A Survey on Earliest Arrival Flows and a Study of the Series-Parallel Case

Mechthild Steiner

July 16, 2009

Acknowledgement

I am most grateful to my Lord and God Jesus Christ who is truly worthy of all praise. I especially praise and thank Him for all His help concerning this work.

I am also very grateful to Jun. Prof. Dr. Stefan Ruzika for his help and guidance and especially for his patience. I also thank Heike Sperber for sharing her knowledge of series-parallel graphs with me.

Contents

Chapter 1

Introduction

Travelling from one place to another requires time, as everybody experiences. To create a mathematical model of travelling or transportation, it is thus necessary to take this travel time into account. In mathematical optimization travelling or transportation problems are modelled by network flows.
Static network flows, which do not take into account the aspect of time, have been studied extensively in the last century. A network consists of *nodes* and *arcs*, which connect the nodes. Each node models a town, a factory, a computer, etc. and each arc corresponds to the connection between these nodes, e.g. a street, a cable. To each arc a capacity is assigned, which bounds the amount that can be transported on this arc.
Static network flow theory is the basis of dynamic network flow theory. In a dynamic network, each arc has a transit time, which models the time it takes to traverse the arc. In this thesis we consider discrete time steps. The capacity of an arc then defines the amount that may enter the arc at each time step. Some dynamic network flow problems can be viewed as a generalization of static network flow problems, e.g. the maximal dynamic flow problem. Furthermore solution approaches for static network flow problems can be transferred to dynamic network flow problems by using the time-expanded graph, which was introduced by Ford and Fulkerson [11]. The time-expanded graph is a copy of the original network for every time step until time horizon T is reached and models transit times implicitly. Thus the time-expanded network is a static network. But all algorithms working on time-expanded graphs depend on T and thus are pseudo polynomial.
Dynamic network flows are applicable to various real life problems, e.g. production planning, communication, personnel assignment, building evacuation, scheduling. For a more complete list of applications and more information about static network flows see the textbook of Ahuja et al.[2].

Taking the example of building evacuation, there are different interesting real life questions. Imagine that you receive a bomb scare for a building. You know that the bomb will explode after a certain time T and you want to evacuate as many people as possible within this given time horizon T. This is a maximal dynamic flow problem, which was first studied by Ford and Fulkerson [11], [12], who also gave a clever algorithm for solving this problem. We introduce and discuss the maximal dynamic flow problem in Chapter 3.
Another question related to the maximal dynamic flow problem is the earliest arrival flow problem: Consider the same situation as described above, but now you want to evacuate the people in such a way, that at every point of time t, $0 \leq t \leq T$, as many people as possible arrive in the secure region outside the endangered building. Existence of a solution to such problems was proved by Gale [14]. Minieka [25] and Wilkinson [28] gave independently of each other pseudo polynomial algorithms to solve the earliest arrival flow problem. Hoppe and Tardos [19] gave a polynomial approximation algorithm to find earliest arrival flows. In Chapter 4 we introduce and discuss extensively the earliest arrival flow problem.

To the best of our knowledge no polynomial time exact algorithm for the earliest arrival flow problem is known unto today. Thus, we consider maximal dynamic and earliest arrival flows on the special class of series-parallel graphs. We develop a maximal dynamic flow algorithm for series-parallel graphs that takes the special structure of these graphs into consideration. Then we show that this polynomial algorithm also solves the earliest arrival flow problem on series-parallel graphs.

Outline of this thesis: In Chapter 2 we define dynamic networks and dynamic network flows. The maximal dynamic flow problem is studied in Chapter 3, where we introduce time-expanded graphs and present the algorithm of Ford and Fulkerson [11]. The extensive study of earliest arrival flows in Chapter 4 includes a formal problem formulation, the existence proof of Gale [14], the exact pseudo polynomial algorithm of Minieka [25], the approximation algorithm of Hoppe and Tardos [19], a summary of the research on flow-dependent earliest arrival flows and an outline on further literature. In Chapter 5 we examine maximal dynamic and earliest arrival flows on series-parallel graphs and give a polynomial algorithm for both. Eventually in Chapter 6 we conclude with summary, evaluation and open problems.

Chapter 2

Preliminaries

In this thesis we consider a directed graph $G = (N, A)$, where N is the set of all nodes of the graph and A is the set of all arcs. We have one source node s and one sink node z and to each arc (i, j) a certain capacity $c_{ij} \in \mathbb{Z}$ with $0 \leq c_{ij} < \infty$ is assigned. The classical maximal flow problem on such a static network is given as follows, where x_{ij} denotes the flow from node i to node j and v is the total flow value induced by this flow x:

$$\begin{aligned}
\text{maximize } & v \\
\text{subject to } & \sum_{i \in N} (x_{ij} - x_{ji}) = 0 && \forall j \in N \\
& \sum_{i \in N \setminus \{s\}} x_{si} = v \\
& \sum_{i \in N \setminus \{z\}} x_{iz} = v \\
& 0 \leq x_{ij} \leq c_{ij} && \forall i, j \in N
\end{aligned} \tag{2.1}$$

As mentioned in the introduction, static flows do not take into account that it takes time to traverse an arc. To model this important property, we assign to each arc (i, j) a traversal time $\tau_{ij} \in \mathbb{N}$, which corresponds to the time it takes one unit of flow to travel from node i to node j on arc (i, j). Furthermore we assume to have a finite amount of time T for the flow to travel through the network. In this thesis we consider discrete time steps, which is an approximation to the continuous time of real world problems. The smaller the discrete time intervals are, the better the approximation. So the accuracy of discrete time models is a trade-off to the computation time, as it takes more time to consider more time intervals. In the discrete-time dynamic

setting the capacity c_{ij} on each arc represents the maximal amount of flow that may enter arc (i, j) per time period.
We have the following definition:

Definition 1 *A discrete-time dynamic network $G = (N, A, T)$ is a directed network $G = (N, A)$ with capacities c_{ij} and transit times τ_{ij} on each arc (i, j) and a given finite integer T, where the time horizon from 0 to T is discretized into time intervals.*

A feasible dynamic flow on a dynamic network has to fulfil some constraints. We have a capacity constraint which ensures that on every arc $(i, j) \in A$ at most c_{ij} units of flow enter at every time step:

$$0 \leq x_{ij}(t) \leq c_{ij}, \quad \forall i, j \in N, t \in \{0, .., T\}, \tag{2.2}$$

where $x_{ij}(t)$ denotes the amount of flow that enters arc (i, j) at time t. Furthermore we have dynamic flow conservation constraints for each node $i \in N$, which imply that at every point of time t the amount of flow that reaches a node i is the same as the amount of flow that leaves this node at time t. For every intermediate node $i \in N \setminus \{s, z\}$ the flow conservation constraint is defined as follows:

$$\sum_{j \in N} (x_{ji}(t - \tau_{ji}) - x_{ij}(t)) = 0, \quad \forall i \in N \setminus \{s, z\}, t \in \{0, .., T\}. \tag{2.3}$$

Note that flow that reaches node i at time t, must have left node j at time $t - \tau_{ji}$, for the flow needs time τ_{ji} to traverse arc (j, i).
For the source and the sink nodes we want that all the flow that leaves the source arrives at the sink within time horizon T and thus we get the following dynamic flow conservation constrains for these nodes:

$$\sum_{i \in N} \sum_{t=0}^{T} (x_{si}(t) - x_{is}(t - \tau_{is})) = - \sum_{i \in N} \sum_{t=0}^{T} (x_{zi}(t) - x_{iz}(t - \tau_{iz})) \tag{2.4}$$

Note that hold-overs, i.e. storage of flow in a node, are modelled for all nodes $i \in N \setminus \{s, z\}$ implicitly by (2.3). Of course we can also allow hold-over at the source and sink node, resp., where we would just have to add a loop starting in s and ending in s (resp. z) with capacity ∞.

In the following we present two important dynamic network flow problems: The maximal dynamic flow problem and the earliest arrival flow problem.

Chapter 3

Maximal dynamic flows

3.1 Problem Formulation

First we consider maximal dynamic flows, which were introduced by Ford and Fulkerson [11],[12] in 1958. The maximum dynamic flow problem asks the following question: Find a feasible dynamic flow on $G = (N, A, T)$ that sends the maximal amount of flow from source s to sink z in time T.

Example 1 *Consider the dynamic graph given in Figure 3.1 to get a better understanding of the maximal dynamic flow problem. The maximal dynamic flow problem is for example: What is the maximal amount of flow that can travel from source node s to sink node z within time horizon $T = 6$?*

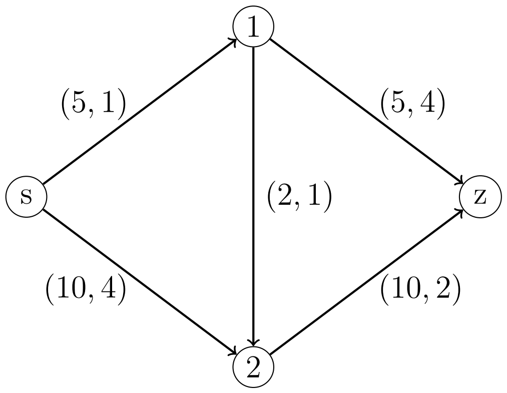


Figure 3.1: Dynamic network with capacities c_{ij} and transit times τ_{ij} on the arcs.

We can formulate the maximal dynamic flow problem as a linear program:

$$\begin{aligned}
&\text{maximize } v \\
&\text{subject to } \sum_{i\in N}\sum_{t=0}^{T}(x_{si}(t) - x_{is}(t-\tau_{is})) = v \\
&\sum_{i\in N}\sum_{t=0}^{T}(-x_{zi}(t) + x_{iz}(t-\tau_{iz})) = v \\
&\sum_{j\in N}(x_{ji}(t-\tau_{ji}) - x_{ij}(t)) = 0 \qquad \forall i \in N\setminus\{s,z\}, t\in\{0,..,T\} \\
&0 \le x_{ij}(t) \le c_{ij} \qquad \forall i,j\in N, t\in\{0,..,T\}
\end{aligned}$$

where x_{ij} only appears if $(i,j)\in A$ and $\tau_{ij}\geq 0$. The constraints are essentially those we introduced in Chapter 2 and ensure the feasibility of the maximal dynamic flow.
Ford and Fulkerson [11] gave two important solution strategies for this problem: time-expanded graphs and a clever algorithm to solve single source single sink dynamic network flow problems.

3.2 Time-Expanded Graph

Ford and Fulkerson [11] introduced a *time-expanded network* $D(T)$ which is a static network obtained by an expansion of the dynamic network. A dynamic flow through the dynamic network $G = (N, A, T)$ corresponds to a static flow in the time-expanded network $D(T)$. Time-expanded networks are a mighty tool to solve dynamic network flow problems, for they allow to apply the solution methods for static network flow problems to dynamic network flow problems.
Such a time-expanded network $D(T)$ may be constructed as follows: Copy each node i of the graph T times, so that the nodes in $D(T)$ are of the form $i(t)\ \forall i\in N, t\in\{0,..,T\}$. Also copy the original arcs between the nodes, that is, $D(T)$ has arcs $(i(t), j(t+\tau_{ij}))$, where $0\le t\le T-\tau_{ij}$, with capacity c_{ij}. The hold-over property, which allows storage at the nodes, can be modelled by adding arcs $(i(t), i(t+1)), 0\le t<T$ and capacity ∞ for each node.

Example 2 *In Figure 3.2 the time-expanded graph of Figure 3.1 is shown. The capacity of each arc is given in Figure 3.2 and corresponds to the capacity of the original arc of the underlying dynamic network. Note that the transit*

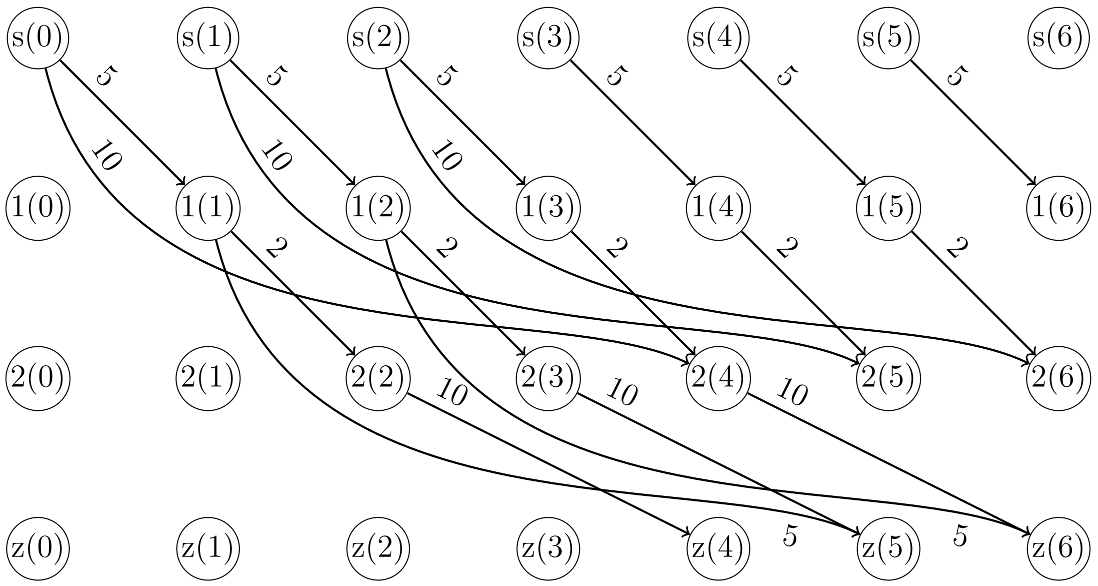


Figure 3.2: Time-expanded graph of the dynamic network shown in Figure 3.1 with time-horizon $T = 6$.

times are modelled implicitly, e.g. $\tau_{s1} = 1$ *is expressed in the time-expanded graph by the form of that arc always pointing from* $s(t)$ *to* $1(t+1)$*. Due to clarity, we have left out the hold over arcs, which would just point from* $i(t)$ *to* $i(t+1)$ *for each node* $i \in \{s, 1, 2, z\}$ *and each* $t \in \{0, .., 6\}$*.*

As stated above we can find a maximal flow on this time-expanded graph in Figure 3.2 by applying the usual static network flow theory. One such maximal flow is presented in Figure 3.3: The total flow arriving at sink z *within time* $T = 6$ *is* $\sum_{t=0}^{6} v(z(t)) = 20$*.* 5 *units of flow arrive at node* $z(5)$ *and* 15 *units of flow arrive at node* $z(6)$*. This corresponds to* 5 *units arriving at the sink at time* 5 *and* 15 *units arriving at the sink at time* 6*. Before time* 5 *no flow arrives at the sink.*

The main problem with time-expanded graphs is their size, which is dependent on the time horizon T. Thus any algorithm that works on the time-expanded graph can only be a pseudo-polynomial algorithm.

3.3 Algorithm of Ford and Fulkerson

Considering the drawbacks of the time-expanded graph, Ford and Fulkerson [11] work on the underlying static network $G = (N, A)$ to develop a polynomial algorithm.

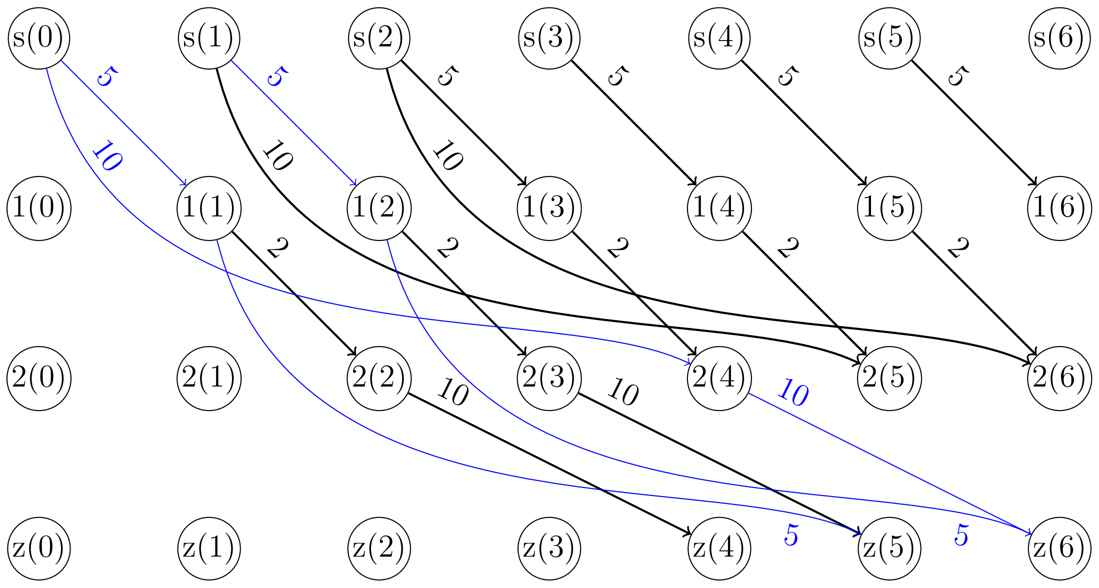


Figure 3.3: A maximal flow on the time-expanded graph of the dynamic network shown in Figure 3.1

To get a static network they interpret the transit times τ_{ij} as costs a_{ij} of each arc. Then they solve a minimum cost flow problem on the static network $G = (N, A)$. The minimum cost flow problem $P(v)$ answers the question: "What is the minimum cost of sending a flow with given flow value v from s to z in G?" Problem $P(v)$ is defined by:

$$\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in A} a_{ij} \cdot x_{ij} & \\
\text{subject to} \quad & \sum_{j\in N} x_{sj} - v = 0 & \\
& \sum_{j\in N} x_{jz} - v = 0 & \\
& \sum_{j\in N} (x_{ji} - x_{ij}) = 0 & ,\forall i \in N \\
& 0 \leq x_{ij} \leq c_{ij} & ,\forall i,j \in N
\end{aligned} \tag{3.1}$$

A feasible solution x of this problem fulfils all constraints. We denote the maximal integer value v for which $P(v)$ has a feasible solution by v_{max}.
In the maximal dynamic flow problem we want to maximize the flow, so that the costs of this flow are minimal and that all flow arrives at the sink within time T. Thus we need to model the time horizon T in the minimum

cost flow problem to be allowed to transfer the results of this problem to the maximum dynamic flow problem. Ford and Fulkerson presented the primal-dual Algorithm 1 to solve the minimum cost flow problem with respect to the time horizon.

In the next step Ford and Fulkerson decompose with Algorithm 2 the flow they found with Algorithm 1 into chain flows. To get a maximal dynamic flow, these chain flows have to be repeated over time as often as possible, i.e. each chain flow starts at zero and is repeated (at every time interval) as long as there is enough time for the flow to reach the sink within time horizon T.

Definition 2 *A (standard) chain decomposition $\mathbb{P}_s$ of a flow x is a set of paths $P_1, P_2, \dots, P_q$ with associated flows $(\gamma_1, \gamma_2, \dots, \gamma_q)$ and flow values $v(\gamma_1), v(\gamma_2), \dots, v(\gamma_q)$. The sum of these flows equals x.*

Definition 3 *A* temporally-repeated flow (TRF) *is a dynamic flow on $G = (N, A, T)$ that can be generated by repeating all chain flows $\gamma_1, \dots, \gamma_q$ of a chain decomposition $\mathbb{P}_s$ of a flow x. All chain flows in the chain decomposition use arcs in the direction of the flow. Every chain flow γ_m starts at $t = 0$ and is repeated $T + 1 - \tau(\gamma_m)$ times, where $\tau(\gamma_m)$ denotes the total travel time of chain flow γ_m. Thus the flow value $v(\mathbb{P}_s)$ of the temporally repeated flow is defined by:*

$$v(\mathbb{P}_s) := \sum_{\gamma_m \in \mathbb{P}_s} v(\gamma_m)(T + 1 - \tau(\gamma_m))$$

Example 3 *Now we apply the algorithm of Ford and Fulkerson to our example 1: Starting the procedure as stated in Algorithm 1 with $\pi_i = 0$, $i \in N \backslash \{z\}$ and $\pi_z = 1$, we see that no flow can travel from s to z, that is for no node $j \in \{1, 2, z\}$ we have $\pi_s + a_{sj} = \pi_j$ and the other case is not possible since no arcs are pointing at s. So we are in Step 4 and increase the node numbers π_1, π_2, π_z. Now flow can reach node 1, i.e. we have $\pi_s + a_{s1} = 0 + 1 = 1 = \pi_1$, but not yet node 2 or z. Therefore we jump to Step 4 immediately and increase π_2 and π_z. With these updated node numbers $\pi_s = 0$, $\pi_1 = 1$, $\pi_2 = 2$ and $\pi_z = 3$ nodes s, 1 and 2 can be labeled, but not yet node z, since $\pi_1 + a_{1z} = 1 + 4 \neq 3 = \pi_z$ and $\pi_2 + a_{2z} = 2 + 2 = 4 \neq 3 = \pi_z$. Again we are in Step 4 and increase π_z. Via path $s - 1 - 2 - z$ the sink node z is reached for the first time. Now we assign in Step 3 to this path flow value 2, since $c_{12} = 2$ is the smallest capacity on this path and of course all nodes are labeled with $+$, since $x_{ij} = 0$, $\forall (i, j) \in A$ at the beginning. This yields*

