerson's algorithm to produce a universal maximal flow for the network studied by Ford and Fulerson. In the book of *Time-Varying Network Optimization*, Cai, Sha & Wong has been proved the problem to be NP-complete[4]. Cai at el propose a labeling algorithm which can solve the problem in pseudopolynomial time for the network studied by them. Also, they consider the situation that waiting at the intermediate nodes is allowed.

At the same time, Cai, Sha & Wong introduce the concept of the earliest arrival maximal flow and the latest arrival maximal flow no later than a given time limit T. The problem also raises from our daily lives. For example, in the market, if some food is going to be sold out, the producer should provide it to the market as early as possible. Sometimes, we may need to package whole goods again at the interchange station. In order to save the storage cost, we hope that the arrival time of the flow is as late as possible before a deadline. So it is meaningful for us to study the problem.

Moreover, one may want to find another kind of maximum flow. Let us introduce the concept *the shortest duration dynamic path* first. Suppose that $P = (x_1, ..., x_r)$ is a dynamic path from x_1 to x_r. If the time duration between the departure time of a flow at node x_1 and its arrival time at node x_r is minimized, we call such a path as the shortest duration dynamic path. The maximum flow with the property that each subflow is transshipped along the shortest duration dynamic path is called *the shortest duration flow*. Also, such a flow is more important in our daily lives. In a cargo-transportation network, when someone want to send the putrescible commodity from one place to another, he could not spend much time on the way or the commodity will go to bad easily. That is to say, we need to short the time duration between the departure time and the arrival time of a flow.

1.1 Algorithms for the Maximum Flow Problem

To investigate those algorithms for the static maximum flow problem will benefit us for solving the time-varying maximum flow problem.

There are many polynomial algorithms for the static maximum flow problem, such as shortest augmenting path algorithm, capacity scaling algorithm, generic preflow push algorithm, FIFO preflow-push algorithm and so on. Although most of the algorithms presented in literatures to solve the static maximum flow problem look rather different from one another, quite often they share a common structure. A large class of algorithms is characterized by the fact that the value of the current flow is increased at each iteration by adding additional flow along augmenting paths, until no augmenting path is found, so obtaining a maximum flow. It converts the feasible flows into the optimal. Another important class of algorithms is preflow-push algorithm. It flood the network so that some nodes have excesses. These algorithms incrementally relieve flow from nodes with excesses by sending flow from the node forward toward the sink node or backward toward the source node and gradually converting these preflows into a feasible flow. It also makes sense for the time-varying network.

Ford and Fulkerson (1962) have developed an efficient procedure to find the optimal solution for their model that the transit time to traverse an arc is taken into consideration, which first finds the static flow from the source to the sink, and then develops a set of temporally repeated flows, called chain flows, to form the optimal flow. It similar to finding augmenting paths within T.

In this thesis, we will propose two algorithms to solve the time-varying maximum flow problem, one is an excess scaling algorithm and the other is a labeling algorithm. Both of them are based upon the concept preflow.

1.2 Applications

The central theorem in the study of network flows not only provides us with an instrument for analyzing algorithms, but also permits us to model a variety of applications. Examples of the maximum flow problem include determining the maximum flow of (1) petroleum products in a pipeline network, (2) cars in a road network, (3) messages in a telecommunication network, and (4) electricity in an electrical network etc. The problem also arises directly in problems as far reaching as machine scheduling, the assignment of computer modules to compute processors, the rounding of census data to retain the confidentiality of individual households and tanker scheduling. Sometimes the maximum flow problem occurs as a subproblem in the solution of more difficult network problems, such as the minimum cost flow problem or the generalized flow problem. The maximum flow problem also arises in a number of combinatorial applications that on the first sight it does not appear to be a maximum flow problem. As we have known that, the maximum-flow min-cut theorem establishes an important correspondence between flows and cuts in networks. Indeed, as we will see, by solving a maximum flow problem, we also solve a complementary *minimum cut problem.* The fact that maximum flow problems and minimum cut problems are equivalent has practical implications as well. It means that the theory and algorithms that we develop for the maximum flow problem are also applicable to many practical problems that are naturally cast as minimum cut problems.

In the rest of this thesis, we will, in Chapter 2 and Chapter 3, study the time-varying maximum flow problem with zero waiting time constraint and arbitrary waiting time constraint respectively. In each chapter, we will propose algorithms and prove their correctness. To illustrate how the algorithm works, some numerical examples are given either.

Chapter 2

TIME-VARYING MAXIMUM FLOW PROBLEMS WITH ZERO WAITING TIME

In this chapter, we consider the time-varying maximum flow problem under the situation that no flow can wait at a node at any time except the source node. It means that the flow departs from the node as soon as it arrives at the node.

2.1 Definitions and Formulations for the Time-Varying Maximum·Flow Problem with Zero Waiting Time

Let $G(N, A, b, l)$ be a network without parallel arcs and loops, where N is the set of nodes, A is the set of arcs, $b(i, j, t)$ is the transit time of arc (i, j), $l(i, j, t)$ is the capacity of arc (i, j). Both $b(i, j, t)$ and $l(i, j, t)$ are functions of the departure time t, where $t = 0, 1, 2, ..., T$, and $T > 0$ is a given integer. We assume that the transit time b is a positive integer and the capacity l is a nonnegative integer. We further assume that there are two particular nodes s and ρ, being the source node and the sink node respectively and waiting at any node is prohibited except the source node s. The problem is to send the flow as much as possible from the source node to the sink node no later than the time limit T. Let $f(\lambda, T)$ be the total flow value under the solution λ, which specifies when and how to send flows from the source s to the sink ρ within the time limit T. The time-varying maximum flow problem is to find a solution λ such that $f(\lambda, T)$ is maximized.

Let $x(i,j,t)$ be the flow transshipped on arc (i,j) starting at time t. Therefore we have $\lambda = \{x(i,j,t), (i,j) \in A, t = 0, 1, ..., T\}$. The problem can be presented formally as below:

Maximize $\quad v$

s.t.

$$\left\{\begin{array}{l} \sum_{t=0}^{T-1}\{\sum_{\{j:(i,j)\in A\}} x(i,j,t) - \sum_{\{j:(j,i)\in A, u=t-b(j,i,u)\}} x(j,i,u)\} = v, \\ \qquad\qquad\qquad\qquad \text{for } i = s \\ \sum_{\{j:(i,j)\in A\}} x(i,j,t) - \sum_{\{j:(j,i)\in A, u=t-b(j,i,u)\}} x(j,i,u) = 0, \\ \qquad\qquad \text{for all } i \in N\backslash\{s,\rho\}, t = 1, 2, ..., T-1 \\ \sum_{t=1}^{T}\{\sum_{\{j:(i,j)\in A\}} x(i,j,t) - \sum_{\{j:(j,i)\in A, u=t-b(j,i,u)\}} x(j,i,u)\} = -v, \\ \qquad\qquad\qquad\qquad \text{for } i = \rho \\ 0 \le x(i,j,t) \le l(i,j,t), \text{ for all } (i,j) \in A,\ 0 \le t \le T. \end{array}\right. \tag{2.1}$$

A solution $\lambda = \{x(i,j,t), (i,j) \in A, t = 0, 1, ...T\}$ is called *feasible* if it satisfies all constraints. A solution $\lambda = \{x(i,j,t)\}$ is called a *preflow* if it satisfies arc capacity constraints only.

Look the following example. Given $T = 9$, the capacity and transit time of all arcs are presented in Table 2.1.1, while the capacity of the arc or the transit time of the arc is negative we use " $-$ " stands for.

Obviously, $\lambda = \{x(s,a,1) = 5, x(a,d,3) = 5, x(d,h,6) = 5, x(h,\rho,8) = 5$, other $x(i,j,t)$ equal zeros $\}$ is a feasible flow. $\lambda = \{x(s,a,1) = 6, x(s,d,1) = 3, x(a,d,3) = 5, x(d,h,6) = 5, x(h,\rho,8) = 5$, other $x(i,j,t)$ equal zeros $\}$ is a *preflow*. It is easy to see that zero flow is always a feasible flow.

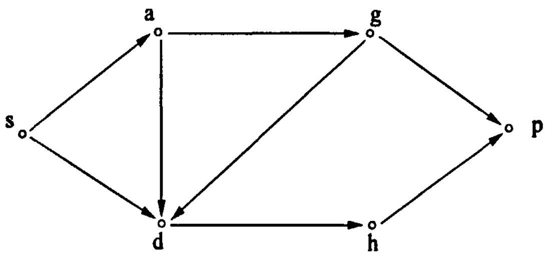

Figure 1. Example 2.1.1

Table 2.1.1 $l(i,j,t)$ and $b(i,j,t)$ of Figure 1

t	(s,a)	(s,d)	(a,g)	(a,d)	(d,h)	(g,d)	(g,p)	(h,p)
0	-,1	1, -	1, -	3, -	-, -	-,-	-, -	0, -
1	6,2	3, 1	2, -	4, 1	0, -	-,-	0, 1	1, -
2	13,3	5, 2	3, 1	5, 2	1, -	0,1	2, 2	2, -
3	20,4	7, 3	4, 2	6, 3	2, -	1,2	4, 3	3, -
4	27,5	9, 4	5, 3	7, 4	3, -	2,3	6, 4	4, -
5	34,6	11, 5	6, 4	8, 5	4, 1	3,4	8, 5	5, -
6	41,7	13, 6	7, 5	9, 6	5, 2	4,5	10, 6	6, -
7	48,8	15,7	8, 6	10, 7	6, 3	5,6	12, 7	7, -
8	55,9	17,8	9, 7	11, 8	7, 4	6,7	14, 8	8, 1
9	62,10	19,9	10,8	12, 9	8, 5	7,8	16, 9	9, 2

Definition 1 *Let $P(s,x) = (s = x_1, ..., x_r = x)$ be a directed path from s to x and assume that a flow traverses the path. Let $\tau(x_1) = 0$ and define recursively*

$$\tau(x_i) = \tau(x_{i-1}) + b(x_{i-1}, x_i, \tau(x_{i-1})), \qquad \textit{for } i = 2, ..., r.$$

The departure time of the flow at a node x_i on P is defined as $\tau(x_i)$ if $1 \leq i < r$. Accordingly, the arrival time of the flow at a node x_i on P is defined as

$$\alpha(x_1) = 0; \quad \alpha(x_i) = \tau(x_{i-1}) + b(x_{i-1}, x_i, \tau(x_{i-1})), \qquad \textit{for } i = 2, ..., r.$$

A directed path P together with all transit times, departure times, and arrival times is called a *dynamic path.* In this chapter, the departure time of a node x_i is equal to the arrival time of the node.

Definition 2 *Let $P = (x_1, ..., x_r)$ be a dynamic path from x_1 to x_r. The time of P is defined as $\alpha(x_r)$. A path has time at most t, if its time is less than or equal to t. Specifically, a path is said to have time exactly t, if its time is equal to t.*

Definition 3 *Let $P(s = x_1, x_2, ..., x_r = y)$ be a dynamic path from s to y. Then, the capacity of path $P(s, x_i)$ is recursively defined as:*

$$\begin{cases} Cap(P(s,x_i)) = \min\{Cap(P(s,x_{i-1})), l(x_{i-1},x_i,\tau(x_{i-1}))\}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 2,3,...,r \\ Cap(P(s,x_1)) = \infty. \end{cases}$$

The following example gives a dynamic path from s to ρ. The first variable of the bracket is the capacity of the arc and the second variable is the transit time.

$$\begin{tikzcd} s \arrow[r, "{(2t,t+1)}"] & a \arrow[r, "{(t-1,t-2)}"] & g \arrow[r, "{(t,t)}"] & d \arrow[r, "{(2t-1,t-2)}"] & p \end{tikzcd}$$

Figure 2. Example 2.1.2

Let's assume that a flow traverses the path $P(s,\rho) = (s,a,g,d,\rho)$ which is a directed path from s to ρ. $\tau(s) = 1$, then $\tau(a) = 3, \tau(g) = 4, \tau(d) = 8$ and the exact time of the path is 14 with the departure time of s is 1, because $\alpha(\rho) = 14$. The capacity of the path is 2.

Adopting the strategy same as what the most network flow algorithms applied, we use residual network to measure a remaining flow network for carrying the incremental flow. However, the concept of the residual network should be generalized here while taking consideration of the time factor.