Algorithm 1 Algorithm of Ford and Fulkerson, Routine 1

```
Require: dynamic network $G = (N, A, T)$
  $N =: N_u :=$ set of all unlabeled nodes
  $\emptyset =: N_l :=$ set of all labeled but unscanned nodes
  $\emptyset =: N_s :=$ set of all labeled and scanned nodes
  dual parameters $\pi$ with $\pi_i := 0 \; \forall i \in N \setminus \{z\}$ and $\pi_z = 1$, flow $x$ with
  $x_{ij} = 0 \; \forall (i,j) \in A$, arc costs or transit times $a_{ij}$
  while $\pi_z \leq T + 1$ do
    Step 1: assign label $[n^+, \infty]$ to source $s$, update $N_l := N_l \cup \{s\}$, $N_u =$
    $N_u \setminus \{s\}$
    Step 2:
    while $N_l \neq \emptyset$ or $z \notin N_l$ do
      take node $i \in N_l$ with label $[k^\pm, h]$
      for all $j \in N_u$ such that $\pi_i + a_{ij} = \pi_j$ do
        if $x_{ij} < c_{ij}$ then
          assign label $[i^+, \min(h, c_{ij} - x_{ij})]$ to $j$,
          update $N_l := N_l \cup \{j\}, N_u := N_u \setminus \{j\}$
        end if
      end for
      for all $j \in N_u$ such that $\pi_j + a_{ji} = \pi_i$ do
        if $x_{ji} > 0$ then
          assign label $[i^-, \min(h, x_{ji})]$ to $j$, update $N_l := N_l \cup \{j\}$, $N_u :=$
          $N_u \setminus \{j\}$
        end if
      end for
      consider $i$ as scanned, i.e. $N_l := N_l \setminus \{i\}$, $N_s := N_s \cup \{i\}$
    end while
    Step 3:
    if $z \in N_l$ then
      if $z$ has label $[k^+, h]$ then
      $x_{kz} := x_{kz} + h$
      Otherwise if $z$ has label $[k^-, h]$ then
      $x_{kz} := x_{kz} - h$
      end if
      then go to $k$ and update the flow values accordingly
      do this for all predecessors found in this way until you reach node $s$
      set $N_u := N$, $N_l := \emptyset$, $N_s := \emptyset$ and GOTO Step 1
    else
      Step 4:
      define $\pi_i^{'}$ by: $\pi_i = \pi_i$, $if\ i \in N_l$ and $\pi_i = \pi_i + 1$, $if\ i \in N_u$.
      set $\pi_i := \pi_i^{'}$, $N_u := N$, $N_l := \emptyset$, $N_s := \emptyset$
      GOTO Step 1
    end if
  end while
```

Algorithm 2 Algorithm of Ford and Fulkerson, Routine 2

```
Require: a dynamic network $G = (N, A, T)$ and flow $x$ found by routine 1
  set $N_u := N,\ N_l := \emptyset,\ N_s := \emptyset$, set $m := 1$
  Step 1:
  assign label $[n, \infty]$ to $s$, set $N_u := N_u \setminus \{s\},\ N_l := N_l \cup \{s\}$
  Step 2:
  while $N_l \neq \emptyset$ and $z \notin N_l$ do
    take $i \in N_l$ with label $[k, h]$
    for all $j \in N_u$ do
      if $x_{ij} > 0$ then
        assign label $[i, \min(h, x_{ij})]$ to $j$ and set $N_u := N_u \setminus \{j\},\ N_l :=$
        $N_l \cup \{j\}$
      end if
    end for
    set $N_l := N_l \setminus \{i\},\ N_s := N_s \cup \{i\}$
  end while
  Step 3:
  if $z \in N_l$ and labeled $[k, h]$ then
    $x_{kz} := x_{kz} - h$
    add node $k$ to path $P_m$
    repeat for all predecessors $k$ found in this way until $k = s$
    in each step add the new node to the path $P_m$
    assign to path $P_m = \{s, \dots, z\}$ the flow value $h = v(P_m)$
    set $N_u := N,\ N_l := \emptyset,\ N_s := \emptyset$ and $m := m + 1$
    GOTO Step 1
    Step 4:
  else
    all $x_{ij}$ are zero and we have found all paths, so STOP Routine 2
  end if
```

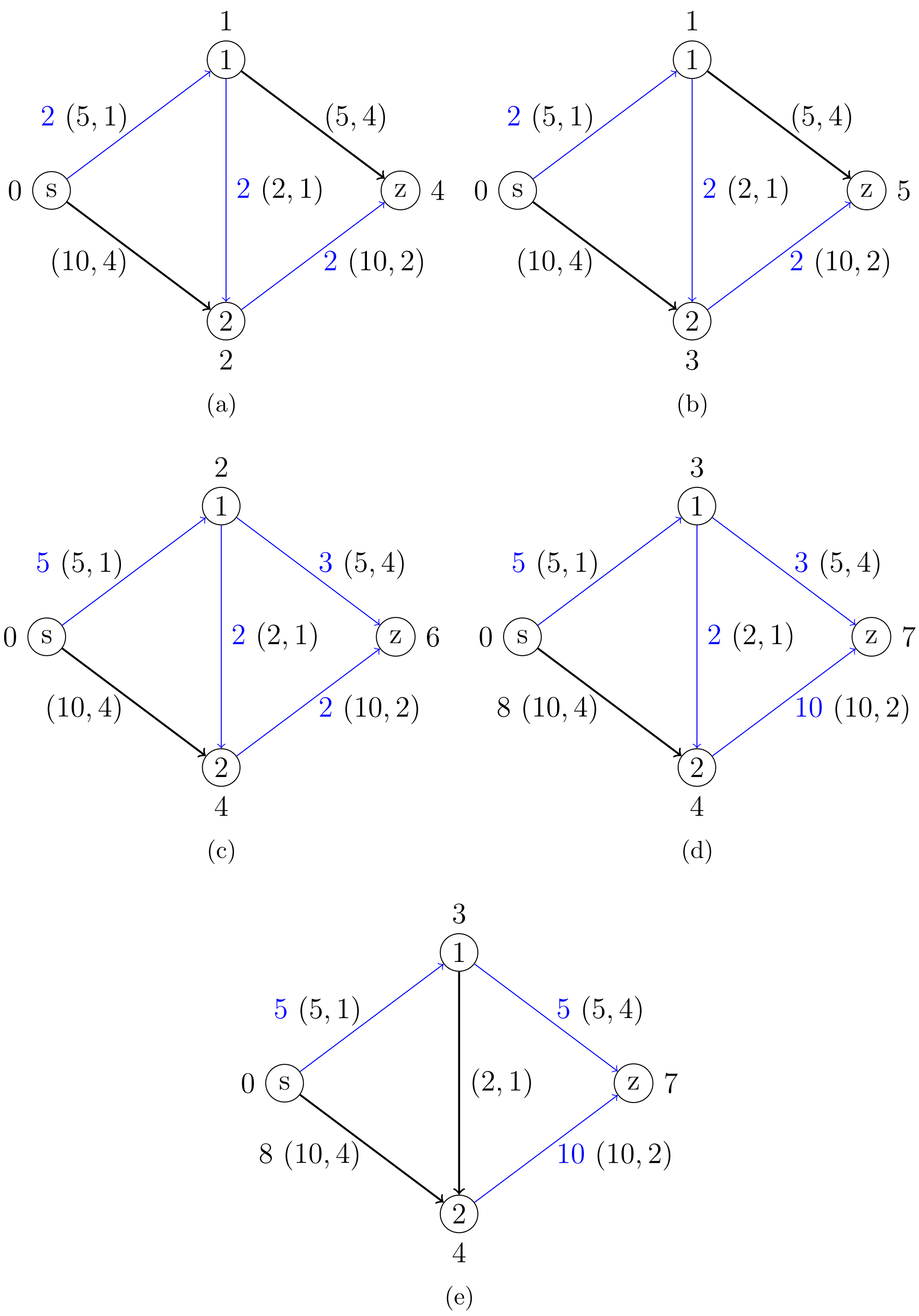


Figure 3.4: the node numbers are given at the nodes and the flow values on the blue arcs.

the graph shown in Figure 3.4a. Then we remove all labels, preserve the flow and return to Step 1.
In Step 1 we can again label node 1*, but we cannot reach node* 2 *any longer since* $c_{12} = x_{12} = 2$*. So we go to Step 4 and* π_2 *and* π_z *are increased which is shown in Figure 3.4b. Again we can label node* 1*, this time with label* $[s^+, 3]$*, where* $3 = \min(\infty, 5 - 2)$*. We cannot reach node* 2*, for on arc* $(1, 2)$ *we have already* $x_{12} = c_{12} = 2$ *and* $\pi_s + a_{s2} = 0 + 4 \neq 3 = \pi_2$*. But we can reach node* z *via arc* $(1, z)$*, since* $\pi_1 + a_{1z} = 1 + 4 = 5 = \pi_z$*. We get a new flow augmenting path* $s - 1 - z$ *and the update of the flow values in Step 3 yields* $x_{s1} = 2 + 3 = 5$ *and* $x_{1z} = 3$*. Then we go to Step 1 and find that we can label no node except node* s*, so all the other node numbers have to be increased and we get the flow shown in Figure 3.4c.*
Going through Step 1 we label node 2*, since* $\pi_s + a_{s2} = 0 + 4 = 4 = \pi_2$*. With* $i = 2$ *we can only reach node* z*, where we have to assign label* $[2^+, 8]$*, since* $8 = \min(10, 10 - 2)$*. The update of the flow in Step 3 yields the following changes:* $x_{s2} = 8$ *and* $x_{2z} = 10$*. Repeating Step 1 with this updated flow, labels nodes* s *and* 2 *but cannot label neither node* 1 *nor node* z*, so we go to Step 4 and increase* π_1*,* π_z *and get the graph shown in Figure 3.4d.*
Starting Step 1 with these node numbers and this flow, we can label node 2 *with* $[s^+, 2]$*. Now with* $i = 2$*, we assign to node* 1 *label* $[2^-, 2]$ *since* $\pi_1 + a_{12} = 3 + 1 = 4 = \pi_2$ *and* $x_{12} = 2 > 0$*. Eventually we assign to node* z *label* $[1^+, 2]$*, since we can reach node* z *via arc* $(1, z)$*, i.e.* $\pi_1 + a_{1z} = 3 + 4 = 7 = \pi_z$*. Updating the flows in Step 3 yields the minimal cost static flow shown in Figure 3.4e. It is not possible to find another flow on this graph, thus we have to increase the node numbers and stop, since the increase yields* $\pi_z = 8 > 6 + 1 = T + 1$*.*

Now we can apply Algorithm 2 to decompose this minimal cost flow into chain flows. Algorithm 2 starts with the flow found in Algorithm 1. A forward-backwards structure is used to find the chain flows: In the forward procedure (Step 1 and Step 2) a path from s *to* z *with positive flow, i.e.* $x_{ij} > 0$ *is determined. In the backwards step (Step 3), we start at* z *and pursue backward the path found in the forwards procedure where each node we encounter this way is added to* P_m *which gives the nodes of the path of the chain flow. On this path we decrease the flow as much as possible, i.e. at least on one arc of the path the flow value will be decreased to 0.*
We detect two chain flows in our example: $\gamma_1 = \{z, 1, s\}$ *with flow value 5 and* $\gamma_2 = \{z, 2, s\}$ *with flow value 10.*
To get a maximal dynamic flow we have to repeat these chain flows as often

as possible. Chain flow $s - 1 - z$ needs five time units to reach the sink, so we can start it at time 0 and at time 1. On γ_2 the flow needs 6 time units to reach the sink, so we can send it only once. Drawing these results in the time-expanded graph we get the time-expanded graph shown in Figure 3.3. Notice that the algorithm works on the underlying static network and not on the time-expanded network.

Note, that the number of chain flows found by Algorithm 2 is bounded by the number of arcs of the static network, since in every iteration the flow value on at least one arc is decreased to zero.
The flow generated by the algorithm of Ford and Fulkerson is not necessarily a maximal flow on the static network. It is not obvious that the algorithm of Ford and Fulkerson always yields a maximal dynamic flow, in other words, it is not obvious that there always exists a maximal dynamic flow within the subclass of temporally-repeated flows. To prove this important result Ford and Fulkerson argue on the time-expanded graph and show that the flow generated through the chain flows and the cuts defined by the algorithm are equal. Thus, due to the max-flow-min-cut theorem, the flow must be maximal and the cut must be minimal.

3.4 Minimum Cost Circulation Problem

As we have seen in the previous section the Ford and Fulkerson maximal dynamic flow algorithm is a primal-dual schema. We have also seen that there is a close relation between the maximal dynamic flow problem and the minimal cost flow problem. In this section we first present the minimal cost flow algorithm of Klein [21] and then modify it, so that it solves the maximal dynamic flow problem.
The minimum cost flow algorithm of Klein [21] is a primal method which works on the residual network $G'(x)$ of the original network G with associated flow x.

Definition 4 *The residual network $G'(x) = (N, A'(x))$ of the original network $G = (N, A)$ with flow x is defined by :*
arc set $A'(x) := A^+(x) \cup A^-(x)$, where

$$\begin{aligned} A^+(x) &:= \{(i,j) \in A \text{ with } x_{ij} < c_{ij}\} \\ A^-(x) &:= \{(j,i) : (i,j) \in A \text{ with } x_{ij} > 0\} . \end{aligned}$$

To each arc in $A'(x)$ we assign the following capacities

$$\begin{array}{lll} c'_{ij} := & c_{ij} - x_{ij} & \text{if } (i,j) \in A^+(x) \\ c'_{ji} := & x_{ji} & \text{if } (j,i) \in A^-(x) \end{array}$$

and arc costs

$$\begin{array}{ll} a'_{ij} := a_{ij} & \text{if } (i,j) \in A^+(x) \\ a'_{ji} := -a_{ji} & \text{if } (j,i) \in A^-(x). \end{array}$$

We can use the following theorem to check whether a flow x is a minimal cost flow or not:

Theorem 1 (Busacker and Saaty [6]) *The flow x is a minimal cost flow if and only if there is no directed cycle C in the residual network $G'(x)$ such that the sum of the costs around the arcs of C is negative.*

In his algorithm Klein uses this theorem to transform a maximal static flow x on network $G = (N, A)$ into a minimal cost flow:

Algorithm 3 Minimal Cost Flow Algorithm of Klein

```
Input: Static network $G = (N, A)$ with capacities $c_{ij}$ and arc costs $a_{ij}$ on each arc $(i,j) \in A$.
Output: minimal cost flow on $G$
1. Find a maximal flow $x$ of flow value $v$ on $G$ by using any appropriate algorithm, e.g. Labeling Algorithm (s.f. [17], page 152)
2. Form the residual network $G'(x)$.
3. Test for negative dicycles.
4. If a negative dicycle $C$ is found, increase the flow around $C$ as much as possible without violating the capacity of any arc $(i,j) \in A(x)$ in the cycle:
   $x_{ij} := \begin{cases} x_{ij} & \text{if } (i,j) \notin C \\ x_{ij} - \delta & \text{if } (i,j) \in C \text{ and } a'_{ij} \leq 0 \\ x_{ij} + \delta & \text{if } (i,j) \in C \text{ and } a'_{ij} > 0 \end{cases}$
   where $\delta := \min \{ c'_{ij} : (i,j) \in C \}$.
   Go to Step 2.
5. If no negative dicycle is found
   STOP: $x$ is a minimal cost flow.
```

In Algorithm 3 we need to determine negative dicycles in Step 3. This can be done, for example by using the matrix multiplication method presented in [1] as Klein [21] proposes.
As mentioned above we need to alter Algorithm 3 to get a temporally repeated flow which solves the maximal dynamic flow problem: This can be done by adding arc (z, s) to the original graph $G = (N, A, T)$ and assigning to this arc capacity $c_{zs} = \infty$ and transit time $\tau_{zs} = -(T + 1)$. Now apply Klein's algorithm where the starting flow is the zero flow. Since arc (z, s) has negative costs, there might be negative dicycles. In the first iteration this negative cycle uses the arcs of the original graph G from s to z and then arc (z, s) to complete the cycle. The arcs on this dicycle, except arc (z, s), form a path from s to z in G and have positive costs. Only if the sum of the costs of these arcs are less than or equal to T, the cycle is a negative cycle and Algorithm 3 sends flow along this dicycle. Translated into the dynamic setting this means, that only if the total transit time of a s-z-path is less than or equal to T flow is sent via this path. This is exactly what we need for the maximal dynamic flow problem. We get Algorithm 4 to solve the maximal dynamic flow problem.

Algorithm 4 Minimal Cost Circulation Flow Algorithm

```
Input: dynamic network $G = (N, A, T)$ with given capacities and transit
times and an additional arc $(z, s)$ with capacity $c_{zs} = \infty$ and transit time
$\tau_{zs} = -(T + 1)$
Output: A maximal dynamic flow
Set the flow $x_{ij} := 0$ for all arcs $(i, j) \in A$
Interpret the transit time on each arc as costs and apply Algorithm 3 to
find a minimal cost flow $x^*$.
Decompose $x^*$ into path flows using Algorithm 2
```

Note that it might happen that Algorithm 4 uses arc (z, s) in the reverse direction during the execution and that the minimal cost flow it finds might not be a maximal flow on the static graph $G \setminus \{(z, s)\}$. Example 4 illustrates these cases:

Example 4 *Consider the graph given in Figure 3.5: We apply the Minimum Cost Circulation Flow Algorithm given above. Let the first dicycle we find, be* $C^1 = (s, 1, 2, z, s)$ *with costs* $a(C^1) = -1$ *and flow* x *with flow value* $v(C^1) = 5$. *Now consider the residual network* $G'(x)$ *and look for negative dicycles. We find* $C^2 = (s, 3, 4, z, s)$ *with costs* $a(C^2) = -3$ *and flow value* $v(C^2) = 10$. *We update flow* x *by adding the flow on* C^2 *and again consider the residual*

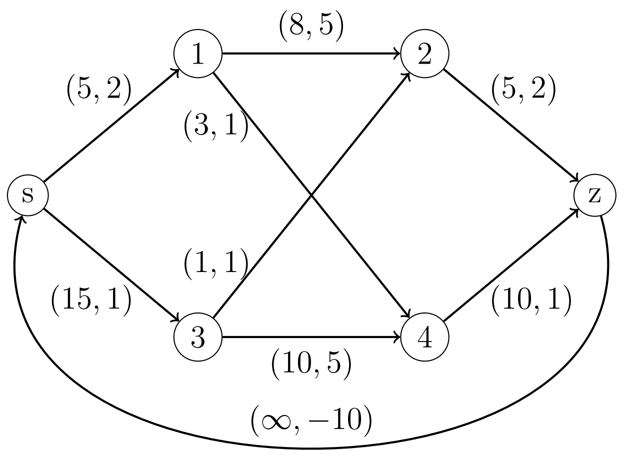


Figure 3.5: Dynamic network with capacities and transit times as given on the arcs.

network $G'(x)$, which is given in Figure 3.6. In Figure 3.6 we have marked a negative dicycle $C^3 = (s, z, 2, 1, 4, 3, s)$ which uses the backwards arc of (z, s) and takes flow back. The costs of this dicycle are $a(C^3) = -2$ and the flow value is $v(C^3) = 3$. We have to update the flow x again and look for negative dicycles in the corresponding residual network $G'(x)$. We find negative dicycle $C^4 = (s, 3, 2, z, s)$ with flow value $v(C^4) = 1$ and costs $a(C^4) = -6$. Update the residual network once more and look for negative dicycles. As there are no more negative dicycles in $G'(x)$ the algorithm stops. We apply Algorithm 2 and find the maximal dynamic flow on G with time horizon $T = 9$. This maximal dynamic flow is not a maximal flow on the static network, since it sends only 13 units of flow from s to z in the static network.

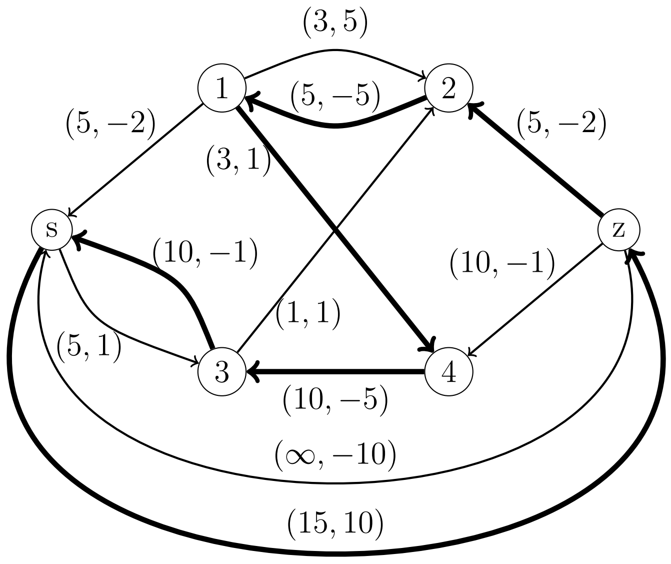


Figure 3.6: Residual network of the graph given in Figure 3.5 after the second negative circulation was found. The third negative circulation is marked.

Chapter 4

Earliest Arrival Flows

In this chapter we study earliest arrival flows, which specify the maximal dynamic flow problem by asking the flow to deliver the maximum amount of flow into the sink up to every point of time and not only at the end at time T. We start with giving the problem formulation of the earliest arrival flow problem and make some observations. Next we answer the question of existence of earliest arrival flows by following the results of Gale [14]. An algorithm for computing earliest arrival flows was given independently by Minieka [25] and Wilkinson [28] in the early 70s. We follow the approach of Minieka and present his earliest arrival flow algorithm. The idea of the algorithm of Minieka is the same as the idea of the algorithm of Wilkinson and both are pseudo polynomial. As far as we know it is still an open question, if it is possible to find an exact polynomial algorithm for the earliest arrival flow problem. Therefore we present the polynomial-time approximation scheme for earliest arrival flows of Hoppe and Tardos [19].
Next we give a short summary about the research on flow-dependent transit times. Flow-dependent transit time means, that the transit time on each arc is dependent of the flow that enters the arc or is on the arc. We present two models for modeling flow-dependent transit times and evaluate them. Then we present the fan graphs of Köhler et al. [8] which try to transform the concept of time-expanded graphs to flow-dependent transit times and the bow graph, which is the underlying dynamic version of the fan graph. Baumann and Köhler [3] show that existence of earliest arrival flows with flow-dependent transit times is not given in general for any of the presented models and give an approximation.
Finally we give a short overview of further literature about earliest arrival flows.

4.1 Problem Formulation and Observations

As presented in the previous section Ford and Fulkerson showed how to obtain for each integer T a maximal dynamic flow x with largest possible flow value v within this time horizon T.
Shortly after their contribution, Gale [14] came up with a closely related problem called *earliest arrival flow* or *universal maximal flow problem*: Does there exist a dynamic flow in $G = (N, A, T)$ from s to z which is maximal *at all times* $0 \leq t \leq T$, for a given integer T?

Such earliest arrival flows are of interest for us, because applied to evacuation problems an earliest arrival flow ensures that at any point of time the maximal number of persons is evacuated. In any case of emergency like a fire or a terroristic attack, where it is not sure how long the building will resist before it collapses, an earliest arrival flow evacuation strategy makes sure that as many lives as possible are saved.
Due to the definition of earliest arrival flows given above, we can make the following simple observation:

Observation 1 *All earliest arrival flows are maximal dynamic flows.*

A natural question arising from this observation is if the reverse is also true, i.e. are all maximal dynamic flows also earliest arrival flows? In the following example we show that this is not the case.