To uniform our presentation, initially we create an *artificial arc*, denoted by $[j,i]$, for each arc $(i,j) \in A$ while let $l[j,i,t] = 0$ and

$$b[j,i,t] = \begin{cases} -b(i,j,u), & 0 \le t = u + b(i,j,u) \le T, u = 0, i, ...T, \\ +\infty, & \text{otherwise.} \end{cases} \tag{2.2}$$

We still denote the network as G. Note that $b[j,i,t]$ may take more than one value for some arcs $[j,i]$ at some time t, since there may exist more than one u satisfying $u + b(i,j,u) = t$. Supposing that $P(s,\rho) = (x_1, x_2, ..., x_r)$ is a dynamic path from s to ρ and $f_p > 0$ is the flow value sent along $P(s,\rho)$, we update the network by an *updating procedure*. Particularly, for $i = 1, 2, ..., r-1$, do:

Case 1: $(x_i, x_{i+1}) \in A^+$. Let

$$l(x_i, x_{i+1}, \tau(x_i)) := l(x_i, x_{i+1}, \tau(x_i)) - f_p;$$
$$l[x_{i+1}, x_i, \alpha(x_{i+1})] := l[x_{i+1}, x_i, \alpha(x_{i+1})] + f_p.$$

Case 2: $[x_i, x_{i+1}] \in A^-$. Let

$$l(x_{i+1}, x_i, \alpha(x_{i+1})) := l(x_{i+1}, x_i, \alpha(x_{i+1})) + f_p;$$
$$l[x_i, x_{i+1}, \tau(x_i)] := l[x_i, x_{i+1}, \tau(x_i)] - f_p,$$

where A^+ and A^- denote sets of arcs with positive and negative transit times respectively ($A = A^+ \cup A^-$). We call such a network as a *time-varying residual network*. We also take figure 1 for example and the capacity and the transit time are same as Table 2.1.1.

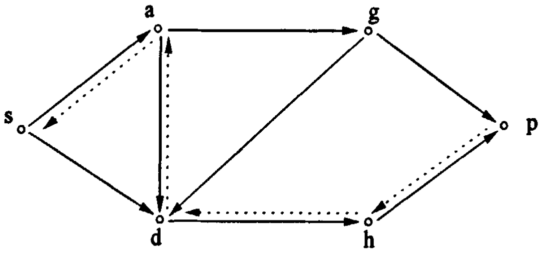

Figure 3. Example 2.1.1(continued)

Suppose that there is a flow flows along the path $P = s - a - d - h - \rho$, starting from s at time 1 and arriving at ρ at time 9. And arrival time at node a, d, h are 3,6,8, respectively. The value of the flow along the path is 5. Then, after doing the updating procedure, the capacity and the transit time will be

changed as follows (others remains unchanged):
$l[a,s,3]=5, b[a,s,3]=-2$; $l[d,a,6]=5, b[d,a,6]=-3$; $l[h,d,8]=5, b[h,d,8]=-2$; $l[\rho,h,9]=5, b[\rho,h,9]=-1$; $l(s,a,1)=1$; $l(a,d,3)=1$; $l(d,h,6)=0$; $l(h,\rho,8)=3$. We obtain the residual network as what in Figure 3.

2.2 A Time-Varying Excess Scaling Algorithm for the Maximum Flow Problem

2.2.1 Basic Definitions and Properties for the Excess Scaling Algorithm

A dynamic path has the *length* k if it consists of k arcs. To measure the length of a dynamic path from node $x \in N$ to node ρ of time exactly t, we use a distance label $d(x,t)$ which is defined as follows:

Definition 4 *A distance function d with respect to the residual network is a function from the couple sets, the node set N and the time set $\{0,1,...,T\}$, to the set of nonnegative integers. We say that a distance function is valid with respect to a flow if it satisfies the following two conditions:*

(i) $d(\rho,t)=0$, $0 \le t \le T$;

(ii) $d(i,u) \le d(j,t)+1$ for every arc $(i,j) \in A^+$ (or $[i,j] \in A^-$) in the time-varying residual network while $0 \le t = u+b(i,j,u) \le T$ (or $0 \le t = u+b[i,j,u] \le T$).

We refer to $d(i,t)$ as the *distance label* of node i at time t and the above two conditions as the *validity conditions.* For a distance label, we have two properties.

Property 1 *If the distance labels are valid, the distance label $d(i,t)$ is a lower bound on the length of the shortest dynamic path from node i to node ρ at time t in the residual network.*

Proof: Let $P(i,\rho)=P(i=i_1,i_2,...,i_{k+1}=\rho)$ be any dynamic path of length

k from node i to node ρ in the time-varying residual network with $t_{j+1} = t_j + b(i_j, i_{j+1}, t_j)$, for $j = 1, 2, ..., k$, where $t_j = \alpha(i_j)$ is the arrival time at node i_j on this path. The validity conditions imply that

$d(i_k, t_k) \leq d(i_{k+1}, t_{k+1}) + 1 = d(\rho, t_{k+1}) + 1 = 1,$

$d(i_{k-1}, t_{k-1}) \leq d(i_k, t_k) + 1 \leq 2,$

$d(i_{k-2}, t_{k-2}) \leq d(i_{k-1}, t_{k-1}) + 1 \leq 3,$

$\ldots$

$d(i, t_1) = d(i_1, t_1) \leq d(i_2, t_2) + 1 \leq k.$

Therefore the claim is true.

Property 2 *If $d(s,t) \geq nT$ $(t < T)$, the time-varying residual network contains no directed path from the source node to the sink node.*

Proof: From Property 1, since $d(s,t)$ is a lower bound on the length of the shortest dynamic path from s to ρ of time exactly t in the residual network, there is no directed path which can contain more than $(nT - 1)$ arcs. Therefore, if $d(s,t) \geq nT$, the time-varying residual network contains no directed path from node s to node ρ at time t.

Let $B(i)$ be the set of arcs emanating to node i. For each node i and each time t, the initial distance labels $d(i,t)$ could be obtained by performing the backward search starting from ρ in the time-varying residual network. For instance, set $d(\rho, t) = 0$ for $0 \leq t \leq T$ and for each node i such that (i,j) (or $[i,j]$) in $B(j)$, let

$$d(i,u) = \min_{\{u | t=u+b(i,j,u), l(i,j,u)>0 \text{ or } t=u+b[i,j,u], l[i,j,u]>0\}} \{d(j,t) + 1\}.$$

Denote $d(i,t)$ to be null if node i can not obtained a distance label at time t. Note that this process can be performed in $O(mT)$ time.

Definition 5 *We say that an arc (i,j) (or $[i,j]$) in the dynamic residual network is admissible if it satisfies the condition that $d(i,u) = d(j,t) + 1$, $t = u + b(i,j,u)$ (or $t = u + b[i,j,u]$), we refer to all other arcs as inadmissible. We also refer* *to a dynamic path from node s to p consisting entirely of admissible arcs as a dynamic admissible path.*

Suppose that P is a dynamic admissible path, the following conditions must be satisfied:

(1) $l(i,j,u) > 0$, $b(i,j,u) > 0$ (or $l[i,j,u] > 0$, $b[i,j,u] > 0$),

(2) $d(i,u) \leq d(j,t) + 1$.

It is obviously that the admissible path P must be a shortest dynamic path in the time-varying residual network.

For a preflow x in the network G, we define the excess of each node $i \in N$ at time t as below:

$$e(i,t) = \sum_{\{j:(j,i)\in A(or[j,i]\in A),u=t-b(j,i,u)(or\, u=t-b[j,i,u])\}} x(j,i,u) - \sum_{\{j:(i,j)\in A(or[i,j]\in A)\}} x(i,j,t).$$

In a preflow, we have $e(i,t) \geq 0$ for each $i \in N \backslash \{s\}$ at each time t. A node is said to be *active at time t* if $e(i,t) > 0$. Node s is the only one with negative excess at any time. Throughout this chapter, we adopt the convention that the source and the sink nodes are never active at any time.

2.2.2 A Time-Varying Excess Scaling Algorithm

We will develop an excess scaling algorithm to solve the time-varying maximum flow problem in this section. Let $e_{max} = \max\{e(i,t) | i \text{ is an active node at time } t, t < T\}$, which provides one measure of the infeasibility of a preflow. The basic idea of the time-varying excess scaling algorithm is to apply an excess scaling technique[3] that systematically reduces the value of e_{max} to 0.

Let Δ denote an upper bound on e_{max}. We refer to a node with $e(i,t) \geq \Delta/2 \geq e_{max}/2$ as *a node with large excess,* and as a node with small excess otherwise. The time-varying excess scaling algorithm always carries out *nonsaturating pushes* which push flow from a node i with a large excess to a node j along an admissible arc (i,j) (or $[i,j]$) at time t. This choice assures that during nonsaturating pushes, the algorithm sends relatively large excess closer to the sink. In order to ensure that the algorithm permits no excess to exceed Δ, it pushes $\delta = \min_{t=u+b(i,j,u)}\{e(i,u), l(i,j,u), \Delta - e(j,t)\}$(or $\delta = \min_{t=u+b[i,j,u]}\{e(i,u), l[i,j,u], \Delta - e(j,t)\}$) units of flow on arc (i,j) (or $[i,j]$). During the whole process of our algorithm, we adopt the following node selection rule: among all nodes with a large excess at any time $t < T$, select a node with the smallest distance label.

Denoting $U = \max_{(i,j)\in A, t<T} l(i,j,t)$, we are now ready to present the algorithm.

```
Algorithm Time-varying excess scaling:
begin
  preprocess;
  $\Delta := 2^{\lceil \log U \rceil}$;
  while $\Delta \geq 1$ do
  begin
    while there is an $e(i,u) \geq \Delta$ do
    begin
      Select a node $i$ with the smallest distance label among all nodes with
      $e(i,u) \geq \Delta$;
      Perform push/relabel($i$) while ensuring that no $e(i,t) \geq \delta$ with $t < T$;
    end;
    $\Delta := \Delta/2$;
  end;
end;
```

```
(a) procedure preprocess:
begin
  Set $x(i,j,t) := 0$ for each $(i,j) \in A$ and each $t$;
  Compute the distance labels $d(i,t)$;
  Let $x(s,j,t) := l(s,j,t)$ for each arc $(s,j) \in A$ with $t = 0,1,2,...,T-1$;
  Calculate $e(i,t)$ for each $i \in N$ and each $t$;
```

```
end;
```

(b) *procedure* compute the distance label $d(i,t)$:

```
begin
  Initialization: Set $d(\rho,t) := 0$, $t = 0,1,2...T$;
                  Let $LIST := \{d(\rho,t), t = 0,1,...,T\}$;
  while $LIST \neq \emptyset$ do
    Select the first label in $LIST$, denoted by $d(j,t)$;
    for all $i \in N$ such that $(i,j) \in A^+$ (or $[i,j] \in A^-$) do
      for each $u$ such that $u = t - b(i,j,u)$ and $l(i,j,u) > 0$ (or $u = t - b[i,j,u]$
          and $l[i,j,u] > 0$) do
      begin
        if $i \neq s$ then $d(i,u) := \min\{d(j,t)+1\}$;
        if $i = s$ then
          $d(i,u) := \max\{d(j,t)+1, nT\}$; $LIST := LIST \cup \{d(i,u)\}$;
      end;
    $LIST := LIST \backslash \{d(j,t)\}$
  end;
end;
```

(c) *procedure* push/relabel(i);

```
begin
  if $0 \leq u \leq T$ and the network contains an admissible arc $(i,j)$ (or $[i,j]$) then
    push $\delta := \min\{e(i,u), l(i,j,u), \Delta - e(j,t)\}$( or $\delta := \min\{e(i,u), l[i,j,u], \Delta -$
    $e(j,t)\}$) units of flow from node $i$ to node $j$ at time $u$;
  else $d(i,u)$ :
    $= \min_{\{(i,j)\in A(i), t=u+b(i,j,u), l(i,j,u)>0;\ \text{or}\ [i,j]\in A(i), t=u+b[i,j,u], l[i,j,u]>0\}}\{d(j,t)+1\}$;
end;
```

Lemma 1 *The algorithm satisfies the following two conditions:*

(1) Each nonsaturating push sends at least $\Delta/2$ units of flows;

(2) No excess ever exceeds Δ.

Proof: We consider the case where a nonsaturating push is applied on arc (i,j) at time $u < T$ (the case on an arc $[i,j]$ can be proved in a very similar way). Since arc (i,j) is admissible, $d(j,t) < d(i,u)$. Notice that node i is the node with the smallest distance label at time u among all nodes with the large excess, so we have $e(i,u) \geq \Delta/2$ and $e(j,t) < \Delta/2$. Since this push is a nonsaturating, it sends $\min\{e(i,u), \Delta - e(j,t)\} \geq \Delta/2$ units of flow. So the first part of the lemma holds. Moreover, this push operation increases the excess of node j at time t only. The new excess of node j at time t is $e(j,t) + \min\{e(i,u), \Delta - e(j,t)\} \leq e(j,t) + \{\Delta - e(j,t)\} \leq \Delta$. So all the node excesses remains less than or equal to Δ. This proves the second part of the lemma.

Theorem 1 *Time-varying excess scaling algorithm correctly computes a maximum flow.*

Proof: The algorithm terminates when the excess resides at the source or at the sink, implying that the current preflow is a flow. Since $d(s,t) = nT$ for any time t, by Property 2, the time-varying residual network contains no path from the source to the sink at any time t. This condition is the termination criterion of the augmenting path algorithm, and the excess residing at the sink is the maximum flow value.

2.2.3 Complexity of the Algorithm

In what follows, we will discuss the time complexity of our algorithm.

Lemma 2 *At any stage of the excess scaling algorithm, each node i at time t with positive excess is connected to node s by a directed dynamic path from node i to node s of time exactly t in the time-varying residual network.*

Before proving this lemma, we introduce the flow decomposition theorem in a time-varying network.

Theorem 2 *Every dynamic path and dynamic cycle*[4] *flow has a unique repre-*

sentation as nonnegative arc flows. Conversely, every nonnegative arc flow can be represented as a dynamic path and dynamic cycle flows (though not necessarily uniquely) with the following property: Every directed dynamic path with positive flow connects a deficit node $(e(i,u) < 0)$to an excess node $(e(i,t) > 0)$ while the time is matching.

The proof of Theorem 2 is very similar with that of the theorem for the static network[3]. To save the space, we omit its proof here. Now, we give the proof for Lemma 2.

Proof: Notice that for a preflow x, we have $e(s,t) \leq 0$ and $e(i,t) \geq 0$ for all $i \in N \backslash \{s\}$ at any time t. By Theorem 2, we can decompose the preflow x with respect to the original network G into nonnegative flows along (1) dynamic paths from node s to node ρ, (2) dynamic paths from node s to active nodes, and (3) flows around dynamic cycles. Note that both (1) and (3) do not contribute to the excess from node s to node ρ. So the lemma is true.

Lemma 3 *For each node $i \in N$ and each time t $(0 \leq t \leq T)$, $d(i,t) < 2nT$.*

Proof: The last time the algorithm relabeled node i at time t, the node had a positive excess, so the time-varying residual network contained a path P of length at most $nT - 2$ from node i to node s. The fact that $d(s,t) = nT$ and that $d(k,u) \leq d(l,t) + 1$ ($t = u + b(k,l,u)$ or $t = u + b[k,l,u]$) for every arc (k,l) (or $[k,l]$) in the path P implies that $d(i,u) \leq d(s,t) + |P| < 2nT$.

Lemma 4 *If the algorithm relabels any node at most k times, the total time spent in finding admissible arcs and relabeling the nodes is $O(k \sum_{i \in N} \sum_{t<T} |A(i)|) = O(kmT)$, where $A(i)$ is the set of arcs adjacent to node i.*

Lemma 5 *In the algorithm each distance label increases at most $2nT$ times. Consequently, the total number of relabel operations is at most $2n^2T^2$.*

Proof: Each relabel operation at node i at time t increase the value of $d(i,t)$ by at least 1 unit. After the algorithm has relabeled node i at most $2nT$ times, $d(i,t) \geq 2nT$. Then the algorithm never again selects node i during an advance operation since for every node k in the partial admissible path, $d(k,t_1) < d(s,t_2) < 2nT$. Thus the algorithm relabels a node at most $2nT$ times and the total number of relabel operations is bounded by $2n^2T^2$.

We shall be using the above results to obtain that the total time spent in finding admissible arcs is $O(nmT^2)$.

Lemma 6 *The algorithm saturates arcs at most nmT^2 times.*

Proof: Between two consecutive saturations of an arc (i,j) (or arc $[i,j]$), both $d(i,u)$ and $d(j,t)$ must increase by at least 2 units. Since the algorithm increases each distance label at most $2nT$ times, this result would imply that the algorithm could saturate any arc at most nT times. Therefore, the total number of arc saturations would be bounded above by nmT^2.

Lemma 7 *The time-varying excess scaling algorithm perform $O(n^2T^2)$ nonsaturating pushes per scaling phase and $O(n^2T^2 \log U)$ pushes in total.*

Proof: Consider the potential function $\Phi = \sum_{i \in N, t<T} e(i,t)d(i,t)/\Delta$. The initial value of Φ at the beginning of the Δ-scaling phase is bounded by $2n^2T^2$ because $e(i,t)$ is bounded by Δ and $d(i,t)$ is bounded by $2nT$. During the push/relabel(i,t) operation, one of the following two cases must apply:

Case 1. The algorithm is unable to find an admissible arc at time u along which it can push flow. In this case the distance label of node i, $d(i,u)$, increase by $\epsilon \geq 1$ units. This relabeling operation increases Φ by at most ϵ units because $e(i,u) \leq \Delta$. Since for each node i the total increase in $d(i,u)$ throughout the running of the algorithm is bounded by $2nT$, the total increase in Φ due to the relabeling of nodes is bounded by $2n^2T^2$.

Case 2. The algorithm is able to identify an arc on which it can push flow at time u, so it performs either a saturating or a nonsaturating push. In either case, Φ decreases. A nonsaturating push on arc (i,j) (or $[i,j]$) at time u sends at least $\Delta/2$ units of flow from node i to node j and since $d(j,t) = d(i,u) - 1$, $t = u + b(i,j,u)$ (or $t = u + b[i,j,u]$), after this operation decreases Φ by at least

1/2 unit. Since the initial value of Φ at the beginning of a Δ-scaling phase is at most $2n^2T^2$ and the increase in Φ during this scaling phase sums to at most $2n^2T^2$, the number of nonsaturating pushes is bounded by $8n^2T^2$.

Lemma 8 *The algorithm identifying a node with the minimum distance label among nodes with excess more than $\Delta/2$ runs in $O(nmT^2 + n^2T^2 \log U)$.*

Proof: We use the following data structure. For each $k = 1, 2, ..., 2nT - 1$, we maintain the list $LIST(k) = \{i \in N : e(i,t) > \Delta/2 \ \ and \ \ d(i,t) = k, t \leq T\}$, and the variable L that is a lower bound on the smallest index k for which $LIST(L)$ is nonempty. We identify the lowest-indexed nonempty list by starting at $LIST(k)$ and sequentially scanning the higher-indexed lists. Because the algorithm performs $O(\log U)$ scaling phases, each phase the algorithm spends $O(nT)$ time to scan the list, also the algorithm performs $O(nmT^2 + n^2T^2 \log U)$ pushes in total which would influence $e(i,t)$. So the algorithm identifying a node with the minimum distance label among nodes with excess more than $\Delta/2$ runs in $O(nmT^2 + n^2T^2 \log U)$.

In summary, we have:

Theorem 3 *The excess scaling algorithm runs in $O(nmT^2 + n^2T^2 \log U)$ time.*

2.2.4 A Numerical Example

To illustrate how the algorithm works, we give an example below. Network G is shown in Figure 4. Given $T = 10$, both capacities and transit times of all arcs are listed in Table 2.2.4.1, while the capacity of the arc or the transit time of the arc is negative we use " $-$ " stands for. Table 2.2.4.2 shows the initial exact distance label $d(i,t)$ for every $i \in N$ and for each time t, " $-$ " stands for "∞".

After *procedure preprocess*, we find that $e(a,1) = 2, e(a,4) = 5, e(a,5) = 6, e(g,4) = 7$, and other excess are equal to 0. So nodes a and g are active nodes and $\Delta = 2^{\lceil \log 21 \rceil} = 32$.

Iteration 1. It is easy to see that there is no node with excess more than

$\Delta = 16$ and $\Delta = 8$, so we begin with $\Delta = 4$. Noting that $e(g, 4) = 7 \geq \Delta$, by the node selection rule, we choose node g do the push/relabel operation at time 4. Arc (g, m) is the admissible arc at time 4. After doing the push operation on arc (g, m), we have $e(m, 5) = 7$ and $e(g, 4) = 0$. Now arc (m, f) is the admissible arc, and $e(f, 6) = 7$ and $e(m, 5) = 0$ after the push operation. Then arc (f, ρ) becomes an admissible arc. $e(\rho, 7) = 7$ and $e(f, 6) = 0$ after the push operation.

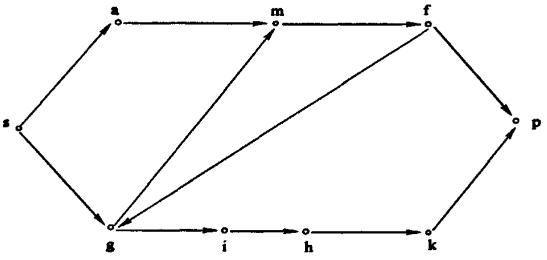

Figure 4. Example 2.2.4.1

Next, we choose node a for doing the push/relabel operation, because $e(a, 5) = 6 \geq \Delta$. Arc (a, m) is the admissible arc at time 5. After the push operation, $e(a, 5) = 1$ and $e(m, 6) = 5$. Arc (m, f) becomes an admissible arc and we have $e(m, 6) = 0$ and $e(f, 7) = 5$. Arc (f, ρ) is the admissible arc at time 7, $e(f, 7) = 0$ and $e(\rho, 9) = 5$.

Node a is the active node at time 4 because $e(a, 4) = 5 \geq \Delta$. Arcs (a, m) and (m, f) are admissible arc at time 4 and 5 respectively. After doing push operations on those arcs, we have $e(a, 4) = 1$, $e(m, 5) = 1$ and $e(f, 6) = 3$. Since no node excess exceeds $\Delta = 4$ now, this iteration is terminated.

Iteration 2. $\Delta := \Delta/2 := 2$. As $e(f, 6) = 3 \geq \Delta$, we choose node f do the push/relabel operation. After the operation, $e(f, 6) = 1$ and $e(\rho, 7) = 2$. Now, because $e(a, 1) = 2 \geq \Delta$, a is the active node at time 1. After doing the push operation, we have $e(a, 1) = 1$ and $e(m, 2) = 1$. At this time, $e(a, 1) = 1, e(a, 4) = 1, e(a, 5) = 1, e(m, 2) = 1, e(m, 5) = 1, e(f, 6) = 1$.

Iteration 3. Let $\Delta = 1$. f is the active node at time 6. Because arc (f, ρ) is a saturating arc at time 6, we can not send any flow along the arc. By the algorithm, we relabel $d(f,6)$ by letting $d(f,6) := d(m,5)+1 := 3$. Now $[f,m]$ is the admissible arc. After doing push operation on arc $[f,m]$, we have $e(m,5)=2$.