Example 5 *Looking at Figure 3.1 again, we see that flow taking the path* $s-1-2-z$ *would only need four time units, so at time four the maximal amount of flow that could have arrived at the sink is* 2 *(due to the capacity on arc* $(1, 2)$*). In the maximal dynamic flow in Figure 3.3 no flow at all arrives at time four. Consequently this cannot be an earliest arrival flow. In Figure 4.1 an earliest arrival flow for our original problem is shown: In Figure 4.1 at time four the first* 2 *flow units arrive. At time five another* 5 *flow units are added, so in total at time five* $2 + 5 = 7$ *flow units have already arrived at the sink (note that this is also more than in the maximum dynamic flow presented in Figure 3.3). At time six* 13 *more units of flow enter the sink, so in total* 20 *units have reached the sink by time six. Note that this flow is not a temporally repeated flow, since the chain flow* $s-1-2-z$ *is only started at times zero and one but not at time two, though there would be enough time*

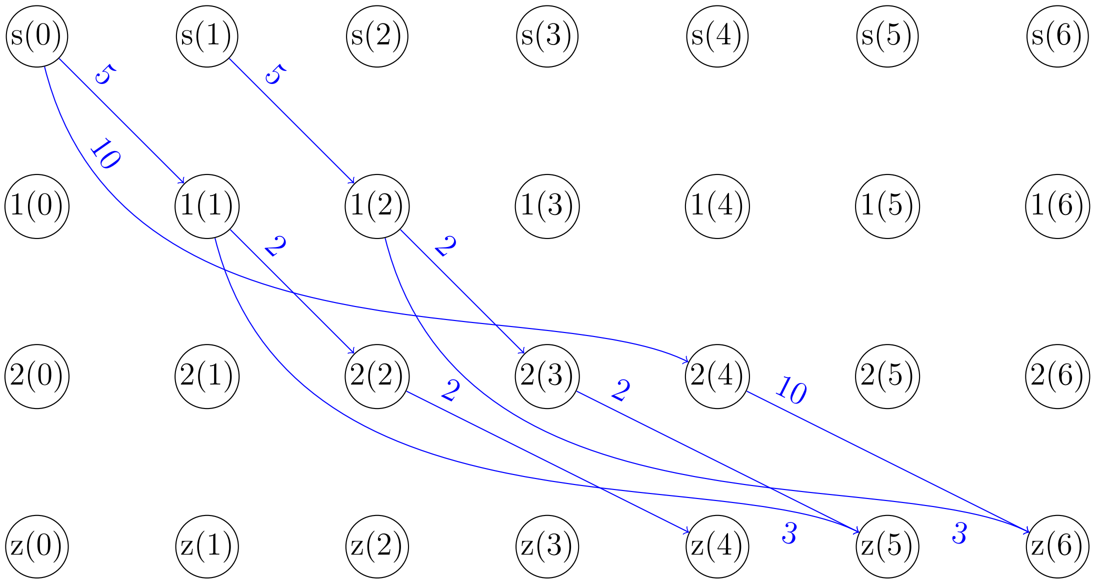


Figure 4.1: Earliest arrival flow for the dynamic graph shown in Figure 3.1. The flow values are given on the arcs.

for that chain flow to reach the sink.

We have thus derived the following observation:

Observation 2 *Maximal dynamic flows are not necessarily earliest arrival flows.*

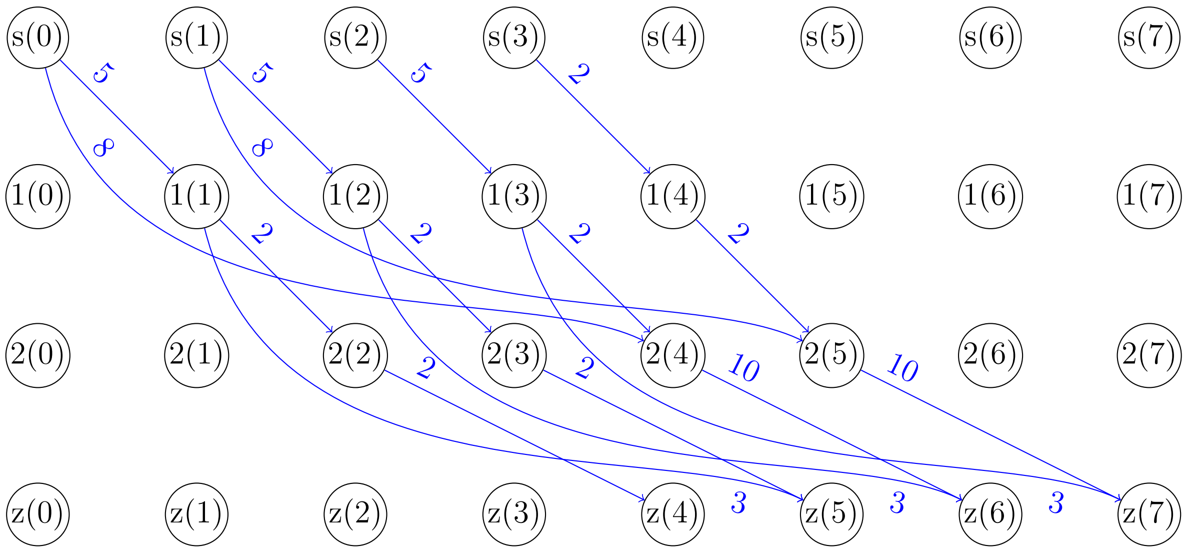
It is interesting to consider the structure of earliest arrival flows. It would be nice, if for every graph and every time horizon T, we could find an earliest arrival flow which has the property, that it is also a temporally repeated flow. Unfortunately, this is not true in general, but we show in Chapter 5 that for a special class of graphs, it is always possible to find an earliest arrival flow which has the temporally repeated flow property. For general graphs we have the following observation:

Observation 3 *On general graphs $G = (N, A, T)$ it is not always possible to find an earliest arrival flow which is a temporally repeated flow.*

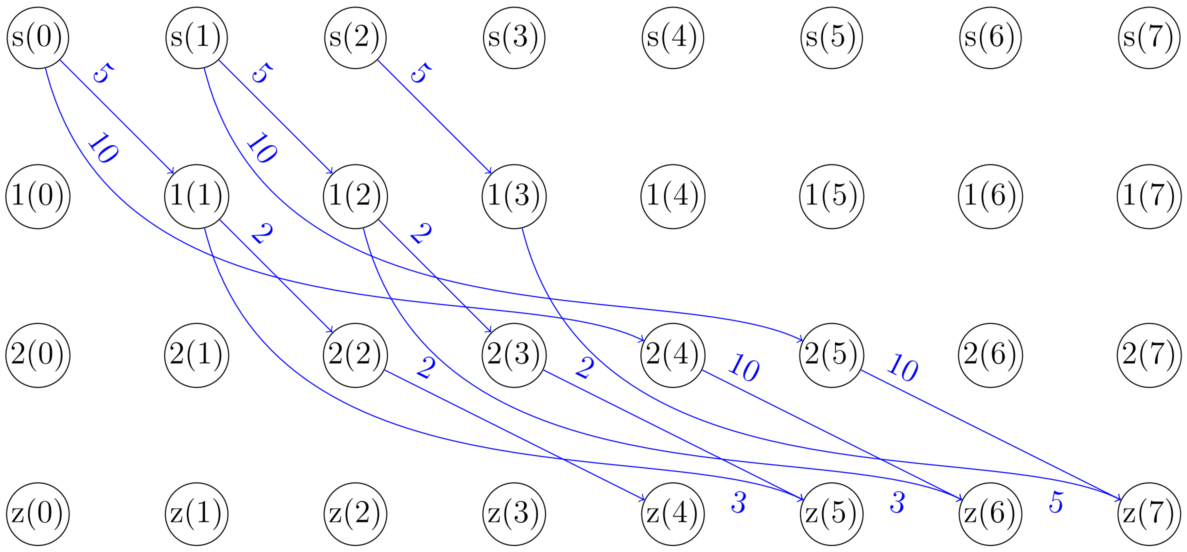
PROOF: We prove this observation by counterexample:

Assume that on every graph $G = (N, A, T)$ it is possible to find an earliest arrival flow that is a temporally repeated flow.

Now consider again the graph given in Example 1 with time horizon $T = 7$. We have already argued in Example 5, that flow must be send via path $s - 1 - 2 - z$ to get an earliest arrival flow. In order to get an earliest arrival flow with the temporally repeated flow property, we thus have to repeat this chain flow as often as possible. The flow on the time-expanded graph in Figure 4.2a shows the maximal temporally repeated flow we can get on this graph with $T = 7$, when we repeat chain flow $s-1-2-z$. In Figure 4.2a the total amount of flow that has



(a) Maximal temporally repeated flow on the graph shown in Figure 3.1 with time horizon $T = 7$.



(b) Maximal dynamic flow on the graph shown in Figure 3.1 with time horizon $T = 7$.

Figure 4.2: Counterexample proving Observation 3.

reached the sink z after 7 time units is 33. But this is not the maximal amount of flow that can reach the sink within time horizon $T = 7$, as the maximal dynamic flow given in Figure 4.2b shows. There we see that 35 flow units have reached the sink after 7 time units. Thus, due to Observation 1, the maximal temporally repeated flow in Figure 4.2a cannot be an earliest arrival flow, since it is not a maximal dynamic flow. □

4.2 Existence of Earliest Arrival Flows

The main contribution of Gale [14] is an existence proof of earliest arrival flows in the following cases:

- single source - single sink dynamic network
- generalized single source - single sink dynamic network with time dependent capacities and transit times
- multiple sources - single sink dynamic network (with time dependent capacities and transit times)

Unfortunately, as Gale pointed out, his existence theorem does not extend to the case of multiple sinks.

In the following we give the proof of existence of earliest arrival flows in the single source - single sink case according to Gale [14]. Therefore we need to introduce the concept of *demands* on nodes:

Definition 5 (Gale [14]) *A* demand *is a function $\delta : N \setminus \{s\} \rightarrow \mathbb{Z}^+$ assigning to each node $i \in N \setminus \{s\}$ a certain demand $\delta(i)$, which has to be satisfied by a flow. If there exists a flow satisfying the demand of each node, i.e.,*

$$\delta(i) \leq \sum_{j \in N} x_{ji} \quad , \forall i \in N \setminus \{s\}$$

then the demand is feasible*.*

The following feasibility theorem relates feasible demands to the capacity of cuts:

Theorem 2 (Feasibility Theorem [14]) *The demand δ is feasible if and only if, for every subset $B \subseteq N \setminus \{s\}$ δ satisfies the relation*

$$\sum_{i \in B} \delta(i) \leq \sum_{j \in N \setminus B,\ i \in B} c_{ji}. \tag{4.1}$$

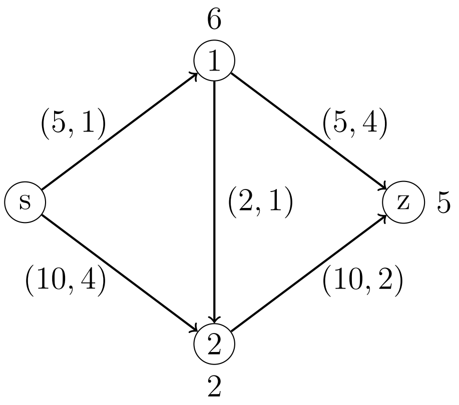


Figure 4.3: Dynamic network with demands on the nodes.

We give an illustration of this result in the following example:

Example 6 *Consider the dynamic network as given in Figure 4.3, where demands are assigned to the nodes. Then for subset* $B_1 = \{1, 2, z\}$ *we have* $\sum_{i \in B_1} \delta(i) = 6 + 2 + 5 = 13 \leq 15 = 5 + 10 = \sum_{j \in N \setminus B_1,\ i \in B_1} c_{ji}$
and for $B_2 = \{1, 2\}$ *we get* $6 + 2 = 8 \leq 15 = 5 + 10$.
For $B_3 = \{1, z\}$ *we get* $6 + 5 = 11 \leq 15 = 5 + 10$.
However, for $B_4 = \{1\}$*, we have* $\sum_{i \in B_4} \delta(i) = 6 > 5 = \sum_{j \in N \setminus B_4,\ i \in B_4} c_{ji}$.
The demand of the sets B_1*,* B_2*,* B_3 *could be satisfied, since the capacities on the arcs entering these sets are large enough. But the demand of set* $B_4 = \{1\}$ *cannot be satisfied, since the only arc entering node* 1*, i.e. arc* $(s, 1)$*, has capacity* 5 *which is less than the demand. Since the feasibility theorem states that the flow is feasible if and only if the inequality (4.1) holds for* every *subset, we can conclude that the demand in the example is not feasible.*

For the existence proof we need the following lemma which is a simple corollary of the feasibility Theorem 2.

Lemma 1 (Gale [14]) *Choose* $i_1, \ldots, i_m$ *distinct nodes of* $N \setminus \{s\}$ *and let* $\delta_1, \ldots, \delta_m$ *be feasible demands such that*

$$\delta_l(i_l) =: \mu_l \leq \delta_{l+1}(i_{l+1}) =: \mu_{l+1} \quad , l < m$$

Let δ be defined as follows:

$$\begin{aligned}\delta(i_1) &= \mu_1\\ \delta(i_l) &= \mu_l - \mu_{l-1} \quad , l > 1\\ \delta(i) &= 0 \quad \textit{otherwise.}\end{aligned}$$

Then the demand δ is feasible.

Proof: Let B be any subset of $N \setminus \{s\}$ and let t be the largest index for which i_t still belongs to B. Then

$$\sum_{i \in B} \delta(i) \leq \sum_{l \leq t} \delta(i_l) = \mu_t \tag{4.2}$$

where the first equality follows, since for all unchosen nodes we have $\delta(i) = 0$ and t was the largest index which belonged to B, so all positive demands up to t are summed up on the right hand side where on the left hand side it might be that not all nodes $i_l, l \leq t$ are in B. The equality follows since the sum is a telescopic sum and so every term except the last μ_t is cancelled out.
We know that δ_t is feasible, so from Theorem 2 follows:

$$\mu_t = \delta_t(i_t) \leq \sum_{i \in B} \delta_t(i) \leq \sum_{j \in N \setminus B, i \in B} c_{ji} \tag{4.3}$$

where the first equality is due to the definition of μ_t. Combining (4.2) and (4.3) we get

$$\sum_{i \in B} \delta(i) \leq \mu_t \leq \sum_{j \in N \setminus B, i \in B} c_{ji} \tag{4.4}$$

so (4.1) holds and the constructed δ is feasible □

This lemma will be very useful, since we can apply it to the time-expanded graph. There we assign to each of the sink nodes $z(t), 0 \leq t \leq T$, a certain demand which should be maximal. We need the following definition of *maximal demand*:

Definition 6 (Gale [14]) *A* maximal demand *δ on a single source - single sink network is a feasible demand δ for which the value $\delta(z)$ is as large as possible.*

To construct the demand in such a way that it indicates an earliest arrival flow, we consider the time-expanded graphs $D(t)$ for each time step $0 \leq t \leq T$. The only sink of such a network $D(t)$ should be $z(t)$, which is easily achieved by hold-over arcs with infinite capacity, directing the flow from $z(0)$ to $z(1)$ and from $z(1)$ to $z(2)$ etc. In each network $D(t)$ the demand assigned to the sink node $z(t)$ should be maximal. We would like to have that these maximal demands $\delta_t(z(t))$ are increasing as t increases:

Lemma 2 (Gale [14]) *Let δ_t be the maximal demand on $D(t)$ and let $\mu_t = \delta_t(z(t))$. Then $\mu_t \leq \mu_{t+1}$ for all $0 \leq t \leq T$.*

Proof: Let x^t be the maximal flow in $D(t)$ such that $\mu_t = \delta_t(z(t)) = \sum_{j \in D(t)} x^t_{jz(t)}$. Now we can define a new demand $\tilde{\delta}$ on $D(t+1)$:

$$\begin{aligned} \tilde{\delta}(i) &= \tilde{\delta}_t(i) \text{ for } i \in D(t), i \neq z(t) \\ \tilde{\delta}(z(t+1)) &= \mu_t \\ \tilde{\delta}(i) &= 0 \text{ otherwise} \end{aligned}$$

that is all demands are conserved except the demand of $z(t)$ which is set to 0 and instead the demand of $z(t+1)$ is set to μ_t; all other nodes $i(t+1)$ have demand zero.
This demand is feasible, since flow x^t sends μ_t units into $z(t)$ and these can be directed to $z(t+1)$ via the hold-over arc $(z(t), z(t+1))$ which has infinite capacity. All other demands are also satisfied by x^t since they are unchanged or 0.
$\Rightarrow \delta_{t+1}(z(t+1)) \geq \mu_t$. □

Now the theorem stating the existence of earliest arrival flows is merely a corollary of Lemma 1 and Lemma 2:

Theorem 3 (Gale [14]) *Let $\delta_1, \ldots, \delta_T$ be maximal demands on $D(1), \ldots, D(T)$ and let $\mu_t = \delta_t(z(t))$. Then the demand δ, where $\delta(z(1)) = \mu_1$, $\delta(z(t)) = \mu_t - \mu_{t-1}$ for $t > 1$ and $\delta(i) = 0$ otherwise is feasible.*

The proof follows immediately since due to Lemma 2 the assumptions of Lemma 1 are fulfilled and Lemma 1 gives the feasibility.

4.3 Lexicographically Maximal Flows or How to Find Earliest Arrival Flows

Gale's proof is not constructive and it took some time until algorithms for finding earliest arrival flows have been developed. In the early 1970s Wilkinson [28] and Minieka [25] independently of each other came up with an algorithm for finding earliest arrival flows. In the following we present and discuss the approach of Minieka [25], which allows some interesting insight into the structure of arrival and departure patterns of maximal flows. Minieka develops his theory on static graphs and then translates it to dynamic flows using time-expanded graphs.

Minieka introduces the concept of lexicographically maximal flows on static networks with multiple sources and sinks.
Consider a static network $G = (N, A)$ with a set of sources $S, S \subset N, S \neq \emptyset$ and a set of sinks $Z, Z \subset N \setminus S, Z \neq \emptyset$. The classical maximal flow problem as given in (2.1) changes as follows to obtain a maximal flow problem with multiple sources and sinks:

$$\begin{aligned}
\text{maximize } & v(Z) := -\sum_{i \in Z}\Big(\sum_{j \in N}(x_{ij} - x_{ji})\Big) & \\
\text{subject to } & \sum_{j \in N}(x_{ij} - x_{ji}) = 0 & \forall i \in N \setminus (S \cup Z) \\
& \sum_{j \in N}(x_{ij} - x_{ji}) \geq 0 & \forall i \in S \\
& \sum_{j \in N}(x_{ij} - x_{ji}) \leq 0 & \forall i \in Z \\
& 0 \leq x_{ij} \leq c_{ij} & \forall i, j \in N
\end{aligned} \tag{4.5}$$

For a maximal flow let:

- $V(s)$ denote the greatest number of flow units that can leave source node set $s \subseteq S$.
- $V(z)$ denote the greatest number of flow units that can enter sink node set $z \subseteq Z$.
- $v(s)$ denote the smallest number of flow units that can leave source node set $s \subseteq S$.
- $v(z)$ denote the smallest number of flow units that can enter sink node set $z \subseteq Z$.

Observation 4 *Since the flow is maximal, we have* $V(S) = V(Z) = v(S) = v(Z)$.

Definition 7 *Let the sink set be structured as follows: $Z_1 \subseteq Z_2 \subseteq \ldots \subseteq Z_n \subseteq Z$.*
A lexicographically maximal flow on the sinks *is a maximal flow that sends $V(Z_i)$ units of flow into each subset $Z_i, i = 1, 2, \ldots, n$.*
A lexicographically minimal flow on the sinks *is a maximal flow that delivers $v(Z_i)$ flow units to each subset $Z_i, i = 1, 2, \ldots, n$.*
Similarly a lexicographically maximal flow on the sources and a lexicographically minimal flow on the sources can be defined.

Definition 8 *A* departure schedule *or* pattern *α is a function from $S \to \mathbb{R}^+$ assigning to every source node $s \in S$ a certain amount of flow $\alpha(s)$ that has to leave node s. An* arrival schedule *or* pattern *is a function $\omega : Z \to \mathbb{R}^+$ that assigns to every sink node $z \in Z$ an amount of flow $\omega(z)$ which has to arrive at the corresponding sink z.*

In his article Minieka [25] first shows the independence between departure and arrival schedules in maximal flows. If we consider maximal flows, then for *any* departure schedule and *any* arrival schedule there exists a maximal dynamic flow satisfying this departure schedule and this arrival schedule simultaneously.
We know from static network flow theory that every flow from S to Z can be decomposed into paths and cycle flows, where a path flow is a path that contains only forward arcs.

Lemma 3 (Minieka, [25]) *Let c_1 and c_2 be two circulations in graph G. The circulation c_2 can be constructed from the circulation c_1 by generating a finite sequence $C := \{C^1, C^2, \ldots, C^n\}$ of circulations in graph G, where $C^1 = c_1$ and $C^n = c_2$ and where C^i differs from C^{i-1} only in a flow change along a cycle in graph G, for $i = 2, 3, \ldots, n$.*

Using this lemma, we can show the independence between departure and arrival patterns for maximal flows:

Theorem 4 (Minieka,[25]) *If there exists a maximal flow f_1 in graph $G = (N, A)$ with $\alpha(s)$ units departing from each source $s \in S$ and if there exists a maximal flow f_2 in graph $G = (N, A)$ with $\omega(z)$ units arriving at each sink $z \in Z$, then there exists a maximal flow f_3 in $G = (N, A)$ with $\alpha(s)$ units leaving each source $s \in S$ and $\omega(z)$ units arriving at each sink $z \in Z$.*

Proof: Add a supernode B and arcs (B, s) to each source $s \in S$ and (z, B) to each sink $z \in Z$ with infinite capacity. Then f_1 and f_2 can be

regarded as circulations on this extended graph by directing the flow from sink set Z via supernode B to the source set S. Denote the flow value of f_1 on each arc $(i,j) \in A \cup \{(B,S)\} \cup \{(Z,B)\}$ by $f^1(i,j)$, and the flow value of f_2 on each arc $(i,j) \in A \cup \{(B,S)\} \cup \{(Z,B)\}$ by $f^2(i,j)$.
Now apply the following procedure to get flow f_3 with departure pattern $\alpha(s)$, $s \in S$, and arrival pattern $\omega(z)$, $z \in Z$:

1. Mark all arcs (i,j) with $f^1(i,j) \neq f^2(i,j)$.
2. Now consider only f_1.
3. Choose any sink $z \in Z$ at which less than $\omega(z)$ units of flow arrive (with respect to f_1).
4. There exists a marked arc (i,z) on which $f^1(i,z) < f^2(i,z)$.
5. Label this arc with *INCREASE*, for we want to increase the flow along this arc.
6. To ensure the flow conversation for each node we have to continue this flow-change procedure. That is, choose another marked arc incident to node i and if this arc is of the from (j,i) and $f^1(i,j) < f^2(i,j)$ label it with *INCREASE*. If the arc is of the form (i,j) and $f^1(i,j) > f^2(i,j)$ label it with *DECREASE*. Go on to choose and label arcs in this way (avoiding arcs of the form (B,s)) until you get a cycle.
7. The largest possible flow change on arcs (i,j) labeled *INCREASE* is the minimum of the flow differences $f^2(i,j) - f^1(i,j)$ and the largest possible flow change on arcs (i,j) labeled *DECREASE* is the minimum of the flow differences $f^1(i,j) - f^2(i,j)$. Choose the minimum of these largest possible flow changes to ensure feasibility and make this flow change.
8. Go to Step 3 and repeat this procedure until at every sink $\omega(z)$ units of flow arrive.

Some verification of this procedure is necessary:
If $f_1 = f_2$ then of course there is no arc to mark and no sink to choose and $f_3 = f_1 = f_2$ and otherwise there must exist marked arcs. First we have to ensure the existence such incident marked arcs as claimed in Step 6: Since f_1 delivers less flow units into the considered sink z than f_2 there must exist an arc (i,z) with $f^1(i,z) < f^2(i,z)$. By Lemma 3 we know that there must be at least one other marked arc incident to

node i. Remember that f_1 is a feasible maximal flow and therefore it fulfils the flow conversation. Since $f^1(i,z) < f^2(i,z)$, it could be that less flow arrives in i with f_1 than with f_2 or that f_1 sends more flow out of node i via a different arc (i,j) than f_2. Thus we always find a marked arc incident to such a node i with properties as claimed in Step 6.

Since we do not want the procedure to change any of the $\alpha(s)$ we have to argue why the flow changing cycle never includes any of the source nodes: If the flow-change procedure reaches supernode B first via an arc $(\tilde{z}, B)$, then $f^1(\tilde{z}, B) > f^2(\tilde{z}, B) = \omega(\tilde{z})$. Then we can use arc (B, z) to finish the cycle. Arc (B, z) must be marked since the flow that arrives at z in f_1 is less than $\omega(z)$. Obviously a flow-change along this cycle does not change the flow of any of the source nodes.
Assume that the flow-change procedure first reaches supernode B via an arc of the form $(B, s),\ s \in S$. This means that additional flow can be sent from s to z which is a contradiction to fact that f_1 is a maximal flow.
□

To illustrate the result and the procedure we give the following example:

Example 7 *Consider the network given in Figure 4.4 with three sources and two sinks. We have already included the supernode B to perform the procedure given in the proof of Theorem 4. The two maximal flows f_1 and f_2 are coloured red (f_1) and green (f_2), resp. The arcs on which the flow values differ are printed in bold type. Now we apply Theorem 4 to get a maximal dynamic flow f_3 with the departure pattern of f_1 and the arrival pattern of f_2. So we want that* 15 *units of flow leave source s_1,* 25 *flow units leave source s_2 and* 10 *units of flow leave source s_3. At sink z_1* 20 *flow units should arrive and* 30 *flow units at sink z_2.*
In Step 3 of the procedure we choose sink z_2, since f_1 only delivers 20 *flow units there whereas in f_2* 30 *units of flow arrive at sink z_2. In Step 4 we choose arc $(5, z_2)$, which is marked and on which the flow to z_2 can be increased. The next marked arc incident to node* 5 *is $(5, z_1)$, where we have to decrease the flow according to Step 6. Arc (z_1, B) is the only other marked arc incident to z_1 so we choose it and decrease the flow along this arc. To node B several marked arcs are incident and we choose arc (z_2, B), since we want to avoid arcs of the form (B, s). Along this arc the flow needs to be increased. The largest possible change along the flow change cycle $z_2 - 5 - z_1 - B - z_2$ is*

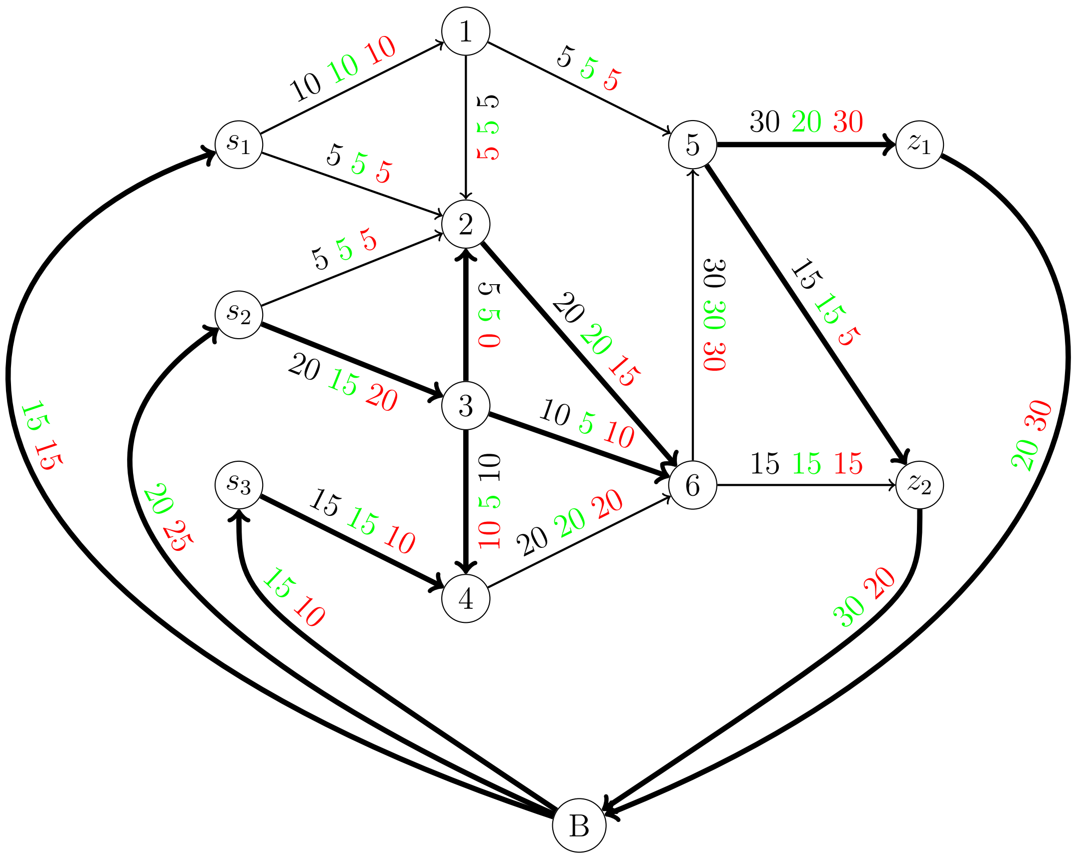


Figure 4.4: Static network with two different maximal flows. The flow values on the arc of f_1 are given in red and those of f_2 in green.

10 *flow units. The new maximal flow we get already fulfils the requirements and therefore is the searched* f_3.

We have seen that for every departure pattern on the sources, corresponding to a maximal flow f_1, and every arrival pattern on the sinks, corresponding to another maximal flow f_2, it is possible to find a maximal flow f_3 fulfilling this arrival *and* departure pattern simultaneously.
In the following we prove that lexicographically maximal flows always exist. We give the proof for lexicographically maximal flows on the sinks, since earliest arrival flows are just a special case of these flows, as we see later on. The proof for the existence of lexicographically maximal flows on the sources follows along the same lines as the one we give for the sinks.

Theorem 5 (Minieka,[25]) *Let* $G = (N, A)$ *be any finite graph with integer capacities for all arcs* $(i, j) \in A$. *If* $Z_1 \subseteq Z_2 \subseteq \ldots \subseteq Z_n = Z$, *then there exists a maximal flow that delivers* $V(Z_i)$ *units into subset* Z_i *for all* $i = 1, \ldots, n$.

Proof: We use an induction argument to prove the statement.

Basis:
Claim: For a subset $Z_1 \subseteq Z$ there exists a maximal flow in G that delivers $V(Z_1)$ units into this subset.
Proof of claim: Let $\tilde{G}$ be the modified graph of G, obtained by deleting all sink nodes $z \in Z \setminus Z_1$ and all arcs pointing to these sinks; i.e. the only sinks in $\tilde{G}$ are $z \in Z_1$. Apply a flow augmenting path algorithm to this modified graph $\tilde{G}$. The algorithm finds a maximal flow which delivers the maximum possible amount of flow into Z_1, i.e. per definition $V(Z_1)$.
Now apply the flow augmenting path algorithm with this flow as initial flow to the original network G, i.e. all sinks are open. Assume that a flow augmenting path generated by the algorithm contains a node in $Z_1 \subset Z$. This is a contradiction to the initial flow being maximal into subset Z_1. Therefore the flow into subset Z_1 is not altered during the repeated execution of the flow augmenting path algorithm. Thus we get a maximal flow on G sending $V(Z_1)$ units into set Z_1.
Inductive Hypothesis: There exists a maximal flow on G that delivers $V(Z_k)$ units into each subset $Z_1 \subseteq Z_2 \subseteq \ldots \subseteq Z_l$ for $k = 1, \ldots, l$.
Inductive Step: To construct a flow that fulfils the inductive hypothesis for the first $l + 1$ subsets, we add to graph G two extra nodes B_1

and B_2. Connect all sinks in Z_{l+1} to node B_1 and all sinks in $Z \setminus Z_{l+1}$ to B_2 and also connect B_2 to all sources. Let all the arcs have infinite capacity. In this augmented graph we regard B_2 as the source node and B_1 as the sink node. So the flow into subset Z_{l+1} in G equals the flow into B_1 in the augmented graph. Obviously there is a one to one correspondence between flows in G and in the augmented graph.
Due to the inductive hypothesis we know that there exists a maximal flow f in graph G that sends $V(Z_k)$ units into each subset $Z_k,\ k = 1, \ldots, l$. To obtain the image of this flow f in the augmented graph, we have to assign to each arc (B_2, s) the flow leaving source s and to each arc $(z, B_1)\ , z \in Z_{l+1}$ and $(z, B_2)\ , z \in Z \setminus Z_{l+1}$, resp., the flow entering the sink node z. Now we use this image of f as initial flow for the flow augmenting path algorithm in the augmented graph.
Obviously no flow augmenting path generated by the algorithm can contain a source node $s \in S$, since otherwise f had not been a maximal flow in G.
Also no flow augmenting path can contain any node in Z_l, since that would imply that additional flow could be send into Z_l, contradicting the inductive hypothesis.
So the flow augmenting path algorithm constructs a maximal flow in the augmented graph without changing the flow of any node in $S \cup Z_l$. The image of this flow in the original graph G is a maximal flow that delivers $V(Z_{l+1})$ units into Z_{l+1} and due to the argumentation above and the inductive hypothesis the flow delivers $V(Z_k)$ units into each subset $Z_k,\ k = 1, \ldots, l+1$. $\square$

The existence of lexicographically minimal flows can be reduced to the existence of lexicographically maximal flows.

Clearly these results apply to the time-expanded graph of any dynamic network $D(T)$. So if

$$Z_k := \bigcup_{t=0}^{k} z(t) \quad , t = 0, 1, \ldots, T$$

and

$$S_k := \bigcup_{t=0}^{k} s(t) \quad , t = 0, 1, \ldots, T$$

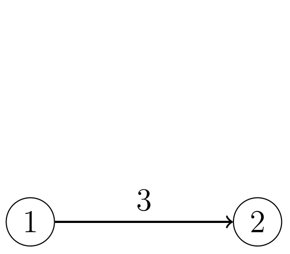
then:

- A lexicographically maximal flow in $D(T)$ on the sinks is a maximal dynamic flow on G with earliest possible arrival schedule.
- A lexicographically maximal flow in $D(T)$ on the sources is a maximal dynamic flow on G with earliest possible departure schedule.
- A lexicographically minimal flow in $D(T)$ on the sinks is a maximal dynamic flow on G with latest possible arrival schedule.
- A lexicographically minimal flow in $D(T)$ on the sources is a maximal dynamic flow on G with latest possible departure schedule.

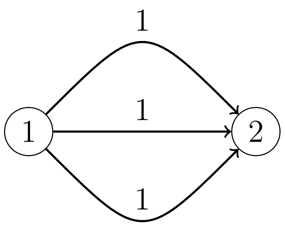
Note, that earliest arrival flows are just a special case of the more general lexicographically maximal flows. Also the notion of lexicographically maximal and minimal flows, respectively, on the sinks and sources, respectively, gives us the possibility to arrange certain departure and arrival patterns, as specified above. So the work of Minieka is much more general than coping only with earliest arrival flows.

Example 8 *Consider the time-expanded graph given in Figure 3.2. The sets Z_k look as follows: $Z_0 = \{z(0)\}$, $Z_1 = Z_0 \cup \{z(1)\}$, $Z_2 = Z_1 \cup \{z(2)\}$, $Z_3 = Z_2 \cup \{z(3)\}$, $Z_4 = Z_3 \cup \{z(4)\}$, $Z_5 = Z_4 \cup \{z(5)\}$, $Z_6 = Z_5 \cup \{z(6)\}$. A lexicographically maximal flow in $D(T)$ on the sinks maximizes the amount of flow entering Z_k subject to the constraint that the flow entering Z_{k-1} is maximal. So we first have to maximize the flow entering Z_0, which is 0, since no flow can arrive in 0 time. Similarly the maximal amount of flow for Z_1, Z_2 and Z_3 is zero. For Z_4 the maximal possible amount of flow entering Z_4 is 2, since the path $s-1-2-z$ takes 4 time units and thus flow travelling via this path can reach $z(4) \in Z_4$. The maximal flow arriving in Z_5 is 7, since 5 more flow units can be send into Z_5 via $s-1-z$ and we have to add them to the 2 flow units from Z_4. The maximal flow arriving in Z_6 is 20, where 13 units are added to the amount in Z_5 corresponding to the amount of flow entering $z(6)$. Obviously this is indeed a maximal flow with earliest arrival schedule.*

Minieka shows that there is a symmetry between earliest arrival and latest departure schedules by considering the network $\hat{G}$ consisting of the same nodes as G but all directions of the arcs are reversed.
He furthermore develops an algorithm for finding a latest departure and earliest arrival schedule which is based on the algorithm of Ford and Fulkerson [11]:
Consider the following modified time-expanded network $\tilde{D}(T)$: replace each



(a) Original capacity $c_{12} = 3$.



(b) Modified arcs $(1, 2)$.

Figure 4.5: Modification of arcs.

arc $(i, j)(t)$ with $c_{ij} > 1$ by c_{ij} arcs of capacity one; for an illustration consider Figure 4.5.
Using this modified time-expanded network $\tilde{D}(T)$ we can state the algorithm of Minieka, given in Algorithm 5.

Algorithm 5 Earliest Arrival and Latest Departure Maximal Dynamic Flow Algorithm [25]

```
Input: modified time-expanded graph $\tilde{D}(T)$
Output: Earliest arrival and latest departure maximal dynamic flow.
Begin
Apply the maximal dynamic flow Algorithm 1 to find a maximal dynamic
flow on the underlying static network $G = (N, A)$.
Whenever a flow augmenting path is found, consider the corresponding
arcs of each copy of this path in $\tilde{D}(T)$:
if the arc is a forward arc then
    label it with the time when the flow enters this arc.
else
    remove the former label.
end if
End
```

At termination these labels indicate a latest departure - earliest arrival schedule. The algorithm of Minieka works on the time-expanded graph. Since the size of this graph grows exponentially with T, Algorithm 5 is only pseudo polynomial. In 1973 Zadeh[29] constructed bad networks for shortest augmenting paths algorithms and thus proved that, in the worst case, the algorithm of Minieka needs an exponential number of general chain flows.

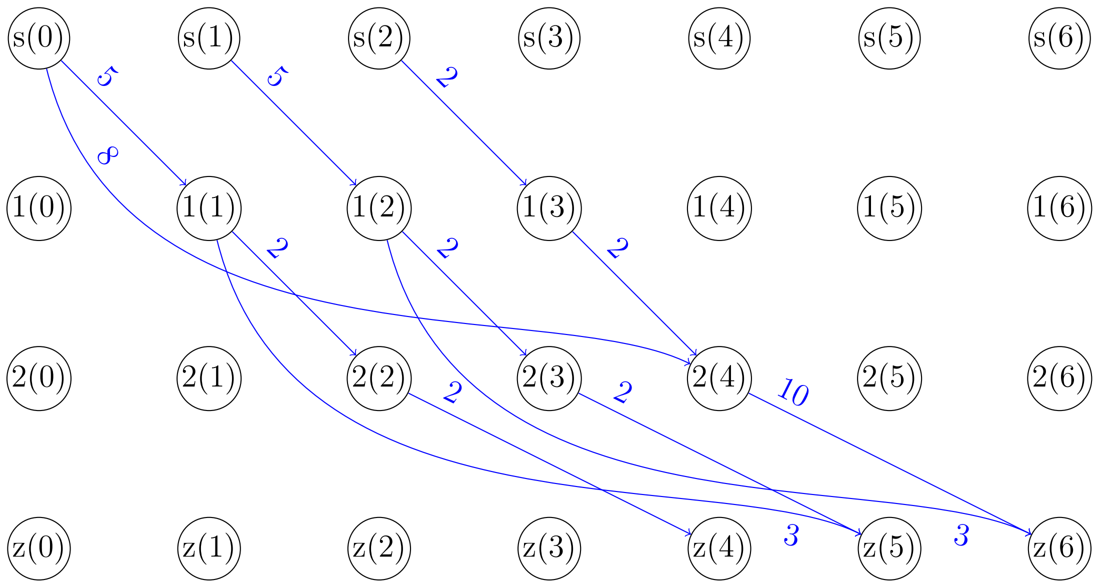


Figure 4.6: Latest departure - earliest arrival flow.

Example 9 *Consider again the graph given in Figure 3.1 with time horizon* $T = 6$*. Applying Algorithm 5, yields the latest departure - earliest arrival schedule shown in Figure 4.6. The numbers on the arcs denote the units of flow using that path. Note, that this flow is not the same as in Figure 4.1, due to the difference in the departure schedules: In Figure 4.1* 15 *units leave* $s(0)$ *and* 5 *units leave node* $s(1)$*, but in Figure 4.6 only* 13 *units leave* $s(0)$*,* 5 *flow units leave* $s(1)$ *and* 2 *units leave* $s(2)$*. We see that in Figure 4.1 we have only an earliest arrival flow but not a latest departure - earliest arrival flow as in Figure 4.6.*

4.4 A Polynomial Time Approximation Algorithm for the Earliest Arrival Flow Problem

As we have seen in the previous section, the exact algorithm of Minieka [25] is a pseudo-polynomial algorithm since it depends directly on T. The algorithm of Wilkinson [28] also works on the time-expanded graph, and thus is a pseudo polynomial algorithm. The main problem that makes computing earliest arrival flows that expensive, is that we cannot always find an earliest arrival flow that has the temporally repeated flow property, as we have shown in Observation 3. We will see in the next chapter, that for a special class of

graphs, we can always find an earliest arrival flow which has the temporally repeated flow property and thus the computation of such an earliest arrival flow is much easier.
For a long time not even a polynomial time approximation algorithm for the earliest arrival flow problem was known. In 1994 Hoppe and Tardos [19] presented the first polynomial time approximation algorithm for the earliest arrival flow problem, which gives a $(1+\epsilon)$ approximation for any fixed $\epsilon > 0$.

4.4.1 Generalized Chain-Decomposable Flows

Hoppe and Tardos solve the problem of the non-existence of temporally repeated flows for the earliest arrival problem by generalizing the chain decomposition defining a temporally repeated flow. The idea of these generalized chain flows corresponds to the idea of Minieka, who transferred the shortest augmenting paths directly to the time-expanded network.

The temporally repeated flows defined in Chapter 3 are a subclass of so-called (general) chain-decomposable flows which were introduced by Hoppe[20] and Hoppe and Tardos [19]. Recall that we defined temporally repeated flows - or standard chain-decomposable flows, as Hoppe and Tardos name them - via a standard chain decomposition $\mathbb{P}_s$ of flow x. This means that the flow chains are only allowed to use arcs in the same direction as the flow x does.

Definition 9 *(Generalized chain decomposition and general chain decomposable flows) A* generalized chain decomposition $\mathbb{P}$ *of a dynamic flow x is a set of chain flows $\{\gamma_1, \dots, \gamma_q\}$, where each chain flow may use arcs in the opposite direction of the flow direction. This means that a chain flow might use a residual arc with negative transit time.*
A general chain decomposable flow *is obtained by repeating these chain flows γ_l, $l = 1, \dots, q$. The first repetition starts at $t = 0$ and the last one at $T - \tau_{\gamma_l}$, where τ_{γ_l} is the total transit time of chain flow γ_l.*

Example 10 *Consider again our example from Chapter 3 with time horizon $T = 7$ and the non-standard chain decomposition of a static flow. In Figure 4.7 chain flow γ_1 is marked red, and chain flow γ_2 is coloured blue, each with flow value 2. In Figure 4.8 is shown the time-expanded graph corresponding to γ_1, and in Figure 4.9 the time-expanded graph corresponding to γ_2. Adding γ_1 and γ_2 together this non-standard chain decomposition induces the dynamic flow shown in Figure 4.10.*

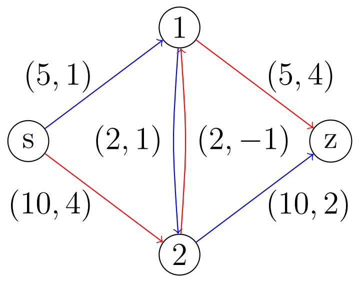


Figure 4.7: Chain flows

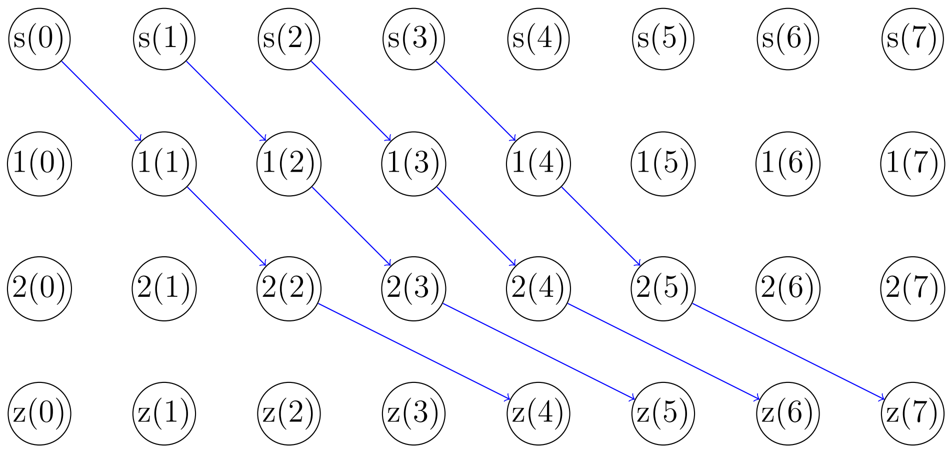

s(0) s(1) s(2) s(3) s(4) s(5) s(6) s(7)
1(0) 1(1) 1(2) 1(3) 1(4) 1(5) 1(6) 1(7)
2(0) 2(1) 2(2) 2(3) 2(4) 2(5) 2(6) 2(7)
z(0) z(1) z(2) z(3) z(4) z(5) z(6) z(7)

Figure 4.8: Time-expanded graph corresponding to chain flow γ_1.