Table 2.2.4.1 $l(i,j,t)$ and $b(i,j,t)$ of Figure 4

t	(s,a)	(s,g)	(a,m)	(g,m)	(m,f)	(f,g)	(g,i)	(i,h)	(h,k)	(k,ρ)	(f,ρ)
0	2,1	5,-	0,1	0,-	5,1	1,-	0,-	-,-	0,-	1,-	-,-
1	3,1	6,1	1,1	2,-	6,1	3,-	1,-	-,-	1,-	2,-	-,-
2	4,1	7,2	2,1	4,-	7,1	5,-	2,-	2,-	2,-	3,-	1,-
3	5,1	8,3	3,1	6,-	8,1	7,1	3,-	3,-	3,-	4,-	3,-
4	6,1	9,4	4,1	8,1	9,1	9,2	4,1	4,-	4,-	5,-	5,-
5	7,1	10,5	5,1	10,2	10,1	11,3	5,2	6,1	5,-	6,1	7,-
6	8,1	11,6	6,1	12,3	11,1	13,4	6,3	8,2	6,1	7,2	9,1
7	9,1	12,7	7,1	14,4	12,1	15,5	7,4	10,3	7,2	8,3	11,2
8	10,1	13,8	8,1	16,5	13,1	17,6	8,5	12,4	8,3	9,4	13,3
9	11,1	14,9	9,1	18,6	14,1	19,7	9,6	14,5	9,4	10,5	15,4
10	12,1	15,10	10,1	20,7	15,1	21,8	10,7	16,6	10,5	11,6	17,5

Table 2.2.4.2 Initial distance label $d(i,t)$ of Figure 4

N	$d(\rho,t)$	$d(f,t)$	$d(k,t)$	$d(h,t)$	$d(i,t)$	$d(g,t)$	$d(m,t)$	$d(a,t)$	$d(s,t)$
$t=0$	0	-	-	-	-	-	-	-	90
$t=1$	0	-	-	-	-	-	-	6	-
$t=2$	0	-	-	-	-	-	5	-	90
$t=3$	0	4	-	-	-	-	-	-	90
$t=4$	0	-	-	-	-	3	-	3	90
$t=5$	0	-	1	-	3	-	2	3	-
$t=6$	0	1	1	2	-	-	2	-	-
$t=7$	0	1	1	-	-	-	-	-	-
$t=8$	0	-	-	-	-	-	-	-	-
$t=9$	0	-	-	-	-	-	-	-	-
$t=10$	0	-	-	-	-	-	-	-	-

Relabel $d(m,5)$ by letting $d(m,5) := d(g,4)+1 := 4$. $[m,g]$ becomes an admissible arc at time 5, and $e(m,5)=0$ and $e(g,4)=2$ after the push operation. Relabel $d(g,4)$ by $d(i,5)+1=4$ and $e(g,4)=0$ and $e(i,5)=2$ after doing the push operation on arc (g,i). (i,h), (h,k) and (k,ρ) are the admissible arcs, $e(i,5)=0$, $e(h,6)=0$, $e(k,7)=0$ and $e(\rho,10)=2$ after push operations.

Node a is the next active node we have chosen. Obviously, by the algorithm, the excess flows at time 4 and 5 must be sent back to node s.

m is the active node at time 2. (m, f), (f, g), (g, i), (i, h), (h, k), and (k, ρ) are the admissible arcs respectively. After the push operation, we have $e(m, 2) = 0$, $e(f, 3) = 0$, $e(g, 4) = 0$, $e(i, 5) = 0$, $e(k, 7) = 0$ and $e(\rho, 10) = 1$.

Finally, we choose node a as it is an active node at time 1. The excess flow must be sent back to node s. Then, the algorithm is terminated.

It is clear that the maximum flow of the network $f(\lambda, 10) = 17$. The solution λ can be decomposed into five dynamic paths which are listed below:

$P_1 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 7$ with $Cap(P_1) = 5$;

$P_2 = s - a - m - f - \rho$ which starts from s at $t = 4$ and reaches ρ at $t = 9$ with $Cap(P_2) = 5$;

$P_3 = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with $Cap(P_3) = 4$;

$P_4 = s - g - i - h - k - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 10$ with $Cap(P_4) = 2$;

$P_5 = s - a - m - f - g - i - h - k - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 10$ with $Cap(P_5) = 1$.

2.3 A Time Labeling Algorithm for the Time-Varying Maximum Flow Problem

We have known that the optimal solution to send the maximum flow may not be unique[4]. One may like to find such a maximum flow solution that has the earliest (or the latest) arrival time at ρ, within a given time limit T. It is common in our daily lives. For example, in the market, if some food is going to be sold out, the producer should send their food to the market as early as possible. Sometimes, we may need to package whole goods again at the interchange station. In order to save storage costs, we hope that the arrival time of the flow is as late as possible before a deadline. Sometimes, we may like to find *the shortest duration dynamic path* and we want to find the solution that the duration of each subflow is as short as possible. In this section, we propose a time labeling algorithm to solve the above time-varying maximum flow problems.

2.3.1 Basic Definitions and Properties for the Time Labeling Algorithm

Definition 6 *$P = (x_1, ..., x_r)$ is said to be an earliest dynamic path from x_1 to x_r with $\tau(x_1) = t$ if the arrival time of P satisfies*

$$\alpha(P) = \min_{P' \in \mathcal{P}} \alpha(P')$$

Similarly, P is said to be a latest dynamic path from x_1 to x_r with $\tau(x_1) = t$ if the arrival time of P satisfies

$$\alpha(P) = \max_{P' \in \mathcal{P}} \alpha(P')$$

where $\mathcal{P}$ is the set of all dynamic paths from x_1 to x_r with $\tau(x_1) = t$.

We show the following figure to explain the above definition. The variables in brackets are capacity and transit time of arcs. Let $T = 20$.

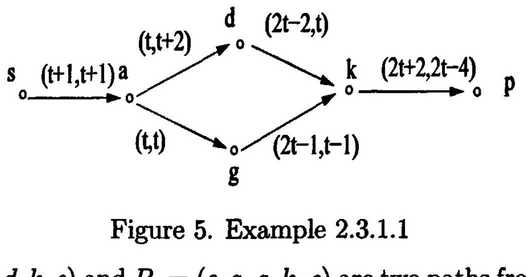

Figure 5. Example 2.3.1.1

$P_1 = (s, a, d, k, \rho)$ and $P_2 = (s, a, g, k, \rho)$ are two paths from node s to node ρ. We assume that the flow starts from node s at time 0. Then we can obtain the arrival times at each node of P_1 are $\alpha(a) = 1, \alpha(d) = 4, \alpha(k) = 8, \alpha(\rho) = 20$ and the arrival times at each node of P_2 are $\alpha(a) = 1, \alpha(g) = 2, \alpha(k) = 3, \alpha(\rho) = 5$. It is obvious that the path P_2 is an earliest arrival dynamic path from s to ρ at time 0 and P_1 is a latest arrival dynamic path from s to ρ at time 0.

For each node $i \in N$ and each time $0 \leq t \leq T$, let $M(i,t)$ denote the arrival time of node ρ in a dynamic path which starts from i at time t. If there is no dynamic path from i to ρ with starting time t or the arrival time of such a path is greater than T, let $M(i,t)$ be null. We define:

Definition 7 *Labels $M(i,t)$ are valid with respect to a flow if they satisfy the following conditions:*

(i) $M(\rho,t) = t$, $0 \leq t \leq T$;

(ii) $M(i,u) = M(j,t)$ for every arc (i,j) (or $[i,j]$) in the time-varying residual network while $t = u + b(i,j,u)$ (or $t = u + b[i,j,u]$) and $0 \leq t \leq T$.

We refer to the above two conditions as *validity conditions.* Obviously, we have:

Property 3 *Suppose that the path $P = (x_1, ..., x_r)$ is the earliest (or latest) dynamic path from x_1 to x_r with $\tau(x_1) = t$, then for each node x_i $(i = 1, 2, ..., r)$, the subpath $P' = (x_i, ..., x_r)$ must be the earliest (or latest) dynamic path from x_i to x_r with $\tau(x_i) = \alpha(x_i)$ under the restriction that for each $(x_k, x_l) \in P$, $\alpha(x_l) = \alpha(x_k) + b(x_k, x_l, \tau(x_k))$ (or $\alpha(x_l) = \alpha(x_k) + b[x_k, x_l, \tau(x_k)]$).*

Property 4 *If labels $M(i,t)$ are valid, $M(i,t)$ is a lower (upper) bound of the arrival time at node ρ of the earliest (latest) dynamic path from node i to ρ in the residual network with $\tau(i) = t$.*

Let $B(i)$ be the set of arcs emanating to node i. For each node i and each time t, the initial labels $M(i,t)$ could be obtained by performing a backward searching. In particular, starting from ρ in the time-varying residual network, for $t = 0, 1, ..., T$ ($t = T, T-1, ..., 0$ in case of the latest maximum flow problem) do

(i) Set $M(\rho,t) = t$ and call ρ to be labeled at time t;

(ii) Examine each labeled node i. For each node j such that (i,j) (or $[i,j]$) in $B(j)$, set $M(i,u) := M(j,t)$, where $t = u + b(i,j,u)$, $l(i,j,u) \geq 0$ (or $t = u + b[i,j,u]$, $l[i,j,u] \geq 0$).

It is easy to see that this process can be performed in $O(mT)$ time.

Note that, for some i and some t, $M(i,t)$ may not be unique. That is to say, there are more than one dynamic path from i to ρ with starting time t in G. Let $LIST(i,t) = \{M(i,t); M(i,t) \neq null\}$ for each $i \in N$ and each time $0 \leq t \leq T$ while keeping all labels in $LIST(i,t)$ in increasing order for the earliest maximum flow case (in decreasing order for the latest maximum flow case), and denote $\Re = \bigcup_{i \in N, 0 \leq t \leq T} LIST(i,t)$.

Definition 8 *An arc (i,j) (or $[i,j]$) in the dynamic residual network is called admissible if it satisfies the condition that $M(i,u) = M(j,t)$, where $t = u + b(i,j,u)$ (or $t = u + b[i,j,u]$). Otherwise, it is called inadmissible. A dynamic path from node s to ρ consisting entirely of admissible arcs is called a dynamic admissible path.*

Note that there is an interesting distinction between the problem of maximum flow and the problem of earliest (or latest) maximum flow. This can be illustrated by the following example.

Example

Consider a time-varying network G as shown in Figure 6, where $T = 6$. Both capacities $l(i,j,t)$ and transit times $b(i,j,t)$ are listed in Table 2.3.1.1, while the capacity of the arc or the transit time of the arc is negative we use " $-$" stands for.

There is a maximum flow λ_1 in G which consists of $P_1 = s - a - \rho$ starting from s at $t = 0$ and reaching ρ at $t = 3$ with $Cap(P_1) = 1$, and $P_2 = s - b - f - \rho$ starting from s at $t = 1$ and reaching ρ at $t = 5$ with $Cap(P_2) = 1$. The total value of flows under λ_1 is equal to 2.

Note that we can have another solution λ_2 which consists of $P_1 = s - a - \rho$

starting from s at $t = 0$ and reaching ρ at $t = 3$ with $Cap(P_1) = 1$, and $P_2 = s - b - \rho$ starting from s at $t = 1$ and reaching ρ at $t = 4$ with $Cap(P_2) = 1$. λ_2 is the

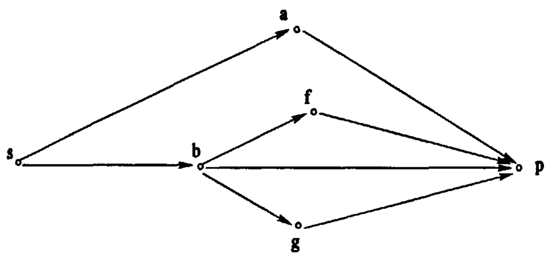

Figure 6. Example 2.3.1.2

Table 2.3.1.1 $l(i,j,t)$ and $b(i,j,t)$ of Figure 6

t	(s,a)	(s,b)	(a,ρ)	(b,ρ)	(b,f)	(f,ρ)	(b,g)	(g,ρ)
0	1, 1	1, 1	0, 3	-, -	0, -	-, 5	-, -	0, 6
1	2, 2	1, 2	1, 2	-, -	1, -	0, 4	-, -	1, 5
2	3, 3	1, 3	2, 1	0, -	2, -	1, 3	0, 1	2, 4
3	4, 4	1, 4	3, -	1, 1	3, 1	2, 2	1, 2	3, 3
4	5, 5	1, 5	4, -	2, 2	4, 2	3, 1	2, 3	4, 2
5	6, 6	1, 6	5, -	3, 3	5, 3	4, -	3, 4	5, 1
6	7, 7	1, 7	6, -	4, 4	6, 4	5, -	4, 5	6, -

earliest maximum flow in G.

Consider another solution λ_3, which consists of $P_1 = s - a - \rho$ starting from s at $t = 0$ and reaching ρ at $t = 3$ with $Cap(P_1) = 1$, and $P_2 = s - g - \rho$ starting from s at $t = 1$ and reaching ρ at $t = 6$ with $Cap(P_2) = 1$. It is the latest maximum flow in G.