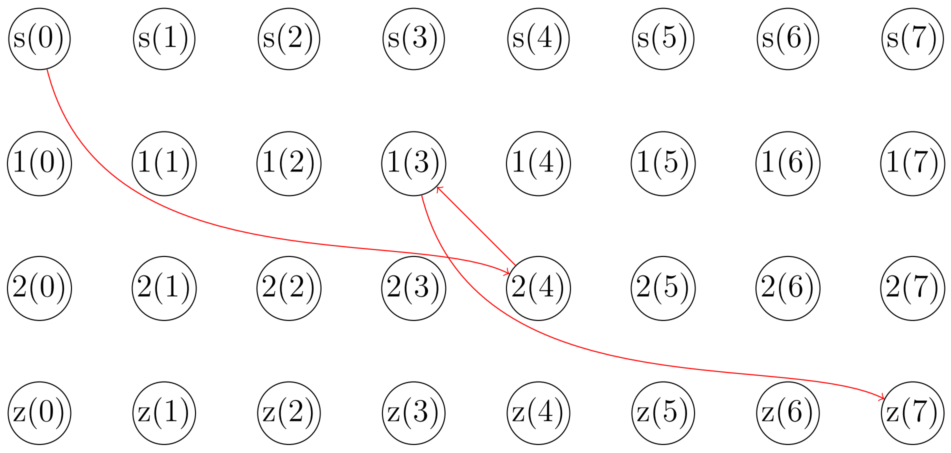

s(0) s(1) s(2) s(3) s(4) s(5) s(6) s(7)
1(0) 1(1) 1(2) 1(3) 1(4) 1(5) 1(6) 1(7)
2(0) 2(1) 2(2) 2(3) 2(4) 2(5) 2(6) 2(7)
z(0) z(1) z(2) z(3) z(4) z(5) z(6) z(7)

Figure 4.9: Time-expanded graph corresponding to chain flow γ_2.

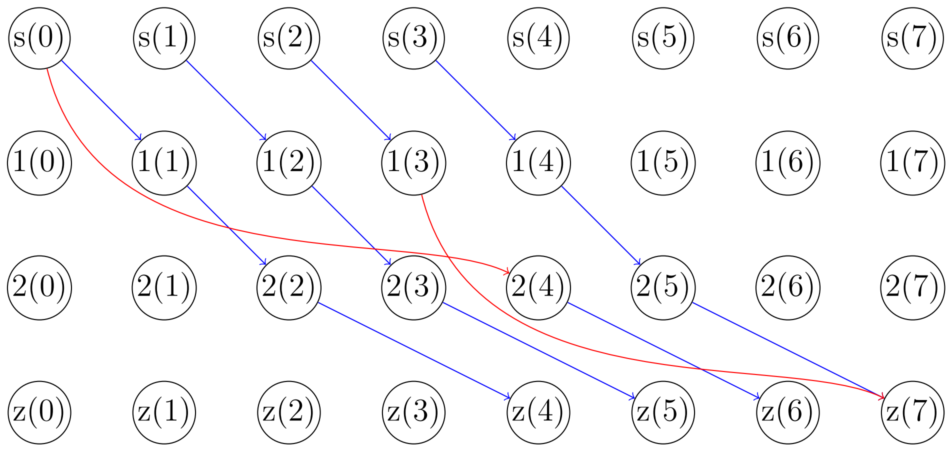


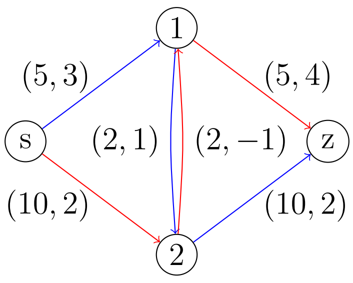
Figure 4.10: Time-expanded graph of the non-standard chain decomposition $\gamma_1 + \gamma_2$.

Using non-standard chain decompositions $\mathbb{P}$, feasibility of the resulting flow $x = \sum_{\gamma \in \mathbb{P}} \gamma$ is no longer clear. It depends on the timing, when the dynamic flow reaches an arc via the chain flow used.

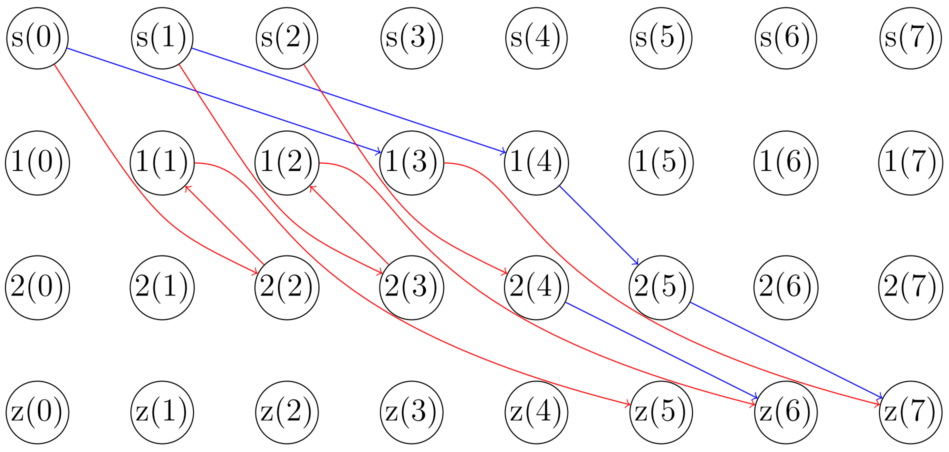
Example 11 *Consider the network given in Figure 4.11 (transit times are changed!) with time horizon* $T = 7$ *and chain flows* $\gamma_1 = 1 - 2 - 3 - 4$ *and* $\gamma_2 = 1 - 3 - 2 - 4$*. Chain flow* γ_2 *reaches node* 2 *at time* $2 + (-1) = 1$*, but* γ_1 *only reaches node* 2 *at time* 3*. Adding* γ_1 *and* γ_2 *the resulting dynamic flow is infeasible, since* γ_2 *uses arcs* $(3(2), 2(1))$ *and* $(3(3), 2(2))$*. But there is no flow on the forward correspondents* $(2(1), 3(2))$ *and* $(2(2), 3(3))$ *and thus it is not possible to take back flow on these arcs.*

4.4.2 Earliest Arrival Flow Approximation Algorithm of Hoppe and Tardos

In the following we derive and explain the algorithm of Hoppe and Tardos and estimate how good the approximation is. First remember that Zadeh [29] showed that it might happen that we need exponentially many chain flows. Thus it is important to bound the number of chain flows by a polynomial, to get a polynomial approximation algorithm. To get such a bound Hoppe and Tardos develop a clever capacity scaling algorithm:
Let all the capacities be bounded by some integer U. Hoppe and Tardos develop a capacity scaling shortest augmented path algorithm, with the unusual feature of scaling upwards. This means that the algorithm starts with $\Delta := 1$ and increases Δ by setting $\Delta := 2\Delta$ after every iteration until no



(a) Dynamic network G with chain flows γ_1 and γ_2 coloured.



(b) Corresponding time-expanded network.

Figure 4.11: Infeasible flow resulting of a non-standard chain decomposition.

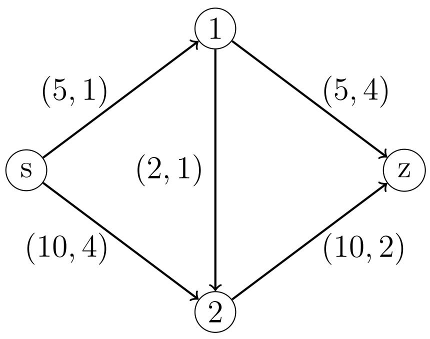
s-z-path of length less or equal than T exists. Since in a dynamic flow a short arc with small capacity might carry more flow than a long arc with large capacity, scaling downwards (as it is usually done) does not yield any good results.
The algorithm works on the residual network $G'(x) = (N, A'(x), T)$ of the dynamic network $G = (N, A, T)$, where the capacities are updated according the to flow changes and are rounded at the end of each iteration by the increasing scaling factor Δ. The update according the changing flow values is done as given in Definition 4. At the beginning the capacities are not yet rounded and thus the shortest augmenting paths found in the first iteration are exact paths. In further iterations, because of the rounding, all capacities are integer multiples of Δ. Thus and since all initial capacities were integers, the flows found in each iteration have at least flow value Δ in the static network.

Algorithm 6 Polynomial Time Approximation Algorithm for the Earliest Arrival Flow Problem [19]

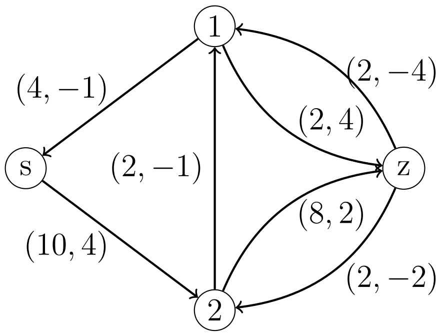
```
*Input:* dynamic network $G = (N, A, T)$ with capacity function $c$ and transit time function $\tau$ $\forall(i,j) \in A$ and let $m$ denote the number of arcs in $A$; chain decomposition set $\mathbb{P} = \emptyset$, scaling factor $\Delta := 1$, rounded capacity function $\tilde{c} := c$, flow $x := 0$, some $\epsilon > 0$
*Output:* $(1+\epsilon)$ Approximation of the Earliest Arrival Flow.
**while** there exists a $s$-$z$-path in $G'_{\tilde{c}}(x)$ with length $\leq T$ **do**
  set $\sigma := 0$
  **while** ($\sigma < \frac{m\Delta}{\epsilon}$) and (there exists a $s$-$z$-path in $G'_{\tilde{c}}(x)$ with length $\leq T$) **do**
    find the shortest $s$-$z$-path in $G'_{\tilde{c}}(x)$ and denote it by $P$
    $l := \min(\tilde{c_{ij}} \,|(i,j) \in P)$
    augment the flow $x$ by $l$ along $P$ and update the residual capacities $\tilde{c}$
    update the chain decomposition set $\mathbb{P} := \mathbb{P} \cup \{\langle l; P\rangle\}$
    set $\sigma := \sigma + l$
  **end while**
  increase the scaling factor: $\Delta := 2\Delta$
  round the residual capacities: $\forall(i,j) \in A'(x)$ set $\tilde{c_{ij}} := \tilde{c}_{ij} - (\tilde{c}_{ij} \bmod \Delta)$
**end while**
If $\mathbb{P} \neq \emptyset$ the dynamic flow $x$ can be obtained by repeating all path flows in $\mathbb{P}$.
```

Example 12 *Let us consider again the following graph shown in Figure*

4.12a, with time horizon $T = 7$ and $\epsilon = 1.25$ to see how Algorithm 6 works.



(a) Dynamic Network



(b) Network after first rounding

Figure 4.12: Example illustrating Algorithm 6.

First we set the scaling factor $\Delta := 1$, the flow $x := 0$ and $\mathbb{P} := \emptyset$. There exists a s-z-path with length less or equal than T. We start with $\sigma := 0$ and since $m \cdot \Delta/\epsilon$ $=5 \cdot 1/1.25 = 4$ is larger than 0 we apply the inner loop. The shortest s-z-path is $P = (1, 2, 3, 4)$ with length 4 and minimum residual capacity $l = 2$. Thus we augment the flow x along P by l, get flow value $v(x) := 2$ and update the residual capacities. Add P to the chain decomposition set $\mathbb{P}$ and set $\sigma := \sigma + 2 = 2$.
Since $\sigma \leq 4$ is still true and there still exists an s-z-path, we repeat the inner loop. The shortest path is now $P = (1, 2, 4)$ with length 5 and flow value 3. Update the residual network and set $\sigma := \sigma + 3 = 5$ which is larger than 4. Thus we have to increase the scaling factor. Set $\Delta := 2 \cdot \Delta = 2$ and round the capacities as shown in Figure 4.12b.

In the next iteration there still exists an s-z-path with length no more than T and thus we set $\sigma := 0$ and start the inner loop. Now we have $m \cdot \Delta/\epsilon$ $=5 \cdot 2/1.25 = 8$ which is of course greater than σ. The shortest s-z-path in the rounded residual network is $P = (1, 3, 4)$ with length 6 and flow value 8. Thus we get $v(x) := v(x) + 8 = 5 + 8 = 13$ and update the residual capacities.

We have to set $\sigma := 8$ and thus cannot repeat the inner loop once again, since σ is not less than 8. Therefore we increase the scaling factor and get $\Delta = 4$ and round the arc capacities. We see that there exists no longer a s-z-path.

To prove the feasibility of Algorithm 6 and to estimate later on how good the algorithm is, we need the following notation:

- Say we have $q + 1$ scaling phases during the algorithm, numbered $0, \ldots, q$. The index of the phases is related to the value $\Delta := 2^k$ during the inner loop of phase k.
- Let $\mathbb{P}_k$ denote the set of chain flows at the *end* of phase k, and $\mathbb{P}_{-1} := \emptyset$.
- Let T_k denote the length of the longest chain flow in $\mathbb{P}_k$.
- Let x_k denote the static flow *after* phase k.
- Let $\tilde{c}_k$ denote the capacity function (according to the flow, e.g. x_k) *before* the rounding of phase k.
- Let Λ_k^* denote a chain decomposition inducing an earliest arrival flow in the residual graph $G'_{\tilde{c}_k}(x_k)$.
- Let $\tilde{\Lambda}_k$ denote a chain decomposition inducing an earliest arrival flow in the further rounded residual network $G'_{\tilde{c}_{k+1}}(x_k)$. This means that here the capacities still correspond to flow x_k, but now they are rounded.

Theorem 6 (Hoppe,[20]) *Let $\mathbb{P}_q^T$ denote the dynamic flow induced by the general chain decomposition $\mathbb{P}_q$ found by Algorithm 6. Then $\mathbb{P}_q^T$ is a feasible dynamic flow.*

PROOF: Argumentation via induction on the number of scaling phases:

Induction Beginning: In the first scaling phase, when $\Delta = 1$, the algorithm finds exact augmenting paths and thus is identical to the algorithm of Minieka in this phase. Therefore the first chain decomposition $\mathbb{P}_0$ induces a feasible dynamic flow.

Inductive Hypothesis: $\mathbb{P}_k$ is a feasible dynamic flow, where $0 \leq k < q$.

Inductive Step: Phase $k+1$ starts with rounded capacity function $\tilde{c}_{k+1}$. We have to show two claims:

Claim 1: $\mathbb{P}_{k+1} - \mathbb{P}_k$ induces a feasible dynamic flow on the residual network $G'_{\tilde{c}_{k+1}}(x_k)$.

Proof of Claim 1: First note that

$$\mathbb{P}_{k+1} - \mathbb{P}_k = \left\{\gamma_{k+1}^1, \ldots, \gamma_{k+1}^l\right\}$$

where each of these chain flows is a shortest augmenting path. Thus the flow conservation constraints are trivially true for $\mathbb{P}_{k+1} - \mathbb{P}_k$.

Consider the capacity constraints of an arc (i,j) at time $0 \leq t \leq T$. If the flow of each of these chain flows $\gamma^1_{k+1}, \ldots, \gamma^l_{k+1}$ has value 0 on arc (i,j), i.e. for arc (i,j) nothing has changed, then the capacity constraint of (i,j) is obviously not violated. Otherwise, at least one of the chain flows, say $\hat{\gamma}_{k+1}$, sends flow via (i,j) at time t. This means that $\tau_{\hat{\gamma}_{k+1}}(s,i) \leq t \leq T - \tau_{\hat{\gamma}_{k+1}}(i,z)$, where $\tau_{\hat{\gamma}_{k+1}}(s,i)$ denotes the time flow on chain flow $\hat{\gamma}_{k+1}$ needs to travel from s to i; analogously $\tau_{\hat{\gamma}_{k+1}}(i,z)$ is defined. Since we find shortest augmenting path after shortest augmenting path, this travel time increases monotonically with k. Thus for any $j < k+1$, if γ_j uses arc (i,j) or its residual correspondent (j,i), then it also sends flow unequal to zero over (i,j) at time t. So the flow value on arc (i,j) stored in the residual network before $\hat{\gamma}_{k+1}$ is found, truly represents the flow on (i,j) at time t when $\hat{\gamma}_{k+1}$ first reaches arc (i,j).

Thus, and since $\mathbb{P}_{k+1} - \mathbb{P}_k = \left\{\gamma^1_{k+1}, \ldots, \gamma^l_{k+1}\right\}$ and each chain flow γ^n_{k+1}, $n = 1, \ldots, l$ is a feasible flow in the residual network $G'_{c_{k+1}}(x_k + \left\{\gamma^1_{k+1}, \ldots, \gamma^{n-1}_{k+1}\right\})$, the capacity constraints are not violated.

Claim 2: $\mathbb{P}_{k+1}$ induces a feasible dynamic flow on the original network G.

Proof of Claim 2: In the algorithm the capacity is rounded after each phase, and the rounding does not increase the capacity of any arc. Thus $\mathbb{P}_{k+1} - \mathbb{P}_k$ induces a feasible dynamic flow in the residual network $G'_c(x_k)$ where x_k is the flow after iteration k and c is the original, not-rounded capacity function. Next notice that for every node i the shortest residual $(s,i)-$ and $(i,z)-$ path lengths cannot decrease between scaling phases. Thus, if some chain flow in $\mathbb{P}_{k+1} - \mathbb{P}_k$ uses arc (i,j) and sends flow over it at time $0 \leq t \leq T$, then every chain flow in $\mathbb{P}_k$ that uses arc (i,j) must also send flow over (i,j) at this time t. Since $\mathbb{P}_{k+1} - \mathbb{P}_k$ is a feasible flow on the residual network (Claim 1), and the flow that is already sent over any arc (i,j) by a chain flow in $\mathbb{P}_0, \ldots \mathbb{P}_k$ is stored in the residual network, it follows that $\mathbb{P}_{k+1}$ induces a feasible dynamic flow on the original network.

□

The maximum dynamic flow value in the rounded network might be less than in the original network. Thus, to find out how good the approximation of Algorithm 6 is, we need to estimate the amount of lost dynamic flow. First we consider the loss of dynamic flow due to a single rounding: For any time t, $0 \leq t \leq T$, the decrease in the dynamic flow value caused by rounding at the end of phase k is $v_t(\Lambda_k^*) - v_t(\tilde{\Lambda}_k)$. (This notation, e.g. $v_t(\Lambda_k^*)$, denotes the flow value of the flow that reaches the sink at time t, where the flow is induced by the chain decomposition given in brackets; in the example Λ_k^*.) Hoppe and Tardos give the following lemma to bound this loss:

Lemma 4 (Hoppe and Tardos, [19]) *If* $0 \leq k \leq q$ *and* $0 \leq t \leq T$*, then*

$$v_t(\Lambda_k^*) - v_t(\tilde{\Lambda}_k) \leq \epsilon \cdot v_t(\mathbb{P}_k - \mathbb{P}_{k-1}). \tag{4.6}$$

PROOF: Denote the static flow that results by summing up all chain flows in Λ_k^* by x_k^*. Construct the "difference flow" $\hat{x}_k$ from x_k^* by applying the following procedure:

While there exists an arc in x_k^* violating the further rounded capacity function $\tilde{c}_{k+1}$ do:

1. Find such and arc (i, j) on which x_k^* sends more flow than $\tilde{c}_{k+1}$ allows.

2. Subtract from x_k^* some 2^k-valued s-z-chain flow (in $G'_{\tilde{c}_k}(x_k)$) that uses this arc (i, j).

Let m be the number of arcs in the original graph. In the rounding at the end of phase k the capacity might decrease on at most every arc and is decreased by at most 2^k. Thus the above procedure subtracts no more than m chain flows from x_k^*. Furthermore, every chain flow that is subtracted has at least length T_k, since after phase k there is no s-z-path of length less than T_k in the residual network $G'_{\tilde{c}_k}(x_k)$. Let the standard chain decomposition of the resulting flow $\hat{x}_k$ be denoted by $\hat{\Lambda}_k$. We get the following estimation:

$$v_t(\Lambda_k^*) - v_t(\hat{\Lambda}_k) \leq m \cdot 2^k(t - T_k + 1).$$

Note that $\hat{\Lambda}_k$ induces a feasible solution of the earliest arrival flow problem which is defined on the further rounded residual network $G'_{\tilde{c}_{k+1}}(x_k)$. $\tilde{\Lambda}_k$ also induces a feasible solution for the earliest arrival flow problem on $G'_{\tilde{c}_{k+1}}(x_k)$ and it is optimal, by definition. This means, that

$v_t(\hat{\Lambda}_k) \leq v_t(\tilde{\Lambda}_k)$ and thus

$$v_t(\Lambda_k^*) - v_t(\tilde{\Lambda}_k) \leq m \cdot 2^k(t - T_k + 1). \tag{4.7}$$

Due to the stopping criterion of the inner loop of Algorithm 6 and since $0 \leq k < q$, we know that the chain flows of $\mathbb{P}_k - \mathbb{P}_{k-1}$ have at least a total value of $m2^k/\epsilon$. Furthermore each of these chain flows has length at most T_k and we get

$$v_t(\mathbb{P}_k - \mathbb{P}_{k-1}) \geq (m2^k/\epsilon)(t - T_k + 1). \tag{4.8}$$

Putting (4.7) and (4.8) together we get the claim. □

Theorem 7 (Hoppe and Tardos,[19]) *Let $G = (N, A, T)$ be a dynamic graph with m arcs and n nodes and let $0 \leq t \leq T$. Denote the maximal dynamic flow value in time t by v_t^*. Algorithm 6 computes a non-standard chain decomposition $\mathbb{P}_q$ in time $O(\frac{m}{\epsilon}(m + m\log n)\log U)$ such that $v_t^* \leq (1+\epsilon)v_t(\mathbb{P}_q)$.*

Proof: The running time follows easily since there are at most $O(\log U)$ scaling phases. The rounding ensures that in every iteration every augmenting path that is found has at least flow value Δ. Thus in each iteration the constraint $\sigma < m \cdot \Delta/\epsilon$ guarantees that there are at most m/ϵ augmentations. The $O(m + n\log n)$ term is the complexity needed for computing the shortest path, according to [13].
To prove the approximate optimality we make use of Lemma 4. Note that in phase $k+1$ the algorithm starts computing an earliest arrival flow in the rounded residual network $G'_{\tilde{c}_{k+1}}(x_k)$, where $\tilde{c}_{k+1}$ is the capacity function after the rounding at the end of phase k. Thus we have

$$v_t(\mathbb{P}_{k+1} - \mathbb{P}_k + \Lambda_{k+1}^*) = v_t(\tilde{\Lambda}_k). \tag{4.9}$$

By a similar consideration we get

$$v_t^* = v_t(\mathbb{P}_0 + \Lambda_0^*)$$

Now we obtain the following chain:

$$\begin{aligned}
v_t(\mathbb{P}_0) + v_t(\Lambda_0^*) &= v_t(\mathbb{P}_q) + \sum_{k=0}^{q-1} (v_t(\mathbb{P}_k - \mathbb{P}_{k+1})) + v_t(\Lambda_0^*) \\
&= v_t(\mathbb{P}_q) + \sum_{k=0}^{q-1} \Big(v_t(\Lambda_{k+1}^*) - v_t(\tilde{\Lambda}_k) \Big) + v_t(\Lambda_0^*) \\
&= v_t(\mathbb{P}_q) + \sum_{k=0}^{q-1} \Big(v_t(\Lambda_k^*) - v_t(\tilde{\Lambda}_k) \Big) + v_t(\Lambda_q^*) \\
&\leq v_t(\mathbb{P}_q) + \epsilon \cdot \sum_{k=0}^{q-1} (v_t(\mathbb{P}_k - \mathbb{P}_{k-1})) + v_t(\Lambda_q^*) \\
&= v_t(\mathbb{P}_q) + \epsilon \cdot (v_t(\mathbb{P}_{q-1}) - v_t(\mathbb{P}_{-1})) + v_t(\Lambda_q^*)
\end{aligned}$$

The second equality follows by applying (4.9) and the inequality follows from Lemma 4. Remember that $\mathbb{P}_{-1} := \emptyset$ and thus $v_t(\mathbb{P}_{-1}) = 0$. Observe that $v_t(\mathbb{P}_{q-1}) \leq v_t(\mathbb{P}_q)$. Note that, since in the residual network $G'_{\tilde{c}_q}(x_q)$ no s-z-path of length less than T exists, it follows that $v_t(\Lambda_q^*) = 0$. Also note that chain flows in the sets $\mathbb{P}_0$ and Λ_0^* are disjoint and thus $v_t(\mathbb{P}_0 + \Lambda_0^*) = v_t(\mathbb{P}_0) + v_t(\Lambda_0^*)$. Thus we obtain:

$$\begin{aligned}
v_t^* &= v_t(\mathbb{P}_0 + \Lambda_0^*) \\
&\leq v_t(\mathbb{P}_q) + \epsilon \cdot (v_t(\mathbb{P}_q)) \\
&= (1+\epsilon) \cdot v_t(\mathbb{P}_q).
\end{aligned}$$

□

4.5 Earliest Arrival Flows with Flow-Dependent Transit Times

For a more realistic model of evacuations it is necessary to take into account how much flow is using an arc. Everybody knows from everyday experience that one can drive fast if the street is empty but the more cars there are on the street the slower one has to drive. Thus the transit time of an arc depends on the flow on that arc.
In this chapter we first introduce two different models to model the flow-dependency of the transit times and discuss their advantages and drawbacks. Then we consider a generalization of time-expanded networks to model flow-dependent transit times. Next we look at the existence of flow-dependent

earliest arrival flows, which is in general not given and eventually have a look at an approximation for flow-dependent earliest arrival flows. We mainly present the results of Baumann and Köhler [3] and also use the results of Köhler et. al [8] when we explain the modified time-expanded graph.

4.5.1 Models for Flow-Dependent Transit Times

To model earliest arrival flows with flow-dependent transit times Baumann and Köhler [3] study two different models:

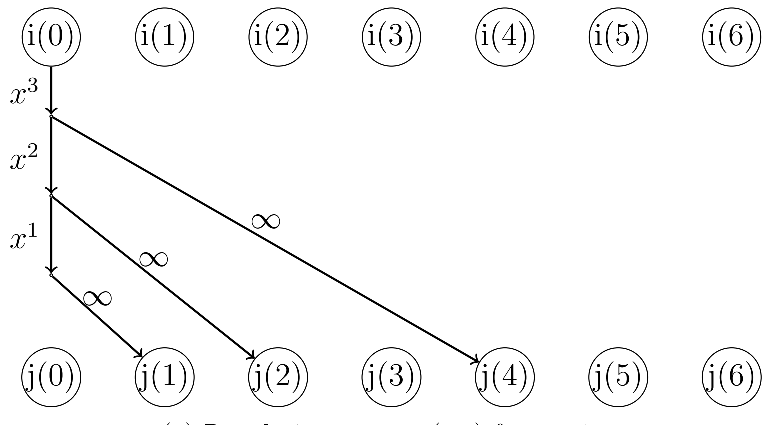
1. *Inflow-dependent transit times:* The transit time depends on the amount of flow that enters arc (i, j) at time t. $\tau((i, j), v(x_{ij}(t)))$ is the transit time on arc (i, j) that flow value $v(x_{ij}(t))$, which enters that arc at time t, needs to traverse it. In this model the speed of the flow units is determined when they enter an arc (i, j) and does not change while they are on (i, j). Thus, if at any time t a large amount of flow enters arc (i, j), a long transit time is assigned. If now at time $t + 1$ only a small amount of flow enters (i, j) a short transit time is assigned to these flow units. Therefore it might happen that the few units entering (i, j) at $t + 1$ overtake the large amount of flow units and leave (i, j) before them. Thus the first in - first out property does not hold in this model.

2. *Load-dependent transit times:* Here the transit times depend on the total amount of flow that is on arc (i, j), the *load* of the arc. Thus the transit time of an arc (i, j) changes at every time t with every flow unit that enters or leaves the arc. All flow units have the same speed on arc (i, j) at every moment. This guarantees the first in - first out property. But it is expensive to model and also not fully realistic: If a small amount of flow enters an arc there is no reason why these flow units should slow down when a large and slow amount of flow enters the arc after them.

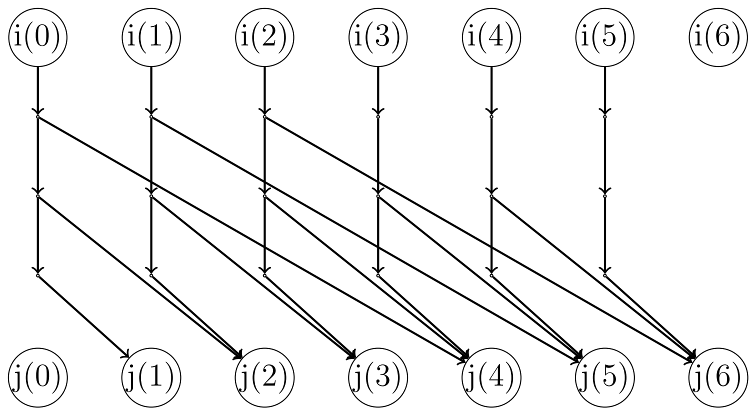
None of these models is able to model the full complexity of traffic flows, but they can model at least some aspects of the flow behaviour.

4.5.2 Fan Graphs and Bow Graphs

One problem with flow-dependent transit times is, that it is not possible to define a time-expanded graph as introduced by Ford and Fulkerson [11], since we only know how much transit time is needed on arc (i, j) when we know



(a) Regulating arcs on (i, j) for $t = 0$.



(b) Fan graph of (i, j) with $T = 6$.

Figure 4.13: Illustration of fan graphs.

how much flow uses (i, j).
In 2000 Carey and Subrahmanian [7] introduced a generalized time-expanded network for flow-depended transit times, where the flow-dependent transit times are modelled with help of special capacity constraints. Thus the static flow problem on this generalized time-expanded graph cannot be tackled with standard static network flow techniques.
For the inflow-dependent model Köhler et al. [8] introduced so-called fan graphs in 2002, which modify the standard time-expanded network by adding regulating arcs. This means that the transit times are modelled by the structure of the fan graphs and not by additional constraints and thus standard static network flow techniques apply to fan graphs. In the following we explain with help of an example how fan graphs are created and evaluate them.

Assume the transit time function of an arc (i, j) is given as piecewise constant and non-decreasing with only integral values. The fan graph has the same nodes as the time-expanded graph. To understand the concept of regulating arcs look at Figure 4.13 where these are modelled for one arc (i,j). Assume that the inflow-dependent transit time function for (i,j) is given as follows:

$$\tau((i,j),v(x_{ij}(t))) = \begin{cases} 1 & \text{, if } x_{ij}(t) \in [0,x^1] \\ 2 & \text{, if } x_{ij}(t) \in (x^1,x^2] \\ 4 & \text{, if } x_{ij}(t) \in (x^2,x^3 := c_{ij}] \end{cases}$$

The vertical arcs try to model the distribution of the flow according to the transit time function. These arcs are called regulating arcs, since they regulate how much flow can travel with the fastest speed (in out example at most x^1 flow units can travel with speed 1), how many flow units can travel with second fastest or fastest speed (in out example at most x^2 flow units can traverse the arc within time 3 or less), etc. The diagonal arcs have infinite capacity and represent the different transit times. If we repeat the structure shown in Figure 4.13 for every point of time we get a fan graph.
One problem of the fan graph is that if we send a certain amount of flow via an arc (i,j) then some units of this flow might travel faster than others: In Figure 4.13 if we send x^3 units of flow from i to j, then x^1 of these units only need time 1, $x^2 - x^1$ flow units need time 2 and only $x^3 - x^2$ units of flow need time 4. But in the inflow-dependent model all flow units x^3 should have the same speed and should need 4 time units to reach j. Thus the fan graph does not fully meet the model. It is possible to apply static network flow theory to the fan graph. But the main drawback of fan graphs is their size which makes static flow approaches on this graph rather awkward.

The time-expanded graph always corresponds to an underlying dynamic network and, as we have seen in Chapter 3, working on this underlying network yields good solution methods for some problems (e.g. the maximal dynamic flow problem). Thus it might be a good idea to find the underlying graph of a fan graph. Again we consider the inflow-dependent transit time functions to be piecewise constant and non-decreasing. The bow graph to the fan graph given in Figure 4.13 is shown in Figure 4.14. The bow graph also exists of regulating arcs, which model the inflow-rates and of bow arcs which model the transit times. The bow graph is considerably smaller than the fan graph, but the other drawback also holds for bow graphs: the units that enter an arc (i,j) are not necessarily travelling with the same speed as some units will use the faster arcs. Thus bow graphs do not fully meet the inflow-dependent transit time model.

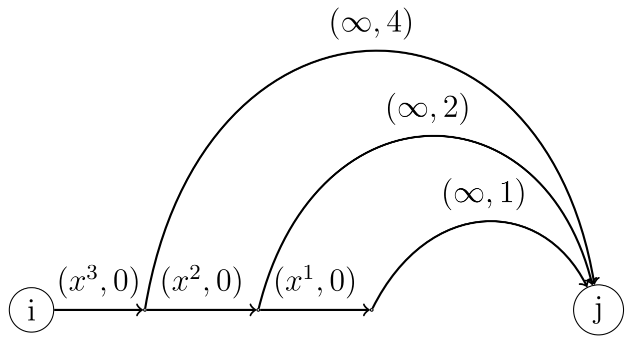


Figure 4.14: Bow graph of arc (i,j).

Köhler et al. [8] showed that every dynamic flow in the original graph G can be viewed as a dynamic flow in the bow graph. Thus the bow graph is a relaxation of the original inflow-dependent transit time model.

4.5.3 Existence of Earliest Arrival Flows with Flow-Dependent Transit Times

For flow-dependent transit times maximum dynamic flows exist, but finding a maximum dynamic flow is NP-hard (see [8],[3]). The existence of earliest arrival flows is given for none of the above models.

Theorem 8 (Baumann, Köhler,[3]) *For inflow-dependent transit times, earliest arrival flows do not exist in general.*

Proof: We prove this statement by counterexample:

Assume that for inflow-dependent transit times, earliest arrival flows exist for every instance.

Consider the following very easy instance: The dynamic network G consists only of source node s and sink node z and one arc (s,z), where $c_{sz} = 2$ and the transit time on (s,z) is given as $\tau_{sz} = 1$, if at most 1 unit of flow enters the arc and $\tau_{sz} = 2$, if 2 units of flow enter the (s,z). In the maximal dynamic flow for $T = 3$, 2 units of flow enter arc (s,z) at time $t = 0$ which arrive at z at time $t = 2$, another 2 units of flow enter (s,z) at time $t = 1$ and leave it at $t = 3$ and eventually 1 unit of flow enters (s,z) at $t = 2$ and leaves it at $t = 3$. Thus in total 5 units of flow arrive at z.

For an earliest arrival flow, we must have that the flow reaching the sink at $t = 1$ and $t = 2$ is also maximal. At $t = 1$ at most 1 unit of

flow could enter the sink, if it enters (s, z) at $t = 0$. Thus if we want to have an earliest arrival flow on our network G we must send only 1 unit of flow on (s, z) at time $t = 0$. But then only 4 flow units can reach the sink up to time $t = 3$, which is not maximal. We see that it is not possible to create an earliest arrival flow on this instance.
$\Rightarrow$ The assumption is wrong and earliest arrival flows do not exist in general for inflow-dependent transit times. $\square$

Theorem 9 (Baumann, Köhler, [3]) *For load-dependent transit times, earliest arrival flows do not exist in general.*

PROOF: Again we prove by counterexample:
Assume that earliest arrival flows exist for every instance with load-dependent transit times.
Consider the dynamic graph G consisting only of source s and sink z and one arc (s, z) with capacity $c_{sz} = 1$. The transit time for (s, z) is given as follows: If at most 1 unit of flow is on the arc, it needs 3 time units to reach the sink. If 2, 3, 4 or 5 flow units are on the arc, they need 6 time units to reach the sink and if more than 6 flow units are on the arc the transit time is infinity, which models that the arc has a maximal load of 6 flow units.
Let $T = 8$ be the time horizon and consider the maximal amount of flow that can reach the sink within this time. We can send one unit of flow on (s, z) at time $t = 0$, which has travelled at time $t = 1$ exactly one third of the way. Then at time $t = 1$ we let the next unit of flow enter (s, z) and thus the travel time increases for all flow units on (s, z) to 6. Thus at time $t = 2$ the first unit has made already 1/2 of the way and the second unit only 1/6 of the way. At $t = 2$ another flow unit enters (s, z), but does not affect the transit time. Thus at $t = 5$ the first unit of flow reaches z. The second unit of flow enters z at $t = 7$ and the last unit of flow has also reached z by $t = 8$. Thus the maximal amount of flow that can be sent through this network G within time $T = 8$ is 3.
Again for an earliest arrival flow we must have that at every point of time t, $0 \leq t \leq T$ the amount of flow that reaches the sink is maximal. Consider the sink at time $t = 3$. If we send one unit on (s, z) at time $t = 0$ and *no* flow unit on the arc at time $t = 1$ and $t = 2$, the first unit needs transit time 3 and thus reaches the sink at time $t = 3$. Therefore in an earliest arrival flow 1 unit of flow should reach the sink at time $t = 3$. But then we can only send 1 more unit of flow on the arc (s, z):

If we let 1 flow unit enter (s, z) at time $t = 3$, it reaches z at time $t = 6$ and there is not enough time to start another flow unit at $t = 6$. Also if we would start one flow unit in $t = 3$ and another one in $t = 4$, the first one would reach z at $t = 8$, but the second one could only reach the sink by $t = 9$ which is too late.
$\Rightarrow$ The assumption is wrong, since for the given instance above no earliest arrival flow for $T = 8$ can exist. □

4.5.4 Approximation of Flow-Dependent Earliest Arrival Flows

Since existence of flow-dependent earliest arrival flows is in general not given, approximation of such flows are of special interest. Baumann and Köhler [3] relax the time up to when a certain amount of flow has reached the sink, i.e. they allow a lateness of the flow. The problem to minimize this lateness is called α-earliest arrival flow problem.

Definition 10 *The α-earliest arrival flow problem asks the following question: Find the minimum α, such that there exists a feasible dynamic flow x that sends for each t, $0 \leq t \leq T$, at least as much flow into the sink z as can be send into z up to time t/α by the maximal flow x_{max}, i.e. $v_t(x) \geq v_{t/\alpha}(x_{max})$.*

Baumann and Köhler show that for dynamic flows with flow-dependent transit times there always exists a 4-earliest arrival flow. Furthermore they give lower bounds on α for the two models. They show

- for inflow-dependent transit times, there are instances that have no α-earliest arrival flow for $\alpha \leq 3/2 - \epsilon$, for all $\epsilon > 0$.
- for load dependent transit times, there exist instances that have no α-earliest arrival flow for $\alpha \leq 5/4$.

Finally they modify an approximation scheme for the quickest flow problem in the inflow-dependent model to approximate an α-earliest arrival flow. Since quickest flows are not subject of this thesis we do not present this algorithm but refer the interested reader to [3] and [16].

4.6 Further Literature on Earliest Arrival Flows

It is also possible to discuss earliest arrival flows with time-dependent transit times and capacities. Considering evacuations, the dependence of transit

times and capacities on the time, corresponds to streets or hallways that are blocked completely or partially after some time. Such a blockage might occur as example, because of the smoke in case of fire or the rising of the water level in case of a flood. Existence of earliest arrival flows with time-dependent transit times or capacities is given by Gale [14]. To compute an earliest arrival flow with time-dependent attributes, it is possible to use the traditional time-expanded graph. The varying transit times or capacities can be modelled directly in the time-expanded graph, according to the function that describes these attributes with respect to the time.
For earliest arrival flows with time-dependent transit times, Tjandra [27] gives a pseudo polynomial time algorithm, that runs in $O(nm^2T^3U)$ time, where U is the maximum of all capacities. Tjandra [27] uses successive earliest arrival augmenting paths on the residual network of the time-expanded graph. For every node he finds the earliest possible arrival time, considering the changing transit times. For an extensive examination of time-dependent dynamic flows we refer the reader to the PhD thesis "Dynamic Network Optimization with Application to the Evacuation Problem" by Tjandra [27].

In 2002, Fleischer and Skutella [10] gave another fully polynomial time approximation scheme for earliest arrival flows. They allow the flow that should arrive at the sink by time t, to be an earliest arrival flow, to arrive at the sink with a certain lateness, i.e. the flow must have reached the sink by time $(1 + \epsilon)t$. They develop condensed time-expanded networks to get this approximation.

Baumann and Skutella [4] consider a modified earliest arrival flow problem. They examine networks with several sources and sinks, where to each source and each sink a certain supply and demand, resp., is assigned. A feasible dynamic flow on this problem must satisfy these supplies and demands as additional constraint. Such earliest arrival flows with supplies and demands do not exist for several sinks, but only for the multiple sources - single sink case. Baumann and Skutella [4] give an algorithm for the multiple sources - single sink case the running time of which is polynomially bounded in the input plus output size of the modified earliest arrival flow problem. We did not discuss this problem in detail, since it is not the original problem and the questions and difficulties arising in this modified problem are not the same as the in the original problem.

As already stated, it is still unknown, whether the earliest arrival flow problem is NP-hard or not. Anyhow, in 1984 Hajek and Ogier [15] presented the first polynomial time algorithm for the special case of earliest arrival flows with zero transit times.

For continuous time, Ogier [26] gave the first polynomial time algorithm for the earliest arrival flow problem with zero transit times and piecewise-constant integer capacities in 1988. In 2001 Fleischer [9] showed that the problem Ogier [26] considered can be reduced to a generalized parametric maximum flow problem. With this knowledge she develops a faster algorithm than Ogier for this problem.

Surveys on dynamic flows which include the earliest arrival flow problem, are for example "Dynamic flows in networks", by Lovetskii and Melamed [24] or "An annotated overview of dynamic network flows", by Kotnyek [23]. For an State of Art of mathematical modelling of evacuation problems we refer the interested reader to [18].

Chapter 5

Maximal Dynamic and Earliest Arrival Flows on Series-Parallel Graphs

Until today it is still an open question whether the earliest arrival flow problem is NP-hard or if it is possible to find an exact polynomial algorithm. In Chapter 4 we presented the polynomial time approximation algorithm of Hoppe and Tardos [19]. In this chapter we consider a special class of earliest arrival flows, namely earliest arrival flows on series-parallel graphs.
First we introduce series-parallel graphs. Next we present the minimum cost flow algorithm of Bein and Brucker [5]. Following the main idea of Ford and Fulkerson [11], we assemble the algorithm of Bein and Brucker and the minimum cost circulation problem algorithm, presented in Chapter 3, to develop a new algorithm that solves the maximal dynamic flow problem on series-parallel graphs in time $O(mn + m \log m)$.
We show that this maximal dynamic flow algorithm also solves the earliest arrival flow problem on series-parallel graphs. Thus we have found a polynomial time algorithm for a special case of the earliest arrival flow problem. Then we discuss an implementation approach for this algorithm and finally have a short look on earliest arrival flows on series-parallel graphs with inflow-dependent transit times.