2.3.2 A Time-Varying Time Labeling Algorithm

Now we will develop a time-varying labeling algorithm for solving the earliest maximal flow and the latest maximal flow problems.

We first label each node at each time by the backward search starting at the sink node. Then set a preflow in N and choose an active node at a time to push the flow along an admissible arc. The algorithm terminates when $\mathfrak{R} = \emptyset$.

During the whole precess of our algorithm, we adopt the following *admissible arc selecting rule*:

[1] For an active node i, if there are nonartificial arc (i, j) and artificial arc $[i, k]$ to be admissible arcs, we choose nonartificial arc (i, j) first;

[2] If there are some nodes whose excess are not zero on the same admissible path, we choose the node nearest to the sink node first.

Now, we are ready to give the algorithm.

```
Algorithm Labeling algorithm
begin
  preprocess;
  while $\Re \neq \emptyset$ do
  begin
    set $a$:=min{M(i,t) for all $i \in N$, $0 \leq t \leq T$};
      if $M(j,t) = a$ then
        choose node $j$ at time $t$ do push operation;
      begin
      for each node $j \in N_1$ which is the set of nodes whose labels are equal to
          $a$ just now;
          let $LIST(j,t) := LIST(j,t) \backslash \{M(j,t) = a, for\ all\ j \in N_1, t = 0, 1, ..., T\}$;
        if there exist some $LIST(j,t) \neq \emptyset$ then
          set $b := min\{M(j,t), j \in N_1\ and\ the\ departure\ time\ of\ each\ node$
          $satisfies\ definition\ 1\}$;
          set $N_2 := \{j | M(j,t) = b, j \in N_1\}$ and sends all the excess on the
          nodes in $N_1$ to the nodes which belongs to $N_2$;
          relabel $M(j,t) := b$ for all the nodes which belong to $N_1$ except the
          sink node $\rho$;
        else $\Re := \Re \backslash LIST(j,t)$ and send the excess of nodes back to the
            source node;
            $LIST(j,t) := LIST(j,t) \backslash \{M(j,t) = b\}$ for $j \in N_1 - N_2$;
            end;
          else remain the label of each node the same as before;
  end;
end;
```

(a) *procedure* compute labels $M(i,t)$ for the earliest (latest) maximum problem

```
begin
  Initialization: Set $M(\rho,t) := t$, for $t = 0, 1, ......, T$ (for $t = T, T-1, ..., 0$ in
                  case of the latest maximum flow problem);
                  set $LIST(\rho,t) := \{M(\rho,t)\}$, and then put all $LIST(\rho,t)$ in $\Re$,
                  $LIST(i,t) = \emptyset (i \neq \rho)$ at first;
  while $\Re \neq \emptyset$ do
     begin
       Select the first $LIST$ in $\Re$, denoted by $LIST(j,t)$;
     for all $M(j,t) \in LIST(j,t)$ and all the $i \in N$ such that $(i,j) \in A^+$ (or $[i,j]$
         $\in A^-$) do
       for each $u$ such that $u = t - b(i,j,u)$ and $l(i,j,t) > 0$ (or $u = t - b[i,j,t]$
         and $l[i,j,t] > 0$) do
         $M(i,u) := M(j,t)$; $LIST(i,u) := LIST(i,u) \cup \{M(i,u)\}$;
     end;
  $\Re := \Re \backslash LIST(j,t)$;
end;
```

(b) *procedure* preprocess

```
begin
  Set $x(i,j,t) := 0$ for each $(i,j) \in A$ and each $0 \leq t \leq T$;
  Compute labels $M(i,t)$;
  Let $x(s,j,t) := l(s,j,t)$ for each arc $(s,j) \in A$ and each time $t$ obtained by
  procedure $(a)$;
  Calculate $e(i,t)$ for each $i \in N$ at each $0 \leq t \leq T$;
```

```
end;

(c) procedure push
begin
  if the network contains an admissible arc $(i,j)$ (or $[i,j]$) then
    push
      $$\delta := \min_{t=u+b(i,j,u)}\{e(i,u), l(i,j,u)\}$$
    (or $\delta := \min_{t=u+b[i,j,u]}\{e(i,u), l[i,j,u]\}$) units of flow from node $i$ to node $j$
    at time $u$, where $0 \leq u \leq T$;
    if the admissible arc is $[i,j]$ under the condition that the label of two nodes
      are not relabeled then
      set $l(j,i,t) := 0$;
    else $l(i,j,u) := l(i,j,u) - \delta$;
  else remain the excesses of the nodes;
end;
```

Property 5 *Procedure(a) computes all the nodes with their departure time that can arrive at the sink node within duration T.*

Property 6 *If node i ($i \in N\backslash\{\rho\}$) is labeled at time t, there must exist an admissible path from node i at time t to the sink node within the time duration T.*

Property 7 *If P is an admissible path, $(i,j) \in P$, $LIST(j,t)$ contains at least two labels. If there exists one $M(i,u)$ in $LIST(i,u)$ which satisfies $M(i,u) = M(j,t)$, then $LIST(i,u) = LIST(j,t)$.*

Theorem 4 *The labeling algorithm correctly solve the time-varying earliest (or latest) arrival maximum flow problem.*

Proof: The algorithm terminates when the excess resides at the source or at the sink, implying that the current preflow is a flow. Since $\Re = \emptyset$, by Property 5, the time-varying residual network contains no path from the source to the sink at any time t, and the excess residing at the sink is the maximum flow value. Also, in each *push* operation, in fact, we choose the earliest (or the latest) arrival dynamic path.

2.3.3 Complexity of the Algorithm

In this subsection, we will analyze the time complexity of the algorithm.

Lemma 9 *For each node $i \in N$ and each time $0 \leq t \leq T$, $M(i,t) < T+1$ if $M(i,t) \neq null$.*

Proof: Note that the initial value of label $M(i,t)$ is less than $T+1$. During the algorithm, it may be set or changed into a value coming from all existing labels That is to say, the new value of label $M(i,t)$ will not excess T too. Therefore, the claim is true.

Lemma 10 *In the algorithm, each label increases (or decrease) at most T times. Consequently, the total number of relabel operations is at most nT^2.*

Proof: In the algorithm, if we relabel node i at time t, the value of $M(i,t)$ will increase (or decrease in the case of the latest maximum flow) by at least 1. By Lemma 9, the algorithm relabels a node at most T times. Since each node have T labels, the total number of relabel operations is bounded above by nT^2.

Lemma 11 *In each iteration, the total time for finding the active node with the minimum label in some $LIST \in \Re$ is $O(nT)$.*

Proof: It follows the structure of $LIST$ directly.

Lemma 12 *The labeling algorithm runs in $O(mnT^3)$ time.*

Proof: For each iteration, the excess on each node do $O(mT)$ pushes at most. There are at most $O(nT)$ nodes, so it runs in $O(mnT^2)$ time in each iteration. Because there are at most $O(T)$ iterations, the algorithm runs in $O(mnT^3)$ time in total until no excess can be sent to the sink node.

In summary, we have:

Theorem 5 *The time labeling algorithm runs in* $O(mnT^3)$ *time.*

2.3.4 A Numerical Example

We illustrate how the time labeling algorithm works. The network is the same as what in Figure 4. Given $T = 10$, both capacities and transit times of all arcs are listed in Table 2.2.4.1. Table 2.3.4.1 shows the initial exact time label $M(i,t)$ for every $i \in N$ and for each time t, " $-$ " stands for "∞".

Table 2.3.4.1 Initial time label $M(i,t)$ of Figure 4 (earliest)

$LIST(i,t)$	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
$M(\rho,t)$	0	1	2	3	4	5	6	7	8	9	10
$M(f,t)$	-	-	-	7,10	-	-	7	9	-	-	-
$M(k,t)$	-	-	-	-	-	6	8	10	-	-	-
$M(h,t)$	-	-	-	-	-	-	10	-	-	-	-
$M(i,t)$	-	-	-	-	-	10	-	-	-	-	-
$M(g,t)$	-	-	-	-	7,10	-	-	-	-	-	-
$M(m,t)$	-	-	7,10	-	-	7	9	-	-	-	-
$M(a,t)$	-	7,10	-	-	7	9	-	-	-	-	-
$M(s,t)$	7,10	-	7,10	7	9	-	-	-	-	-	-

It is clear that the maximum flow of the network $f(\lambda, 10) = 17$. The solution λ can be decomposed into five dynamic paths which are listed below:

$P_1 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 7$ with $Cap(P_1) = 5$;

$P_2 = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with $Cap(P_2) = 4$;

$P_3 = s - a - m - f - \rho$ which starts from s at $t = 4$ and reaches ρ at $t = 9$ with $Cap(P_3) = 5$;

$P_4 = s - a - m - f - g - i - h - k - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 10$ with $Cap(P_4) = 1$;

$P_5 = s - g - i - h - k - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 10$ with $Cap(P_5) = 2$.

After *procedure preprocess*, we find $e(a,1) = 1, e(a,4) = 5$,$e(a,5) = 6$, $e(g,4) = 7$ and the other excesses are 0.

Iteration 1. We find that the minimum label is 7, so we choose node a and node g as active nodes for *push* operation. We can obtain three pathes as follows: $P_1 = s - a - m - f - g - m - f - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 7$ with $Cap(P_1) = 1$; $P_2 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 7$ with $Cap(P_2) = 4$; $P_3 = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with $Cap(P_3) = 4$. At this time, $e(a,1) = 1$, $e(g,4) = 3$. $LIST(a,1)$, $LIST(m,2)$, $LIST(f,3)$ and $LIST(g,4)$ are nonempty. $\Re := \Re \backslash \{M(i,t) = 7\}$.

Iteration 2. Now the active node a with the label which is equal to 9 is minimum. We choose it for *push* operation. $(a,m),(m,f),(f,\rho)$ are admissible arcs. We obtain the path as follows: $P_4 = s - a - m - f - \rho$ which starts from s at $t = 4$ and reaches ρ at $t = 7$ with $Cap(P_4) = 5$. We only leave 1 unit flow on node a. Because the LISTS of all the nodes whose labels are equal to 9 just now are empty, we send the one unit flow back. $\Re := \Re \backslash \{M(i,t) = 9\}$.

Iteration 3. Now the active node a and g with the minimum labels which is equal to 10. First, we choose node g for *push* operation. We obtain the path as follows: $P_5 = s - g - i - h - k - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 10$ with $Cap(P_5) = 3$. Because the one unit of node a can not be sent to ρ, we send the flow back. $\Re := \Re \backslash \{M(i,t) = 10\}$.

Up to now, $\Re = \emptyset$, the algorithm terminates.

Now we use the same example to illustrate the algorithm for the latest. The exact time label $M(i,t)$ for every $i \in N$ are showed in Table 2.3.4.2, " $-$ " stands for "∞".

It is clear that the maximum flow of the network $f(\lambda, 10) = 17$. The solution λ can be decomposed into five dynamic paths which are listed below:

$P_1 = s - g - i - h - k - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 10$

with $Cap(P_1) = 4$.

$P_2 = s - a - m - f - \rho$ which starts from s at $t = 4$ and reaches ρ at $t = 9$ with $Cap(P_2) = 5$.

$P_3 = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with $Cap(P_3) = 4$.

$P_4 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 7$ with $Cap(P_4) = 3$.

$P_5 = s - a - m - f - g - m - f - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 7$ with $Cap(P_5) = 1$.

Table 2.3.4.2 Initial time label $M(i,t)$ of Figure 4 (latest)

$LIST(i,t)$	$t=10$	$t=9$	$t=8$	$t=7$	$t=6$	$t=5$	$t=4$	$t=3$	$t=2$	$t=1$	$t=0$
$M(\rho,t)$	10	9	8	7	6	5	4	3	2	1	0
$M(f,t)$	-	-	-	9	7	-	-	10,7	-	-	-
$M(k,t)$	-	-	-	10	8	6	-	-	-	-	-
$M(h,t)$	-	-	-	-	10	-	-	-	-	-	-
$M(i,t)$	-	-	-	-	-	10	-	-	-	-	-
$M(g,t)$	-	-	-	-	-	-	10,7	-	-	-	-
$M(m,t)$	-	-	-	-	9	7	-	-	10,7	-	-
$M(a,t)$	-	-	-	-	-	9	7	-	-	10,7	-
$M(s,t)$	-	-	-	-	-	-	9	7	10,7	-	10,7

After *procedure preprocess*, we find $e(a,1) = 1, e(a,4) = 5, e(a,5) = 6, e(g,4) = 7$ and the other excesses are 0.

*Iteration 1.*The active node g and a with the maximum labels which are equal to 10. First, we choose node g for *push* operation. We obtain the path as follows: $P_1 = s - g - i - h - k - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 10$ with $Cap(P_1) = 4$. Second, we choose node a. The one unit flow is stopped at node g because of the saturating arc (g,i). At this time, $LIST(a,1)$, $LIST(m,2)$, $LIST(f,3)$ and $LIST(g,4)$ are nonempty. $\Re := \Re \backslash \{M(i,t) = 10\}$.

Iteration 2. Now the active node a with the maximum label which is equal to 9, we choose it for *push* operation. We only leave 1 unit flow on node

a. Because the LISTS of all the nodes whose labels are equal to 9 just now are empty, we send the one unit flow back. We obtain the path as follows: $P_2 = s - a - m - f - \rho$ which starts from s at $t = 4$ and reaches ρ at $t = 9$ with $Cap(P_2) = 5$. $\Re := \Re \backslash \{M(i,t) = 9\}$.

Iteration 3. Now the active nodes a and g with the maximum labels which are equal to 7. First, we choose node a for *push* operation. We obtain the path as follows: $P_3 = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with $Cap(P_3) = 4$. Second, we choose node g for *push* operation. Arc $(g, m), (m, f), (f, \rho)$ are admissible arcs. We obtain two pathes as follows: $P_4 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 7$ with $Cap(P_4) = 3$; $P_5 = s - a - m - f - g - m - f - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 7$ with capacity 1. $\Re := \Re \backslash \{M(i,t) = 7\}$.

Up to now, $\Re = \emptyset$, the algorithm terminates.

2.3.5 The Other Application of the Time-Varying Time Labeling Algorithm

Sometimes, we want to send some putrescible commodity from one place to another in the cargo-transportation network. We could not spend much time on the way, otherwise the commodity will go to bad easily. Therefore, we need to find such a maximum flow that the time duration between the departure time and the arrival time of the flow is as short as possible. We would like to point out that, the previous time labeling algorithm can solve this problem either.

Definition 9 $P = (x_1, ..., x_r)$ *is said to be a shortest duration dynamic path from* x_1 *to* x_r *if the time duration between the departure time at node* x_1 *and the arrival time at node* x_r *is minimized.*

Let's use the example (refer to Figure 4) to explain the definition. All arc capacities and the transit times are the same as what in Table 2.2.4.1. We can see that $P = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at

$t = 7$ is the shortest duration dynamic path, because the time duration is 4 and it is the shortest one.

We would like to point out that, the earliest (or latest) maximum flow problem is different from the shortest time duration maximum flow problem. One can find the difference between these two problems by the following example.

Example

Consider a time-varying network G as shown in Figure 7, where $T = 8$, both capacities and transit times of all the arcs are list in Table 2.3.5.1.

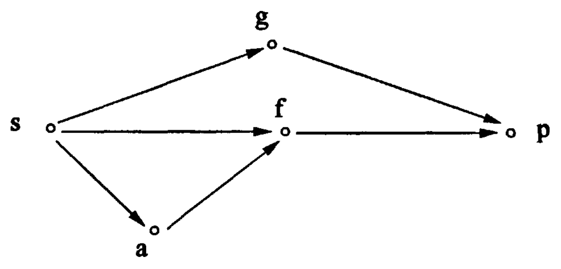

Figure 7. Example 2.3.5.1

Table 2.3.5.1 $l(i,j,t)$ and $b(i,j,t)$ of Figure 7

t	(s,g)	(s,a)	(s,f)	(a,f)	(g,ρ)	(f,ρ)
0	-, -	0, -	1, 5	0, -	0, -	3, -
1	-, -	1, -	2, 6	1, -	1, -	3,-
2	0, 1	2, -	3, 7	2, -	2, -	3, -
3	1, 2	3, 1	4, 8	3, -	3, 1	3, -
4	2, 3	4, 2	5, 9	4, 1	4, 2	3, 1
5	3, 4	5, 3	6, 10	5, 2	5, 3	3, 2
6	4, 5	6, 4	7, 11	6, 3	6, 4	3, 3
7	5, 6	7, 5	8, 12	7, 4	7, 5	3, 4
8	6, 7	8, 6	9, 13	8, 5	8, 6	3, 5

The solution of the earliest maximal flow problem is as follows:

$P_1 = s - f - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 7$ with $Cap(P_1) = 1$;

$P_2 = s - a - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with

$Cap(P_2) = 2$;

$P_3 = s - g - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 8$ with $Cap(P_3) = 1$.

But the solution of the shortest duration maximum flow problem is as follows:

$P_1 = s - a - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with $Cap(P_1) = 3$;

$P_2 = s - g - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 8$ with $Cap(P_2) = 1$.

The total value of the flow is 4 which is the same as the previous one.

Theorem 6 *For a given time-varying network G and a time duration T, there exist an optimal solution that transits the maximum flow from the source node s to the sink node with the shortest duration.*

The proof of this theorem is very similar with that of Theorem 3.10[4], we omit its proof here.

In the above section, we can see that the time labeling algorithm can label the arrival time of all the nodes at any time. So it is easy to find the flow whose duration is as short as possible.

We also use Figure 4 to show this fact. The exact time label $M(i,t)$ for every $i \in N$ and for each time t are the same as what are in Table 2.3.4.1.

We can see that $M(s,3) = 7$, the time duration between the departure time and the arrival time is 4, it is the shortest one. So we choose this flow at first which flows along path $P_1 = s-a-m-f-\rho$ starting from s at $t = 3$ and reaching ρ at $t = 7$ with $Cap(P_1) = 4$. Next, we choose node s with the departure time 4 and get path $P_2 = s-a-m-f-\rho$ which starts from s at $t = 4$ and reaches ρ at $t = 9$ with $Cap(P_2) = 5$. The time duration is the same as the previous one. We now have $P_3 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 7$ with $Cap(P_3) = 3$. Then we choose the flow whose departure time is 2 and the arrival time is 10, and flows along path $P_4 = s-g-i-h-k-\rho$ which starts from s at $t = 2$ and reaches ρ at $t = 10$ with $Cap(P_4) = 4$. At last, the largest duration is 10. The corresponding path is $P_5 = s - a - m - f - g - i - h - k - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 10$ with $Cap(P_5) = 1$. The total value of the flow is 17.

The above two sections discuss two algorithms respectively for the time-varying maximum flow problem with zero waiting time. It is clear that the complexity of them are much better than what we have known. In the next chapter, we will discuss the problem in the case where waiting at any node is arbitrarily allowed.

Chapter 3

TIME-VARYING MAXIMUM FLOW PROBLEMS WITH ARBITRARY WAITING TIME

In this chapter, we will discuss the time-varying maximum flow problems with arbitrary waiting time constraints, that is, the flow can wait at any nodes without the time limitation. Figure 8 shows an example where three numbers inside each pair of brackets associated with an arc are t, $l(i,j,t)$ and $b(i,j,t)$ respectively.

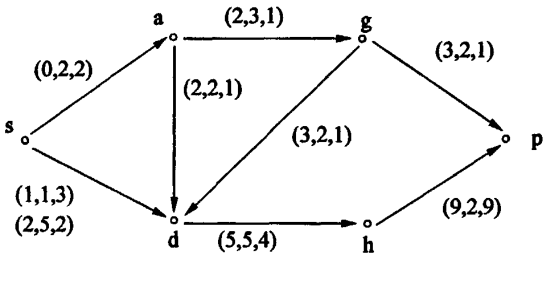

Figure 8. Example 3.1

While the flow flows along arc (s,d) at time 1, it can not flow along arc (d,h) if waiting at any node is forbidden. But if waiting at any node is permitted, the flow can wait at node d until time 5, then starts to flow along arc (d,h). In generally speaking, the maximum flow for the problem with arbitrary waiting time will not less than that for the problem with zero waiting time.

3.1 Definitions and Formulations for the Time-Varying Maximum Flow Problem with Arbitrary Waiting Time

Most definitions and formulations in this chapter are same as Chapter 2, so in this section, we only give some definitions and formulations which are different from Chapter 2.

Let $x(i,j,t)$ be the flow transshipped on arc (i,j) starting at time t. Let $x(i,t)$ be the flow which waits at node i during time $[t, t+1)$. The maximum flow problem with arbitrary waiting time can be presented formally as below:

Maximize $\quad v$

s.t.

$$\begin{cases} \sum_{t=0}^{T-1}\{\sum_{\{j:(i,j)\in A\}} x(i,j,t) - \sum_{\{j:(j,i)\in A, u=t-b(j,i,u)\}} x(j,i,u)\} = v, \\ \qquad \text{for } i = s \\ \sum_{\{j:(i,j)\in A\}} x(i,j,t) + x(i,t) - \sum_{\{j:(j,i)\in A, u=t-b(j,i,u)\}} x(j,i,u) - \\ \quad x(i,t-1) = 0, \text{for all } i \in N\backslash\{s,\rho\},\ t = 1,2,...,T-1 \\ \sum_{t=1}^{T}\{\sum_{\{j:(i,j)\in A\}} x(i,j,t) - \sum_{\{j:(j,i)\in A, u=t-b(j,i,u)\}} x(j,i,u)\} = -v, \\ \qquad \text{for } i = \rho \\ 0 \le x(i,j,t) \le l(i,j,t),\ \forall (i,j) \in A,\ 0 \le t \le T. \end{cases} \tag{3.1}$$

Definition 10 *Let $P(s,x) = (s = x_1, ..., x_r = x)$ be a directed path from s to x and assume that a flow traverses the path. Let $\tau(x_1) = 0$ and define recursively*

$$\tau(x_i) = \tau(x_{i-1}) + b(x_{i-1}, x_i, \tau(x_{i-1})) + \omega(x_i), \qquad \textit{for } i = 2, ..., r.$$

The departure time of the flow at a node x_i on P is defined as $\tau(x_i)$ if $1 \le i < r$, $\omega(x_i)$ is the waiting time of node x_i. Accordingly, the arrival time of the flow at *a node x_i on P is defined as*

$$\alpha(x_1) = 0; \quad \alpha(x_i) = \tau(x_{i-1}) + b(x_{i-1}, x_i, \tau(x_{i-1})), \qquad \textit{for } i = 2, ..., r.$$

Definition 11 *Let $P(s,x) = (s = x_1, ..., x_r = x)$ be a dynamic f − augmenting path from s to x. The capacity of P is defined as $Cap(P)$=*$\min\{\min_{(x,y)\in A(P)} l(x,y,\tau(x)), \min_{[x,y]\in A(P)} l[x,y,\tau(x)], \min_{v\in V(P),\alpha(x)\le t'<\tau(x)} l(x,t'), \min_{v\in V(P),\alpha(x)\ge t'>\tau(x)} l[x,t']\}$.
$l(x,t)$ is defined as the capacity of the node x, which represents the maximum amount of flow that can wait at x during the time period $[t, t+1)$. We define $l(x,t) = \infty$. So the capacity of P is defined as

$$Cap(P) = \min\{\min_{(x,y)\in A(P)} l(x,y,\tau(x)), \min_{[x,y]\in A(P)} l[x,y,\tau(x)]\}.$$

Since a feasible f-augmenting path from y to ρ of time exactly $t-1$ is a feasible f-augmenting path from y to ρ of time exactly t, namely, when we can label y with $d(y, t-1)$ after it is labeled with $d(y,t)$. In fact, if y is labeled with $d(y,t)$, then , for any $t' < t$, y can be labeled with $d(y,t')$.

3.2 A Time-Varying Excess Scaling Algorithm for the Maximum Flow Problem

3.2.1 Basic Definitions and Properties for the Excess Scaling Algorithm

Definition 12 *A distance function d with respect to the residual network is a function from the couple sets, the node set N and the time set $\{0, 1, ..., T\}$, to the set of nonnegative integers. We say that a distance function is valid with respect to a flow if it satisfies the following two conditions:*

(i) $d(\rho,t) = 0$, $0 \le t \le T$;

(ii) $d(i,u) \le \min\{d(j,t)+1, d(i,u+1)\}$ for every arc $(i,j) \in A^+$ (or $[i,j] \in$

A^-) in the time-varying residual network while $0 \le t = u + b(i,j,u) \le T$ (or $0 \le t = u + b[i,j,u] \le T$).