5.1 Definition of Series-Parallel Graphs and Basics

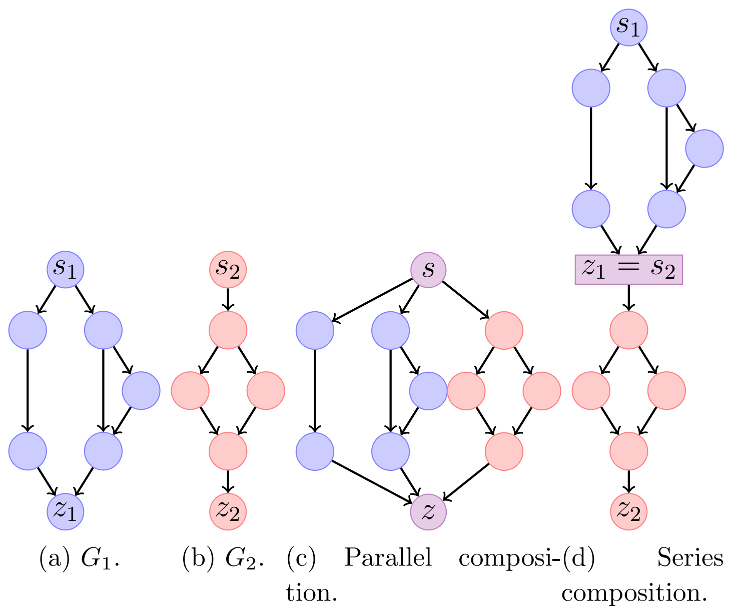
Definition 11 *A (two terminal)* series-parallel graph *$G = (N, A)$ is a directed graph with one source s and one sink z with the following property: G is defined recursively by the rules given below:*

1. *K_2, i.e. a single arc (s, z) together with its nodes s and z, is a series-parallel graph.*
2. *Let G_1 and G_2 be two series-parallel graphs. Then the graph G obtained by one of the following operations is also series-parallel:*
   - *(a)* parallel composition: *Merge the source nodes s_1 of G_1 and s_2 of G_2 to the source s of G. Merge the sink nodes z_1 of G_1 and z_2 of G_2 to the sink z of G.*
   - *(b)* series composition: *Identify the sink z_1 of G_1 with the source s_2 of G_2.*

Example 13 *For a better understanding of the definition of series-parallel graphs, we illustrate the parallel and the series composition in Figure 5.1 It is also useful to know what is* not *a series-parallel graph. Thus the smallest graph that is not a series-parallel graph is given in Figure 5.2*

The construction process of a series-parallel graph can be represented in a binary tree, called *decomposition tree.* Following the recursive definition we denote series compositions by nodes labeled S and parallel compositions by nodes labeled P. We assume without loss of generality that the decomposition tree has $1, \ldots, r$ nodes which are enumerated topologically. This means that the number assigned to a father node is larger than the number assigned to any of its sons. Every leaf b of the decomposition tree is a set of arcs $E(b)$ representing one or several arcs of the underlying series-parallel graph G, where each arc of one set $E(b)$ has the same starting and end node in G, i.e. is of the form (i, j). We show how to get such a decomposition tree in the following example:

Example 14 *Consider again graph G_2, where the arcs are labeled as in Figure 5.3. The topological enumeration is given by the numbers above every node of the decomposition tree.*



(a) G_1. (b) G_2. (c) Parallel composition. (d) Series composition.

Figure 5.1: Illustration of series and parallel composition of series-parallel graphs G_1 and G_2.

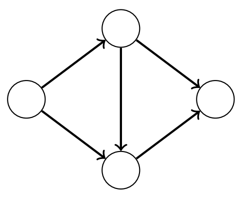
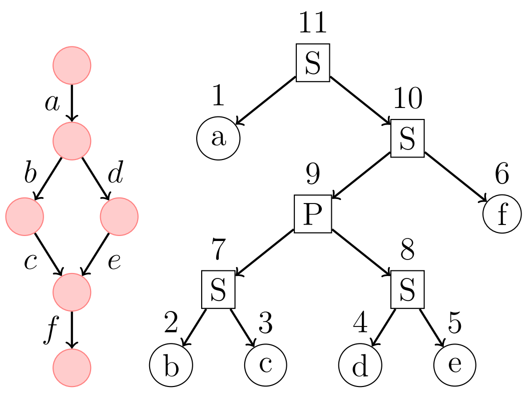


Figure 5.2: Not series-parallel.



(a) G_2 (b) topologically enumerated decomposition tree of G_2

Figure 5.3: Decomposition tree.

Every series-parallel graph can be represented by such a binary decomposition tree. We will make use of this representation in the implementation of our algorithm.
Until now we only took into consideration the structure of the graph, but did not assume any transit times or capacities on the arcs.

5.2 Minimum Cost Flows on Series-Parallel Graphs

We follow the general idea of Ford and Fulkerson, presented in chapter 3 to develop a maximal dynamic flow algorithm that takes into account the special structure of series-parallel graphs. Thus we first analyse the minimum cost flow problem (3.1) introduced in Chapter 3 for series-parallel graphs, with integer capacities and costs.

Bein and Brucker [5] present the greedy Algorithm 7 which finds a solution of the minimum cost flow problem (3.1) on series-parallel graphs in time $O(nm + m \log m)$. Algorithm 7 finds the maximal flow value v_{max} and defines a piecewise linear and convex function f on the interval $[0, v_{max}]$ with $f(0) = 0$. Function f is completely described by a partition of $[0, v_{max}]$ into successive subintervals $I_k,\ k = 1, \dots, q$ of length l_k. For each interval I_k a corresponding path P_k from s to z exists, where the cost of one unit of flow along P_k is denoted by $a(P_k) := \sum_{(i,j)\in P_k} a_{ij}$. On each subinterval I_k function f has slope $a(P_k)$ which does not change. The sequence $a(P_1), \dots, a(P_q)$ of the slopes is non-decreasing. The length l_k of each interval I_k corresponds to the maximal amount of flow that can travel along path P_k, where the flow on the paths P_1 to P_{k-1} has to be observed.

Let $L_d := \sum_{k=0,\dots,d} l_k$ where $l_0 := 0$. Then f is defined recursively:

$$f(0) := 0 \tag{5.1}$$

$$f(v) := f(L_d) + a(P_{d+1}) \cdot (v - L_d) \text{ on interval } I_{d+1} := [L_d, L_d + l_{d+1}]\,. \tag{5.2}$$

A complete solution of $P(v)$ is given by function $f(v)$ which is characterized by these parameters:

$$v_{max} \text{ and } (l_k, a(P_k), P_k) \text{ for } k = 1, \dots, q.$$

Algorithm 7 Greedy Algorithm of Bein and Brucker [5]

```
*Input:* Series-parallel graph $G = (N, A)$ with one source $s$ and one sink $z$
with costs $a_{ij}$ and capacities $c_{ij} \in \mathbb{R}^+$ on each arc $(i, j) \in A$.
*Output:* Solution of $P(v)\ \forall\ v \in [0, v_{max}]$, by giving out $v_{max}$ and the
parameters $l_k, a(P_k)$ and $P_k$ for $k = 1, \dots, q$.
**for all** $(i, j) \in A$ **do**
    $x_{ij} := 0,\ k := 0$
**end for**
**while** there exists a path connecting $s$ and $z$ **do**
    $k := k + 1$
    Find a minimal cost path $P_k$ and the corresponding $a(P_k)$ value.
    $l_k := \min\{c_{ij} \,|(i, j) \in P_k\}$
    **for all** $(i, j) \in P_k$ **do**
        $c_{ij} := c_{ij} - l_k$
        **if** $c_{ij} = 0$ **then**
            then $A := A \setminus \{(i, j)\}$
        **end if**
    **end for**
**end while**
```

It is not obvious which is the best way to find a minimal cost path on series-parallel graphs. For the moment it is enough to know that we can find a minimal path under the given circumstances, as example one can use the Algorithm of Dijkstra (see for [17]). We show a clever way to find minimal cost paths on series-parallel graphs, when we discuss the implementation.
The important property of Algorithm 7 is, that it is an augmenting path algorithm which does not use backward arcs. So whenever Algorithm 7 sends flow over a path P_k, this flow will never be taken back again. This also implies, that the capacity of the cheapest path is completely used up, before flow is send along any more expensive path.

Example 15 *For a better understanding of Algorithm 7 we consider the series-parallel graph given in Figure 5.4. On the arcs (c_{ij}, a_{ij}) are given. Applying the algorithm of Bein and Brucker we get:*
First minimal cost path $P_1 = (s, 1, 2, z)$ with total costs $a(P_1) = 4$ and maximal flow value $l_1 = 5$.
Updating the capacities we get $c_{s1} := 5,\ c_{12} := 0$ and $c_{2z} := 1$. Consequently arc $(1, 2)$ is deleted. In the next iteration of the algorithm we get:

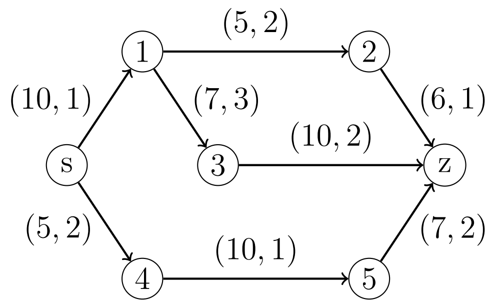


Figure 5.4: Series-parallel graph.

Second minimal cost path $P_2 = (s, 4, 5, z)$ with total costs $a(P_2) = 5$ and maximal flow value $l_2 = 5$. The update of the capacities yields $c_{s4} := 0$, $c_{45} := 5$ and $c_{5z} := 2$. Arc $(s, 4)$ is deleted and the next iteration gives:
Third minimal cost path $P_3 = (s, 1, 3, z)$ with total costs $a(P_3) = 6$ and maximal flow value $l_3 = 5$. The capacity update gives $c_{s1} = 0$, $c_{13} := 2$ and $c_{3z} := 5$ and arc $(s, 1)$ is deleted. Since there exists no longer a path from s to z the algorithm terminates.

The following Theorem 10 of Bein and Brucker [5] shows that Algorithm 7 solves the minimum cost flow problem on series-parallel graphs.

Theorem 10 (Bein and Brucker [5]) *Let G be a directed acyclic graph with a single source s and a single sink z. G is a (two-terminal) series-parallel graph if and only if for every set of costs $\{a_{ij}\}$ for all $(i, j) \in A$ and every set of nonnegative capacity $\{c_{ij}\}$, $(i, j) \in A$, Algorithm 7 solves the corresponding minimal cost flow problem $P(v)$ for $0 \leq v \leq v_{max}$.*

5.3 Maximal Dynamic Flows on Series-Parallel Graphs

To solve the maximal dynamic flow problem on series-parallel graphs we combine the algorithm of Bein and Brucker and the MCCP-Algorithm presented in Chapter 3.
We use again the notation introduced in the proceeding section. We also want to use Theorem 10 in the verification of the algorithm. As the costs correspond to the transit times in the dynamic case and we cannot travel backward in time, we do not allow negative costs. Also, since we only consider the time discretized case, all transit times are integer. Theorem 10

holds for the more general case and therefore also includes this special case.

Algorithm 8 MCCP Greedy Algorithm for Series-Parallel Graphs

```
*Input:* Series-parallel graph $G = (N, A, T)$ with costs $a_{ij} \in \mathbb{Z}^+$ and capacities $c_{ij} \in \mathbb{Z}^+$ for each arc $(i, j) \in A$.
Add an additional arc $(z, s)$ with infinite capacity and cost $-(T + 1)$.
*Output:* Minimum cost paths $P_k$ and the corresponding parameters $a(P_k)$ and $l_k$. Also output the last value $k^*$ of the counting variable $k$ for which a flow was sent via the shortest $s$-$z$ path $P_{k^*}$.
for all $(i, j) \in A$ do
  $x_{ij} := 0,\ k := 0$
end for
while there exists a path connecting $s$ and $z$ in $G$ do
  $k := k + 1$
  Find a minimum cost path $P_k$ and the corresponding $a(P_k)$ value.
  Expand $P_k$ to a circulation $C^k$ by directing the flow that reaches $z$ via $P_k$ to $s$ using arc $(z, s)$.
  if $a(P_k) - (T + 1) < 0$ then
    $l_k := \min \left\{ c_{ij} \,\middle|\, (i, j) \in C^k \right\}$
  else
    Stop the algorithm.
  end if
  for all $(i, j) \in C^k$ do
    $c_{ij} := c_{ij} - l_k$
    If $c_{ij} = 0$ then $A := A \setminus \{(i, j)\}$
  end for
end while
```

As Algorithm 8 does not search for negative dicycles, it is not clear that it really solves the maximal dynamic flow problem. Thus we need to prove the following theorem:

Theorem 11 *Algorithm 8 solves the maximal dynamic flow problem for series-parallel graphs $G = (N, A, T)$ with one source s and one sink z and with given capacities $c_{ij} \in \mathbb{Z}^+$ and transit times $\tau_{ij} \in \mathbb{Z}^+$ on each arc $(i, j) \in A$.*

For the proof of this theorem we need the following definition:

Definition 12 *Let $G = (N, A)$ be a series-parallel graph (with flow x and $G'(x)$ the corresponding residual network). G can be decomposed into its*

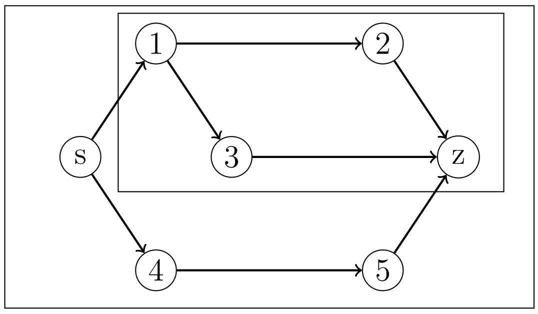


Figure 5.5: Parallel components

parallel components. Each of these components has two terminals: a source node u and a sink node w. We denote these parallel components by $U(u; w)$. Note that every path in the series-parallel graph G that uses node u also has to use node w.

Example 16 *To illustrate this definition, consider the series-parallel graph given in Figure 5.5. In this series-parallel graph we have two parallel components $U(s; z)$ and $U(1; z)$ which are circled.*

Proof: Regard the transit times τ_{ij} as costs and apply Algorithm 8 to the given series-parallel graph G. Due to the *if*-condition the algorithm stops as soon as the costs $a(P_k)$ of the minimum cost path P_k are larger than T. Or, if T is large enough to allow all necessary minimum cost paths, Algorithm 8 terminates when there are no more paths from s to z. At termination the algorithm has found the total amount of flow v^* that is sent from s to z within time horizon T. This amount of flow v^* is given by

$$v^* := \sum_{k=1}^{k^*} ((T+1) - a(P_k)) \cdot l_k$$

Let $\tilde{f}(v)$ denote the function described completely by the parameters P_k, $a(P_k)$, l_k, for $k = 1, \ldots, k^*$, and the maximal flow value v^* which are found during Algorithm 8.

Now apply the algorithm of Bein and Brucker to the given graph G, where the costs $a_{ij} := \tau_{ij}$ for all arcs $(i, j) \in A$. Algorithm 7 terminates when there exist no more paths between s and z and then outputs the parameters

$$v_{max} \text{ and } (l_k, a(P_k), P_k) \text{ for } k = 1, \ldots, q.$$

These parameters completely describe the solution function $f(v)$ for every flow value $0 \leq v \leq v_{max}$. Now compare $f(v)$ and $\tilde{f}(v)$ on the interval $[0, v^*]$. We observe that the paths P_k found by Algorithm 7 are exactly the same as the paths found by our Algorithm 8, since the algorithms do exactly the same. Additionally the costs $a(P_k)$ of each of these paths are the same and also the minimum capacities on each path are identical. Thus, the l_k values are also identical, $\forall\ k = 1, \ldots, k^*$. Since we restricted $f(v)$ to the interval $[0, v^*]$, the maximal flow values are also identical (due to the choice of v^*).
By Theorem 10 we know that Algorithm 7 solves the minimum cost flow problem on series-parallel graphs. Since the functions $f(v)$, which gives the solution of $P(v)$ for all $v \in [0, v_{max}]$, and $\tilde{f}(v)$ coincide on $[0, v^*]$, we conclude that Algorithm 8 solves the minimum cost flow problem $P(v^*)$ on the series-parallel graph G (note that arc (z, s) is not in G).
Next we show that at termination of Algorithm 8 there are no negative circulations possible. To do this we look at the residual network of $G \cup \{(z, s)\}$, with associated flow x found by Algorithm 8, where all the arcs that have been deleted during Algorithm 8 are represented by a backward arc only.

Case 1: There exists a negative circulation using only arcs of G.
This is a contradiction to the fact that the flow on G is a minimum cost flow.

Case 2: A circulation C uses arc (z, s).
Denote the path of C through the original graph G by $P := C \setminus \{(z, s)\}$. If Algorithm 8 stops because there are no more s-z paths, then obviously P cannot exist and therefore also C. Else, if the algorithm stops, because the last s-z path it finds is too expensive, then we know that P was not found by Algorithm 8. For otherwise at least one arc would be a backward arc, i.e. only point from z to s. So the cost $a(P)$ of this path P must be larger than T. $\Rightarrow$ The circulation has non-negative costs, since $a(P) - T > T - T = 0$.

Case 3: A circulation uses arc (s, z).
This means that the circulation C uses backward arcs in $G'(x)$ to travel from z to s. If arc (i, j) has a backward arc in the residual network $G'(x)$, then flow x has sent some units of flow via arc (i, j). We want to show that the costs of this circulation C are nonnegative. Thus we need to show that even if the costs are

as negative as possible , they are still greater than or equal to $-(T+1)$, i.e. $a(P) \geq -(T+1)$.
First we show that, without loss of generality, path P consists only of backward arcs:
Assume that P includes a negative cycle. This is a contradiction to the fact that the flow x solves the minimum cost flow problem for v^* on G. Thus, if P includes any cycles, the cost of these cycles have to be non-negative, and therefore P is more negative, if we omit these cycles. Thus we may assume without loss of generality that P does not inclue any cyles and consists only of backward arcs.
It might happen that P consists of the arcs of several paths $P_1, \ldots, P_{k^*}$ which are the backward paths in the residual network $G'(x)$ where each corresponds to a forward path found by Algorithm 8. The indices of $P_1, \ldots, P_{k^*}$ should correspond to the order in which Algorithm 8 found the corresponding forward paths, i.e. $a(P_1) \geq a(P_2) \geq \ldots \geq a(P_{k^*})$, since the costs of these backward paths are negative.
Let P_d, $1 \leq d \leq k^*$ be the path with the highest index of which P uses arcs. Thus the costs of P_d are more negative than those of any other path which shares arcs with P, but still $a(P_d) > -(T+1)$, since Algorithm 8 found the corresponding forward path of P_d.
Now consider only those parts of P where P and P_d differ:
Let (u,v) be the first arc of P which is not used by P_d. Then node u must be a terminal node of a parallel component $U(u;w)$. Since P and P_d use different ways to travel through $U(u;w)$, let P_{d_1} be the path with the largest index $d_1 < d$ of which P uses arcs in $U(u;w)$. Since Algorithm 8 is a greedy algorithm we know that

$$a(P_{d_1}\big|_{U(u;w)}) \geq a(P_d\big|_{U(u;w)}).$$

If P and P_{d_1} do not coincide on $U(u;w)$ then we can repeat the argument until P and some P_{d_l} coincide on a parallel component $U(u_l;w_l) \subset U(u;w)$.
Thus we get the following estimation:

$$\begin{aligned}
a(P) =& a(P\big|_{U(s;u_1)}) + a(P\big|_{U(u_1;u_2)}) + \ldots + a(P\big|_{U(u_w;z)}) \\
\geq& a(P_d\big|_{U(s;u_1)}) + a(P_d\big|_{U(u_1;u_2)}) + \ldots + a(P_d\big|_{U(u_w;z)}) \\
=& a(P_d) > -(T+1)
\end{aligned}$$

Thus such a circulation $C = P \cup \{(s,z)\}$ has costs

$$\begin{aligned} a(C) = & \quad a(P) + a(\{(s,z)\}) \\ \geq & \quad a(P_d) + a(\{(s,z)\} \\ > & \quad -(T+1) + T + 1 = 0 \end{aligned}$$

□

Observe that the validity of Algorithm 8 implies that it never happens that flow has to be taken back on any arc during the algorithm. Thus at termination of Algorithm 8 all the cheapest paths are completely exhausted whereas the too expensive paths, i.e. those with $\sum_{(i,j)\in P} \tau_{ij} > T$, are not used at all. The paths P_k found in the algorithm together with the corresponding flow values l_k are essentially temporally repeated flows.

5.4 Earliest Arrival Flows on Series-Parallel Graphs

Remember that we showed in Chapter 4 that maximal dynamic flows found by applying the minimum cost circulation problem algorithm are not necessarily earliest arrival flows. For series-parallel graphs we show in this section that the maximal dynamic flows found by our Algorithm 8 are always earliest arrival flows. Thus we have solved both problems with one algorithm.