We refer to $d(i,t)$ as the *distance label* of node i at time t and the two conditions as the *validity conditions.* The property 1 which is illustrated in section 2.2.1 remains true in this section.

3.2.2 A Time-Varying Excess Scaling Algorithm

All steps are same as those of the algorithm for the zero waiting time except the following procedure of computing the distance label $d(i,t)$:

```
begin
  Initialization: Set $d(\rho,t) := 0,\ t = 0,1,2...T$;
                  Let $LIST := \{d(\rho,t), t = 0,1,...,T\}$;
  while $LIST \neq \emptyset$ do
    Select the first label in $LIST$, denoted by $d(j,t)$;
    for all $i \in N$ such that $(i,j) \in A^+$ (or $[i,j] \in A^-$) do
      for each $u$ such that $u = t - b(i,j,u) > 0$ and $l(i,j,u) > 0$
        (or $u = t - b[i,j,u]$ and $l[i,j,u] > 0$) do
      begin
        if $i \neq s$ then $d(i,u) := \min\{d(j,t) + 1, d(i,u+1)\}$;
        else $i = s$ then $d(i,u) := \max\{d(j,t) + 1, nT\}$;
        $LIST := LIST \cup \{d(i,u)\}; u := u - 1$;
      end;
    $LIST := LIST \backslash \{d(j,t)\}$;
  end;
end;
```

The complexity of the excess scaling algorithm for the maximum flow problem with arbitrary waiting time is the same as the case for the problem with zero waiting time. Thus, we omit it here.

3.2.3 A Numerical Example

We use Figure 4 to illustrate how the excess scaling algorithm works in the case of the arbitrary waiting time. We allow the flow wait at any node for any time. The transit time and the capacity of arcs are same as what in Table 2.2.4.1. Table 3.2.3.1 shows the distance label $d(i,t)$ for every $i \in N$ and for each time t, where " – " stands for "∞".

Table 3.2.3.1 The distance label $d(i,t)$ of Figure 4 (arbitrary)

N	$d(\rho,t)$	$d(f,t)$	$d(k,t)$	$d(h,t)$	$d(i,t)$	$d(g,t)$	$d(m,t)$	$d(a,t)$	$d(s,t)$
$t=0$	0	-	-	-	-	-	-	-	90
$t=1$	0	1	1	2	3	3	2	3	90
$t=2$	0	1	1	2	3	3	2	3	90
$t=3$	0	1	1	2	3	3	2	3	90
$t=4$	0	1	1	2	3	3	2	3	90
$t=5$	0	1	1	2	3	-	2	3	-
$t=6$	0	1	1	2	-	-	2	-	-
$t=7$	0	1	1	-	-	-	-	-	-
$t=8$	0	-	-	-	-	-	-	-	-
$t=9$	0	-	-	-	-	-	-	-	-
$t=10$	0	-	-	-	-	-	-	-	-

After doing *procedure preprocess*, we find that $e(a,1) = 2, e(a,2) = 3, e(a,3) = 4, e(a,4) = 5, e(g,2) = 6, e(g,4) = 7$, and the excess of other nodes are equal to 0. So nodes a and g are active nodes and $\Delta = 2^{\lceil \log 21 \rceil} = 32$.

Iteration 1. It is easy to see that there is no node with excess more than $\Delta = 16$ and $\Delta = 8$, so we begin with $\Delta = 4$. Noting that $e(g,4) = 7 \geq \Delta$, we choose node g do the push/relabel operation at time 4. Arc (g,m) is the admissible arc at time 4. After doing the push operation on arc (g,m), we have $e(m,5) = 7$ and $e(g,4) = 0$. Now arc (m,f) is the admissible arc, and $e(f,6) = 7$ and $e(m,5) = 0$ after the push operation. Then arc (f,ρ) becomes an admissible arc. $e(\rho,7) = 7$ and $e(f,6) = 0$ after the push operation.

Next, we still choose node g do the push operation because $e(g,2) = 6 \geq \Delta$. The flow can not flow along any arc at time 2, so it waits until at time 4. Arc

(g, m) is admissible, so $e(m, 5) = 1, e(g, 4) = 5$. Then we choose node a do the push operation because $e(a, 4) = 5 \geq \Delta$. (a, m) is an admissible arc. $e(a, 4) = 1$ and $e(m, 5) = 4+1 = 5$. Now we choose node m do the push operation. (m, f) is admissible arc, the residual capacity is 3, so $e(m, 5) = 2$ and $e(f, 6) = 3$. Because $e(g, 4) = 5$, we choose node g do the push operation. $(g, i), (i, h), (h, k), (k, \rho)$ are admissible arcs, we push the flow of 4 units and arrives at the sink node at time 10, only left $e(g, 4) = 1$. Then we choose node a do the push operation, $e(a, 3) = 4$. (a, m) is an admissible arc. $e(m, 4) = 3, e(a, 3) = 1$.

Iteration 2. $\Delta := \Delta/2 := 2$. As $e(f, 6) = 3 \geq \Delta$, we choose node f do the push/relabel operation. After the operation, $e(f, 6) = 1$ and $e(\rho, 7) = 2$. Now, because $e(m, 4) = 3 \geq \Delta$, m is the active node at time 4. After doing the push operation, we have $e(f, 5) = 3$. Because arc (f, ρ) is saturated at time 6, the flow waits until at time 7. At last, $e(f, 7) = 0$ and $e(\rho, 9) = 3$. Then we choose node m do the push operation because $e(m, 5) = 2$. Because arc (m, f) at time 5 is 5 is saturated, the flow waits until at time 6. $(m, f), (f, \rho)$ are admissible arcs. $e(\rho, 9) = 2$. Then we choose node a do the push operation at time 2. $(a, m), (m, f), (f, \rho)$ are admissible arcs. When the flow arrives at node f at time 4, it must wait until at time 7, it can flow along the arc (f, ρ). $e(a, 2) = 1$ is left. At last, we choose node a do the push operation at time 1 because $e(a, 1) = 2$. (a, m) is admissible arc, $e(a, 1) = 1$, $e(m, 2) = 1$.

Iteration 3. $\Delta := \Delta/2 := 1$. We choose node f at time 6, the flow waits at f until at time 7, then it arrives at the sink node at time 9. Then we choose node m at time 2 do the push operation. The flow arrives at node f at time 3, it waits until at time 7, then it arrives at the sink node at time 9. Then we choose node a at time 2 do the push operation. At first, the flow waits at node a until at time 5, then the flow flows along arcs $(a, m), (m, f), (f, \rho)$ and arrives at the sink node at time 9. At last, we choose node a at time 3 do the push operation. The flow waits until at time 5, then it arrive at the sink node at time 9 along arcs $(a, m), (m, f), (f, \rho)$. The excess flow must be sent back to node s. Then, the algorithm is terminated.

It is clear that the maximum flow of the network $f(\lambda, 10) = 24$. The solution λ can be decomposed into ten dynamic paths which are listed below:

$P_1 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 7$ with $Cap(P_1) = 7$ and there is no waiting at any node;

$P_2 = s - g - i - h - k - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 10$ with $Cap(P_2) = 4$ and the flow waits at node g at time 2 until at time 4;

$P_3 = s - g - m - f - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 9$ with $Cap(P_3) = 1$ and the flow waits at node g at time 2 until at time 4, also the flow waits at node f at time 6 until at time 7;

$P_4 = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 7$ with $Cap(P_4) = 2$ and there is no waiting at any node;

$P_5 = s - a - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 9$ with $Cap(P_5) = 3$ and the flow waits at node f at time 5 until at time 7;

$P_6 = s - a - m - f - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 9$ with $Cap(P_6) = 2$ and the flow waits at node m at time 5 until at time 6;

$P_7 = s - a - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 9$ with $Cap(P_7) = 2$ and the flow waits at node f at time 4 until at time 7;

$P_8 = s - a - m - f - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 9$ with $Cap(P_8) = 1$ and the flow waits at node f at time 3 until at time 7;

$P_9 = s - a - m - f - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 9$ with $Cap(P_9) = 1$ and the flow waits at node a at time 2 until at time 5;

$P_{10} = s - a - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 9$ with $Cap(P_{10}) = 1$ and the flow waits at node f at time 3 until at time 5.

It is clear that the total value of flows is 24.

3.3 A Time-Varying Time Labeling Algorithm for the Maximum Flow Problem

3.3.1 Basic Definitions and Properties for the Time Labeling Algorithm

Definition 13 *Labels $M(i,t)$ are valid with respect to a flow if they satisfy the following conditions:*

(i) $M(\rho,t)=t$, $0 \le t \le T$;

(ii) $M(i,u)=\{M(j,t), M(i,u+1)\}$ for every arc (i,j) (or $[i,j]$) in the time-varying residual network while $t=u+b(i,j,u)$ (or $t=u+b[i,j,u]$) and $0 \le u+1, t \le T$.

We refer to the above two conditions as *validity conditions.*

3.3.2 A Time-Varying Time Labeling Algorithm

All steps are same as those in the algorithm for the case of the zero waiting time except the following procedure of computing the time label $M(i,t)$:

```
begin
 Initialization: Set $M(\rho,t) := t$, for $t = 0,1,......,T$ (for $t = T, T-1, ..., 0$ in
                 case of the latest maximum flow problem), set $LIST(\rho,t) :=$
                 $\{M(\rho,t)\}$, and then put all $LIST(\rho,t)$ in $\Re$, $LIST(i,t) = \emptyset$
                 $(i \neq \rho)$ at first;
 while $\Re \neq \emptyset$ do
   begin
     Select the first $LIST$ in $\Re$, denoted by $LIST(j,t)$;
     for all $M(j,t) \in LIST(j,t)$ and all the $i \in N$ such that $(i,j) \in A^+$ (or
         $[i,j] \in A^-$) do
       for each $u$ such that $u = t - b(i,j,u)$ and $l(i,j,t) > 0$ (or $u = t - b[i,j,t]$
           and $l[i,j,t] > 0$) do
```

```
            $M(i,u) := \{M(j,t), M(i,u+1)\}$;
            $LIST(i,u) := LIST(i,u) \cup \{M(i,u)\}$;
            let $u := u-1$;
        end;
      end;
   $\Re := \Re \backslash LIST(j,t)$;
end;
```

The complexity of the time labeling algorithm for the maximum flow problem with arbitrary waiting time is same as the case of the zero waiting time. So we omit it here.

3.3.3 A Numerical Example

We also use Figure 4 to explain the algorithm. The time labels are listed in Table 3.3.3.1, where " $-$ " stands for "∞".

Table 3.3.3.1 The time label $M(i,t)$ of Figure 4 (arbitrary)

$LIST(i,t)$	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
$M(p,t)$	0	1	2	3	4	5	6	7	8	9	10
$M(f,t)$	-	7,9,10	7,9,10	7,9,10	7,9	7,9	7,9	9	-	-	-
$M(k,t)$	-	6,8,10	6,8,10	6,8,10	6,8,10	6,8,10	8,10	10	-	-	-
$M(h,t)$	-	8,10	8,10	8,10	8,10	8,10	10	-	-	-	-
$M(i,t)$	-	10	10	10	10	10	-	-	-	-	-
$M(g,t)$	-	7,9,10	7,9,10	7,9,10	7,9,10	-	-	-	-	-	-
$M(m,t)$	-	7,9,10	7,9,10	7,9	7,9	7,9	9	-	-	-	-
$M(a,t)$	-	7,9,10	7,9	7,9	7,9	9	-	-	-	-	-
$M(s,t)$	7,9,10	7,9,10	7,9,10	7,9	9	-	-	-	-	-	-

After doing *procedure preprocess*, we find that $e(a,1) = 2, e(a,2) = 3, e(a,3) = 4, e(a,4) = 5, e(a,5) = 6, e(g,2) = 6, e(g,4) = 7$ and other excess are equal to 0. So nodes a and g are active nodes.

Iteration 1. We find that the minimum label is 7, so we choose node a and node g as active nodes for *push* operation. We can obtain five pathes as follows:

$P_1 = s - a - m - f - \rho$ which starts from s at $t = 0$ and reaches ρ at $t = 7$ with $Cap(P_1) = 1$ and the flow waits at node f at time 3 until at time 6;

$P_2 = s - a - m - f - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 7$ with $Cap(P_2) = 1$ and the flow waits at node a at time 1 time until at time 2, waits at node f at time 4 until at time 7;

$P_3 = s - a - m - f - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 7$ with $Cap(P_3) = 1$ and waits at node f at time 4 until at time 6;

$P_4 = s - a - m - f - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 7$ with $Cap(P_4) = 2$ and waits at node a at time 2 until at time 3, waits at node f at time 5 until at time 6;

$P_5 = s - g - m - f - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 7$ with $Cap(P_5) = 4$ and waits at node g at time 2 until at time 4.

At this time, $e(a, 1) = 1$, $e(a, 2) = 2, e(g, 4) = 2$. All $LISTS$ are nonempty. $\Re := \Re \backslash \{M(i, t) = 7\}$, arc (f, ρ) is saturated at time 6.

Iteration 2. Now the active node a with the label which is equal to 9 is minimum. We choose it for *push* operation. $(a, m), (m, f), (f, \rho)$ are admissible arcs. We obtain the path as follows: $P_6 = s - a - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 9$ with $Cap(P_6) = 1$ and waits at node f at time 5 until at time 7. We leave 3 units flow on node a. $P_7 = s - a - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 9$ with $Cap(P_7) = 3$ and waits at node a at time 3 until at time 4, waits at node f at time 6 until at time 7. $P_8 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 9$ with $Cap(P_8) = 3$ and waits at node f at time 6 until at time 7. We leave $(4 + 2 =)6$ units flow on node g at time 4. At this time, arc (m, f) is saturated at time 5. $P_9 = s - g - m - f - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 9$ with $Cap(P_9) = 1$ and waits at node m at time 5 until at time 6. We leave 5 units flow on node g at time 4. $P_{10} = s - a - m - f - \rho$ which starts from s at $t = 3$ and reaches ρ at $t = 9$ with $Cap(P_{10}) = 1$ and waits at node m at time 5 until at time 6. We leave 4 units flow on node a at time 4. $P_{11} = s - a - m - f - \rho$

which starts from s at $t = 3$ and reaches ρ at $t = 9$ with $Cap(P_{11}) = 2$ and waits at node a at time 4 until at time 5. We leave 2 units flow on node a at time 4. At this time, arc (f, ρ) is saturated at time 7. Because the LISTS of all the nodes whose labels are equal to 9 just now are empty, except those whose labels are equal to 10. we send the one unit flow back. $\Re := \Re \backslash \{M(i,t) = 9\}$.

Iteration 3. Now the active node a and g with the minimum labels which is equal to 10. First, we choose node g for *push* operation. We obtain the path as follows: $P_{12} = s - g - i - h - k - \rho$ which starts from s at $t = 2$ and reaches ρ at $t = 10$ with $Cap(P_{12}) = 3$. $P_{13} = s - g - i - h - k - \rho$ which starts from s at $t = 1$ and reaches ρ at $t = 10$ with $Cap(P_{13}) = 1$ and waits at node g at time 2 until at time 4. At this time, arc (g, i) is saturated at time 4. Because the one unit of node a can not be sent to ρ, we send the flow back. $\Re := \Re \backslash \{M(i,t) = 10\}$.

Up to now, $\Re = \emptyset$, the algorithm terminates.

It is clear that the total value of the flow is 24.

Chapter 4

CONCLUSION

In our daily lives, there are a lot of problems are concerned with a time limit T while some parameters of the model are changed over time. In view of this, in this thesis, we study the maximum flow problem in the time-varying network, and one can find that, such a problem exists everywhere.

First, in Chapter 1, we make a brief review on the maximum flow problem, including its models, algorithms and applications, and we believe that it can help us to find some common features of the maximum flow problem. And then, we simply describe the problem in a time-varying network. In Chapter 2 and Chapter 3, we study the time-varying maximum flow problem with two different waiting time constraints, one is called zero waiting time and the other is arbitrary waiting time. For each case, we present two algorithms: the excess scaling algorithm and the time labeling algorithm. The later one can solve the earliest and latest arrival maximum flow problem, i.e., the solution must arrive at the sink node as early as possible or as late as possible. Moreover, it can be applied to solve the problem that the time duration of the flow is as short as possible. For each algorithm, we prove the correctness and the complexity of the algorithm. To illustrate how the algorithm works, we give a numerical example for each algorithm.

Our algorithms presented in this thesis are developed based on the preflow technique. At our best knowledge, it can improve the time complexity of the algorithm.

We noticed that there are many variations of the time-varying maximum flow problem which can be studied further. For example, consider the case where waiting time at a node is allowed, but should be limited within a given time limitation.

Bibliography

[1] L. R. Ford and D. R. Fulkerson(1958), Constructing Maximum dynamic flows from static flows, *Operations Research*, Vol. 6, pp.419-433.

[2] I. Halpern(1979), Generalized dynamic flows, *Networks*, vol. 9, pp.133-167.

[3] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[4] X. Cai, D. Sha, and C.K. Wong, *Time-varying Network Optimization*, Springer Verlag, 2007.

[5] Charu C. Aggarwal C. C. Aggarwal and J. B. Orlin (2002), On multiroute maximum flows in network,*Networks*, Vol. 39, pp.43-52.

[6] G. Mazzoni, S. Pallottino and M. G. Scutella (1991), The maximum flow problem: A max-preflow approach,*European Journal of Operational Research*, Vol. 53, pp.257-278.

[7] A. V. Goldberg(1998), Recent developments in maximum flow algorithms, *Lecture Notes In Computer Science*, Vol. 1432, pp.1-10.

[8] G. Xue, S. Sun and J. B. Rosen(1997), Fast data transmission and maximal dynamic flow, *Information Processing Letters*, Vol. 66, pp.127-132.

[9] R. K. Ahuja and J. B. Orlin (1991), Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems, *Naval Research Logistics*, Vol. 38, pp.413-430.

[10] J. J. Jarvis and H. D. Ratliff(1982), Some equivalent objectives for dynamic network flow problems,*Managment Science*, Vol.28, pp.106-109.

[11] R. K. Ahuja and J. B. Orlin(1989), A fast and simple algorithm for the maximum flow problem,*Operational Research*, Vol. 37, pp.748-759.

[12] Ahuja,R.K.,Magnanti,T.L. and Orlin, J.B. (1991), Some recent advances in network flows, *SIAM Review*, Vol.33, pp.175-219.

[13] Anderson, E.J., Nash, P. and Philpott, A.B.(1982), A class of continuous network flow problems,*Mathematics of Operations Research*, Vol. 7, No.4, pp.501-514.

[14] Anderson, E.J. and Philpott , A.B.(1984), Duality and algorithm for a class of continuous transportation problems,*Mathematics of Operations Research*, Vol.9, No.2, pp.222-231.

[15] Anderson, E.J. and Philpott , A.B.(1994), Optimization of flows in networks over times, Probability, *Statistics and Optimizations* (Edited by F.P.Kelly), John Wiley and Sons Ltd.

[16] Aronson, J.E.(1989), A survey of dynamic network flows,*Annals of Operations Research*, Vol.20, No. 1/4, pp.1-66.

[17] Cai, X., D. Sha and C.K. Wong (1998), The time-varying maximum capacity path problem with no waiting time. In J.F. Gu, G.H. Fan, S.Y. Wang, and B. Wei (eds): Advances in Operations Research and Systems Engineering.*Global-link Publishing Co.*, 9-16.

[18] Cai, X., Sha D. and Wong, C.K.(2001b), Time-varying universal maximum flow problems, *Mathematical and Computer Modelling 33* pp.407-430.

[19] Abdallaout,G. (1987), Maintainability of a grade structure as a transportation problem,*Journal of the Operational Research Society 38*, pp.367-369.

[20] Ahlfeld, D.P., R.S. Dembo, J.M. Mulvey and S.A.Zenios(1987), Nonlinear programming on generalized networks,*ACM Transactions on Mathematical Software 13*, pp.350-367.

[21] Aho, A.V., J. E. Hopcroft and J. D. Ullman(1983), Data Structures and Algorithms, Addison-Wesley,*Reading*, MA.

[22] Ahuja,R.K. and Orlin, J.B.(1992a), The scaling network simplex algorithm,*Operations Research 40*, Supplement 1, S5-S13.

[23] Ahuja, R.K., M. Kodialam, A.K. Mishra and J.B.Orlin(1992), Computational testing of maximum flow algorithms, Sloan Working Paper, Sloan School of Management, MIT, Cambridge, MA.

[24] Ahuja,R.K., Magnanti,T.L., Orlin, J.B. and M.R.Reddy(1992), Applications of network optimization, Sloan Working Paper, Sloan School of Management, MIT, Cambridge, MA.

[25] Ahuja,R.K., Orlin, J.B., C.Stein and R.E.Tarjan(1990), Improved algorithms for bipartite network flow problems, Technical Report, Sloan School of Management, MIT, Cambridge, MA. Submitted to SIAM Journal on Computing.

[26] Aronson, J.E. (1989), A survey of dynamic network flows,*Annals of Operations Research 20*, pp.1-66.

[27] Assad, A.A. (1978), Multicommodity network flows: A survey.*Networks 8*, pp.31-97.

[28] Anderson, W.N.(1975), Maximum matching and the rank of a matrix, *SIAM Journal on Applied Mathematics 28*, pp.114-123.

[29] Ali, A.I., R. Padman and H. Thiagarajan(1989), Dual algorithms for pure network problems,*Operations Research 37*, pp.159-171.

[30] Balinski, M.L. (1986), A competitive (dual) simplex method for the assignment problem, *Mathematical Programming 34*, pp.125-141.

[31] Bazaraa, M.S., J.J. Jarvis and H.D.Sherali(1990), Linear Programming and Network Flows, 2nd ed. *Wiley*, New York.

[32] Bellman, R.E.(1957), Dynamic Programming, Princeton University Press,*Princeton*, NJ.

[33] Cheruyan, J. and S.N. Maheshwari(1989), Analysis of preflow push algorithms for maximum network flow,*SIAM Journal on Computing 18*, pp1057-1086.

[34] Christophides, N.(1975), Graph Theory: An Algorithmic Approach, *Academic Press*, New York.

[35] Chvatal, V. (1983), Linear Programming, *W.H. Freeman* , New York.

[36] Derigs, U.(1988), Programming in Networks and Graphs, Lecture Notes in Economics and Mathematical System, Vol. 300.*Springer-Verlag*, New York.

[37] Edmonds, J. (1971), Matroids and the greedy algorithm,*Mathematical Programming 1*, pp.127-136.

[38] Gabow, H.N.(1985), Scaling algorithm for network problems,*Journal of Computer and System Sciences 31*, pp.148-168.

[39] Cheriyan, J. and T. Hagerup.(1989), A randomized maximum-flow algorithm,*Proceedings of the 30th IEEE Conference on the Foundations of Computer Science*, pp.118-123.

[40] Bixby, R.E. (1991), The simplex method: It keeps getting better, Presented at the 14th International Symposium on Mathematical Programming,*Amsterdam*, The Netherlands.

Publications

[1] Wenhua Yu, Dan Sha, An Excess Scaling Algorithm for the Time-Varying Maximum Flow Problem, Journal of Shanghai Normal University, 2008, 37(3);

[2] Wenhua Yu, Jianming Zhu, Dan Sha, Uniqueness of cycle length distribution of certain bipartite graphs $K_{n,n} - A(|A| = 6)$, Journal of Shanghai Normal University, 2007,36(6):17-21;(in chinese)

[3] Wenhua Yu, Jianming Zhu, Determination of Hamilton MCD graph in 2-connected simple graph containing a subgraph homeomorphic to K_4, have finished;(in chinese)

[4] Jianming Zhu, Wenhua Yu, Dan Sha, Uniqueness of cycle length distribution of certain bipartite graphs $K_{n,n+7} - |A|(|A| \leq 3)$. (Accepted by Journal of Mathematical Research and Exposition)

[5] Jianming Zhu, Wenhua Yu, Dan Sha, Uniqueness of cycle length distribution of certain bipartite graphs $K_{n,n+8} - |A|(|A| \leq 3)$, Journal of Shanghai Normal University (Natural Sciences), 2007, 36(2)17-23.(in chinese)