Theorem 12 *Let $G = (N, A, T)$ be a series parallel graph. Let x be the maximal dynamic flow (MDF) on G obtained by applying Algorithm 8. Then x is also an earliest arrival flow (EAF).*

PROOF: Assume that the theorem is wrong, that is the maximal dynamic flow x is not an earliest arrival flow.
On the corresponding time expanded network $D(T)$ of G the MDF x has arrival pattern $\omega(z)$ for all sinks $z \in Z$. Let $\omega^*(z)$ denote the earliest arrival pattern of the sinks. Since we assume that x is not an EAF it follows, that $\omega(z) \neq \omega^*(z)$ for at least two sinks $z \in Z$. According to Theorem 4 of Minieka, there exists a path P_{early} starting at one sink $z(t')$ and going to another sink $z(t)$ with $t < t'$. Since we assume that the MDF x is not an earliest arrival flow on G, x does not

deliver as many flow units into the "earlier" $z(t)$ as the earliest arrival flow requires there:

$$\omega^*(z(t)) > \omega(z(t)) \text{ and } \omega^*(z(t')) < \omega(z(t')) \text{ with } t < t'.$$

This path P_{early} uses arcs of the residual network $D'(T)(x)$ of the time-expanded network.
We want to use the argument of the parallel components $U(u;w)$ which we introduced in Definition 12. Since our argumentation takes place on the residual network $D'(T)(x)$ of the time-expanded graph, we have to reformulate the definition to apply it to this network:

Definition 13 *Let $G = (N, A, T)$ be a series-parallel graph (with flow x and $G'(x)$ the corresponding residual network). Let $D'(T)(x)$ be the residual network of the time expanded network of G with associated flow x. Remember that G can be decomposed into its parallel components and each of these components has two terminals: a source node u and a sink node w. We denote the corresponding structure to such a parallel component $U(u;w)$ in $D'(T)(x)$ by $U'(u;w)$ and call it $U(u;w)$-box. Such a $U(u;w)$-box contains all copies of the nodes of the parallel component $U(u;w)$ including the terminal nodes u and w and the copies of all arcs of the residual network between any of the nodes of $U(u;w)$.*

Observation 5 *One U-box might contain several smaller U-boxes and it might happen that two different U-boxes have a terminal node in common. But if a U-box $U'(u_1;w_1)$ contains a node of another U-box $U'(u_2;w_2)$ which is not a terminal node of $U'(u_1;w_1)$, then $U'(u_2;w_2) \subset U'(u_1;w_1)$.*

Now we consider path P_{early} in detail: P_{early} uses backward and forward arcs in $D'(T)(x)$ to connect $z(t')$ to $z(t)$. We call those nodes at which P_{early} changes the arc direction a *turning node*, e.g. P_{early} reaches a turning node by using a backward arc and leaves it by using a forward arc of the residual network $D'(T)(x)$. We have at least one such turning node, since otherwise it is impossible for P_{early} to reach sink node $z(t)$. If P_{early} includes more than one turning node, we must have a *loop* and the number of turning points must be odd. A *loop* means, that this part of P_{early} corresponds to a cycle in the residual network $G'(x)$ of the original graph. Each loop contains two turning nodes which are both terminal nodes of a U-box.
Let L^1 be the smallest loop that P_{early} contains and let u_1 and w_1 be the

turning nodes in L^1. Then this loop L^1 only uses arcs of the $U(u_1;w_1)$-box. Thus it is sufficient to consider only this U-box. Assume without loss of generality that the loop L^1 starts in $w_1(t_1)$ and leaves this node by using backward arcs, reaches $u_1(t_2)$, and leaves $u_1(t_2)$ by using forward arcs. Finally the loop reaches $w_1(t_3)$ via these forward arcs, where t_1 and t_3 might differ. P_{early} does not use another loop within L^1, since otherwise L^1 would not be the smallest loop.
Since P_{early} uses backward and forward arcs in the $U(u_1;w_1)$-box, Algorithm 8 has sent flow from u_1 to w_1 in the original graph G, but did not fully exhaust the capacities of all possible u_1-w_1-paths. Let $P_1,\dots,P_k$ be the paths found by Algorithm 8 that travel via nodes u_1 and w_1 in G, where the indices correspond to the order in which the paths were found.
Let $P_{early}(w_1(t_1);u_1(t_2))$ denote the backward part, which only uses backward arcs, and $P_{early}(u_1(t_2);w_1(t_3))$ the forward part, which only uses forward arcs, of P_{early} in L^1. We want to estimate the travel time of P_{early} on L^1, where we first consider the backward part and then the forward part:

$P_{early}(w_1(t_1);u_1(t_2))$: Here P_{early} only uses backward arcs. This means that the flow x, which the Algorithm 8 found, has sent flow over these arcs and therefore a backward arc exists in the residual network.
Claim: The travel time of $P_{early}(w_1(t_1);u_1(t_2))$ is not less than the backward travel time of $P_k\big|_{U'(u_1;w_1)}$, i.e.

$$\tau(P_{early}(w_1(t_1);u_1(t_2))) \geq -\tau(P_k\big|_{U'(u_1;w_1)}).$$

Note that the travel time of $P_{early}(w_1(t_1);u_1(t_2))$ is negative, since it uses backward arcs. So the backward travel time of $P_k\big|_{U'(u_1;w_1)}$ must also be negative.
Proof of Claim: If $P_{early}(w_1(t_1);u_1(t_2)) = P_k\big|_{U'(u_1;w_1)}$, then the travel times are identical and the claim holds. So consider those parts of $P_{early}(w_1(t_1);u_1(t_2))$ where $P_{early}(w_1(t_1);u_1(t_2))$ and the time-expanded graph correspondents $\tilde{P}_k$ of $P_k\big|_{U'(u_1;w_1)}$ differ.
$P_{early}(w_1(t_1);u_1(t_2))$ might use several correspondents $\tilde{P}_k$ of path $P_k\big|_{U'(u_1;w_1)}$. They can only differ on a $U(u_2;w_2)$-box, for which $U'(u_2;w_2) \subset U'(u_1;w_1)$. Let W_l be the u_2-w_2-way correspondent which $\tilde{P}_k$ uses in G. Then all the other u_2-w_2-ways $W_1,\dots,W_{l-1}$ which carry flow must be exhausted, since otherwise $\tilde{P}_k$ would

have used one of these ways, since they are faster. (They are faster, since the other paths $P_1, \dots, P_{k-1}$ use these ways and the greedy Algorithm 8 found them before P_k.) The other u_2-w_2-ways $W_{l+1}, W_{l+2}, \dots$, which do not carry flow, have no backward arcs in the residual network. Thus $P_{early}(w_1(t_1); u_1(t_2))$ must use a faster way than $\tilde{P}_k$ if they differ.

$$\Rightarrow \tau(P_{early}(w_1(t_1); u_1(t_2)) \left|_{U'(u_2;w_2)}\right.) \geq -\tau(P_k \left|_{U'(u_2;w_2)}\right.)$$
$$\Rightarrow \tau(P_{early}(w_1(t_1); u_1(t_2))) \geq -\tau(P_k \left|_{U'(u_1;w_1)}\right.)$$

$P_{early}(u_1(t_2); w_1(t_3))$: Here P_{early} only uses forward arcs. This means that P_{early} can only use arcs the capacity of which is not exhausted by the flow x the Algorithm 8 found.

Claim: The travel time of $P_{early}(u_1(t_2); w_1(t_3))$ is not less than the travel time of $P_k \left|_{U'(u_1;w_1)}\right.$, i.e.

$$\tau(P_{early}(u_1(t_2); w_1(t_3))) \geq \tau(P_k \left|_{U'(u_1;w_1)}\right.).$$

Proof of Claim: If $P_{early}(u_1(t_2); w_1(t_3)) = P_k \left|_{U'(u_1;w_1)}\right.$, then the travel times are identical and the claim holds. So consider those parts of $P_{early}(u_1(t_2); w_1(t_3))$ where $P_{early}(u_1(t_2); w_1(t_3))$ and the time expanded graph correspondents $\tilde{P}_k$ of $P_k \left|_{U'(u_1;w_1)}\right.$ differ. They can only differ on a $U(u_3; w_3)$-box, where $U'(u_3; w_3) \subset U'(u_1; w_1)$. Let W_l^* be the u_3-w_3-way correspondent in G which $\tilde{P}_k$ uses. Then all the other, shorter u_3-w_3 ways $W_1^*, \dots, W_{l-1}^*$ which have flow on them must be exhausted, since otherwise, $\tilde{P}_k$ would use one of these faster ways. All these ways $W_1^*, \dots, W_{l-1}^*$ include at least one arc that has no forward arc in the residual network. So P_{early} cannot use any of these ways, but has to take a u_3-w_3-way which the Algorithm 8 did not use. Thus the travel time from u_3 to w_3 of P_{early} must be larger than that of P_k, if they differ.

$$\Rightarrow \tau(P_{early}(u_1(t_2); w_1(t_3)) \left|_{U'(u_3;w_3)}\right.) \geq \tau(P_k \left|_{U'(u_3;w_3)}\right.)$$
$$\Rightarrow \tau(P_{early}(u_1(t_2); w_1(t_3))) \geq \tau(P_k \left|_{U'(u_1;w_1)}\right.)$$

Thus we get the following estimation for the travel time of P_{early} on the loop L^1:

$$\begin{aligned}
\tau(P_{early} \left|_{L^1}\right.) &= \tau(P_{early}(w_1(t_1); u_1(t_2))) + \tau(P_{early}(u_1(t_2); w_1(t_3))) \\
&\geq -\tau(P_k \left|_{U'(u_1;w_1)}\right.) + \tau(P_k \left|_{U'(u_1;w_1)}\right.) \\
&= 0
\end{aligned}$$

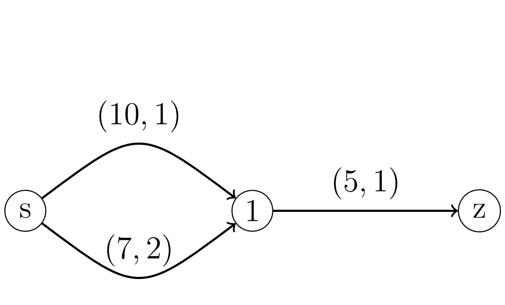
This means, that P_{early} only looses time by using the loop L^1. Thus we assume that P_{early} does not include such a loop L^1 and set $P_{early} := P_{early} - L^1$.
If this new P_{early} still contains other loops, we iterate the above argumentation: We can again find a smallest loop which does not contain any other loops. By the same argumentation as above we can eliminate this loop, because P_{early} only looses time by using the loop.
Thus we may assume that P_{early} contains no loops at all. Then P_{early} consists only of backward arcs from $z(t')$ to the turning node and from the turning node to $z(t)$ it consists only of forward arcs. So P_{early} itself is a loop. Thus by the same argumentation as above we get that $\tau(P_{early}) \geq 0$.
$\Rightarrow$ We cannot get such a path P_{early} starting in $z(t')$ and ending in $z(t)$ with $t < t'$. Thus, since no such "earlifying" path P_{early} exists, it follows that the arrival patterns $\omega(z)$ and $\omega^*(z)$ must be the same for all sinks $z \in Z$. Hence the assumption is wrong and the maximal dynamic flow x found by Algorithm 8 is already an earliest arrival flow.
□

We have seen in Chapter 4, that for general graphs, it is not always possible the find an earliest arrival flow which has the property that it is also a temporally repeated flow. As a simple consequence of Theorem 12 we get the following observation:

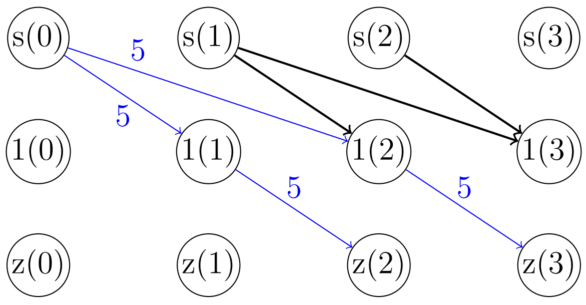
Observation 6 *On series-parallel graphs it is always possible to find for every time horizon $T \geq 0$ an earliest arrival flow, that is a temporally repeated flow.*

The stronger statement that on series-parallel graphs all earliest arrival flows are temporally repeated flows is not true, as Figure 5.6 illustrates. Thus we have the following observation:

Observation 7 *On series-parallel graphs not every earliest arrival flow is a temporally repeated flow.*



(a) Series-parallel graph



(b) An earliest arrival flow for $T = 3$.

Figure 5.6: Not every earliest arrival flow on a series-parallel graph has the temporally repeated flow property

5.5 Implementation and Complexity Analysis

5.5.1 Implementation

Bein and Brucker [5] present a bottom-up procedure for finding a minimal cost path from s to z in a series-parallel graph G. This procedure makes use of the special structure of series-parallel graphs by using the corresponding decomposition tree. Remember that we enumerated the $1, \dots, r$ nodes of the decomposition tree topologically. The minimal cost path P and its cost-value $a(P)$ can be calculated by using Algorithm 9.

Algorithm 9 Bottom-up Procedure of Bein and Brucker [5]

```
for $b := 1$ until $r$ do
  if $b$ is a leaf then
    INITIALIZE($b$)
  else
    Find the left son $c$ and the right son $d$ of $b$
    if $b$ has label $P$ (corresponding to a parallel composition) then
      MERGE($c, d; b$)
    else
      ADD($c, d; b$)
    end if
  end if
end for
```

Algorithm 9 starts at the leaves and processes upwards through the tree

until it reaches the root r.
The three procedures INITIALIZE(b), MERGE($c, d; b$) and ADD($c, d; b$) that are needed in Algorithm 9 are defined as follows:

```
INITIALIZE($b$)
1. **If** set $E(b)$ contains several arcs $e', e'', \dots$
2.     Choose arc $e$ with the minimum cost
3.     $P_b := e$
4.     $a(E(b)) := a(e)$
5. **Else** $E(b) = \emptyset$
6.     $P_b := \emptyset$
7.     $a(E(b)) := \infty$
```

```
MERGE($c, d; b$)
1. **If** $a(c) \leq a(d)$ **then**
2.     $a(b) := a(c)$
3.     $P_b := P_c$
4. **Else**
5.     $a(b) := a(d)$
6.     $P_b := P_d$
```

```
ADD($c, d; b$)
1. $a(b) := a(c) + a(d)$
2. $P_b := P_c \circ P_d$
```

In procedure ADD($c, d; b$) the expression $P_c \circ P_d$ denotes the concatenation of P_c and P_d, where $P_c \circ P_d = \emptyset$ if P_c or P_d are empty. If Algorithm 9 terminates with $P_r = \emptyset$ then there exists no path from s to z in the corresponding graph. We need Algorithm 9 in every iteration of our Algorithm 8. In every iteration of our algorithm at least one arc of the original series-parallel graph is deleted. We can update the decomposition tree in every iteration by just deleting the arc from the corresponding arc set $E(b)$.

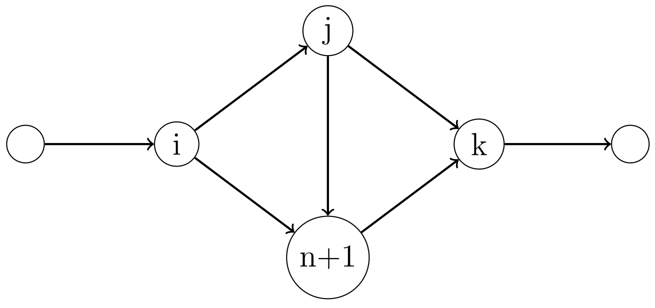


Figure 5.7: A not series-parallel graph.

5.5.2 Complexity Analysis

In this section we will denote the total number of arcs of a network by m and the number of the nodes of the network by n. First we will analyse the complexity of Algorithm 9. Since we go through the decomposition tree and have to perform one of the procedures INITIALIZE(b), MERGE($c, d; b$) or ADD($c, d; b$) at every node of the tree, we need to know how many nodes the decomposition tree can have. The decomposition tree is a binary tree and therefore half of its nodes must be leaf nodes. The leaf nodes of the decomposition tree represent the arcs of the underlying series-parallel graph G. The following lemma gives the maximal number of arcs for any series-parallel graph without parallel arcs:

Lemma 5 *For series-parallel graphs without parallel arcs, the maximal number of arcs is* $2n - 3$.

PROOF: We use an inductive argument to show Lemma 5:

Basis: Consider the easiest possible series-parallel graph: a K_2 consisting only of nodes s and z and one arc (s, z). Since we do not allow parallel arcs, the maximal number of arcs is $1 = 2 \cdot 2 - 3$.

Inductive Hypothesis: For series-parallel graphs without parallel arcs, the maximal number of arcs is $2n - 3$.

Inductive Step: Let G be a series-parallel graph with n nodes and $2n-3$ arcs. Add node $n+1$. Assume that it is possible to add three arcs connecting node $n + 1$ to any three distinct other nodes. Then, since G had the maximal number of arcs and so there are no nodes without any incident arcs, we must get a component as shown in Figure 5.7.

This is not series-parallel. So the assumption must be wrong and it is not possible to connect the new node $n + 1$ to three other distinct nodes. It is also not possible to add more than three arcs, since this would cause the same problem. So we can at most add two new arcs. Hence, the maximal number of arcs for a series-parallel graph with $n+1$ many nodes is: $2 \cdot n - 3 + 2 = 2 \cdot (n+1) - 3$. □

Thus the complexity of Algorithm 9 is $O(n)$, if we neglect the effort involved in the INITIALIZE procedure.
Algorithm 8 deletes at least one arc in every iteration. In the worst case T is large enough to allow all paths from s to z and only after all arcs are deleted there exists no longer a path connecting source and sink. In this case we need m iterations. All steps of Algorithm 8 can be done in constant time, except finding a minimal path from s to z for which we use Algorithm 9. Thus, if we do not count the calls of all the INITIALIZE procedures, the overall complexity of Algorithm 8 is $O(mn)$. Bein and Brucker [5] suggest to use heaps to implement the INITIALIZE procedure. The heaps should represent the sets of parallel arcs $E(b)$. Then it takes constant time to find the arc with minimal costs. The deletion of an arc in Algorithm 8 requires to update the heap, which can be done in $O(\log m)$ time. Thus the overall complexity of Algorithm 8 is $O(mn + m \log m)$.
So our algorithm has the same complexity as the Algorithm 7 of Bein and Brucker and runs in polynomial time. Thus we found a polynomial time algorithm for the special class of earliest arrival flows on series-parallel graphs.

5.6 Earliest Arrival Flows on Series-Parallel Graphs with Inflow-Dependent Transit Times

Consider again the bow graphs of Köhler et al. [8] defined in Chapter 4. Earliest arrival flows always look for the shortest way from s to z and thus will always exhaust the regulating arcs with smaller capacity which lead to faster bow arcs. Thus instead of defining a bow graph with regulating arcs and bow arcs, we can define it by using only parallel bow arcs:

Definition 14 *Let $G = (N, A, T)$ be a dynamic network with an inflow-dependent, piecewise constant and non-decreasing transit time function for every arc $(i, j) \in A$. For each arc denote the inflow rates by $x^1 < x^2 < \ldots$ and the corresponding transit times by $\tau^1 < \tau^2 < \ldots$. In a parallel bow graph we substitute each original arc $(i, j) \in A$ by as many parallel arcs going from i to j as we have different flow rates. To every parallel arc a transit time*

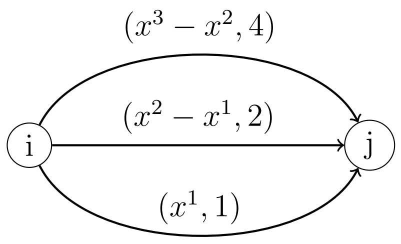


Figure 5.8: Parallel bow graph of the bow graph shown in Figure 4.14.

and the corresponding amount of flow that can travel with this transit time in the bow graph is assigned.

In Figure 5.8 the parallel bow graph of the bow graph given in Figure 4.14 is shown. For a series-parallel graph G the modification of G into a parallel bow graph is still a series-parallel graph where the number of nodes has not changed. Thus we can apply Algorithm 8 to the parallel bow graph. Remember that the bow graph is only a relaxation of the inflow-dependent transit time model. Following the implementation approach of Bein and Brucker [5], presented in the previous section, and modelling parallel arcs via heaps, the computational effort for the parallel bow graph is not much larger than for the original series-parallel graph.

Chapter 6

Summary and Open Problems

We reviewed two classical dynamic flow problems: the maximal dynamic flow problem and the earliest arrival flow problem. We extensively discussed existence, computation and approximation of earliest arrival flows and also gave an overview over the actual research on flow-dependent earliest arrival flows.
As far as we know, it is still an open question if there exists a polynomial time algorithm for the earliest arrival flow problem or not. We have shown that for a special class of graphs, the series-parallel graphs, there does exist a polynomial time earliest arrival flow algorithm.

Application: Both maximal dynamic and earliest arrival flows are used to model transportation or travelling problems where a fixed time horizon is given. Series-parallel graphs do not play an important role in modeling transportation or travelling problems, since the real-world networks are often not series-parallel. Thus our results are mainly of theoretical interest. Also we have not yet implemented our maximal dynamic flow algorithm for series parallel graphs.

Further research: One point for further research is finding a good implementation of Algorithm 8 and testing on a representative set of series-parallel graphs how good the algorithm and its implementation is. Furthermore one could consider earliest arrival flows on series-parallel graphs with varying parameters. We argued that for flow-dependent transit times the relaxation of Köhler et al. [8] can easily be applied to series-parallel graphs and be solved with our algorithm. Köhler et al. [8] only examined the quickest flow problem and developed a polynomial approximation algorithm operating on the bow graph, which is a $(2 + \epsilon)$ approximation for the quickest flow problem with

inflow-dependent transit times. Thus it would be interesting to estimate the quality of this approximation approach for the earliest arrival flow problem. We have seen that even for a graph consisting of only two nodes and one arc, it is not possible to find exact earliest arrival flows for flow-dependent transit times. Thus finding good approximations for this problem is necessary. One could also consider earliest arrival flows with time-dependent transit times on series-parallel graphs.
As already stated above, it is still unknown if the earliest arrival flow problem is NP-hard. So this is also a question for further research.

In the minimum cost dynamic flow problem, each arc has not only assigned a transit time, but also a transit cost. For this problem Klinz and Woeginger [22] showed that the minimum cost dynamic flow problem is NP-hard. Also the minimum cost maximum dynamic flow problem, where the flow value v, that has to be send through the network, is fixed to the maximal possible value, with respect to a fixed time horizon T, and the minimum cost quickest dynamic flow problem, where the time horizon T is fixed to the minimal possible value, with respect to a given flow value v, are NP-hard, even on series-parallel graphs.
Another problem are multicommodity dynamic network flows. Here several commodities with different transit times have to be transhipped through one network. It is an interesting question how to model mulitcommodity network flows and to investigate how to solve mulitcommodity dynamic network flow problems.

Declaration

Hereby I declare that I am the only author of this thesis and no other sources than those listed have been used.

Kaiserslautern, Germany, July 15, 2009

Mechthild Steiner

Bibliography

[1] A decomposition algorithm for shortest paths in a network. *IBM Research Center Report, RC 1562*, 1966.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1993.

[3] N. Baumann and E. Köhler. Approximating earliest arrival flows with flow-dependent transit times. *Discrete Applied Mathematics*, 155(2):161–171, 2007.

[4] N. Baumann and M. Skutella. Solving evacuation problems effeciently - earliest arrival flows with multiple sources. *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 399–410, 2006.

[5] W.W. Bein and P. Brucker. Minimum cost flow algorithms for series-parallel networks. *Discrete Applied Mathematics*, 10:117–124, 1985.

[6] R.C. Busacker and T.L. Saaty. Finite graphs and networks. *McGraw-Hill, New York*, 1965.

[7] M. Carey and E. Subrahmanian. An approach to modeling time-varying flows on congested networks. *Transportation Res. Part B 34*, pages 157–183, 2000.

[8] E. Köhler, K. Langkau, and M. Skutella. Time-expanded graphs for flow-dependent transit times. *Proceedings of the 10th Annual European Symposium on Algorithms*, 2002.

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

[10] L. Fleischer and M. Skutella. The quickest multicommodity flow problem. *Integer Programming and Combinatorial Optimization*, 2337:36–53, 2002.

[11] L.R. Ford, Jr. and D.R. Fulkerson. Constructing maximal dynamic flows from static networks. *Operations Research*, 6(3):419–433, 1958.

[12] L.R. Ford, Jr. and D.R. Fulkerson. *Flows in Networks.* Princeton, N.J.: Princeton University Press, 1962.

[13] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal for the Assoc. for Comp. Mach.*, 34(3):596–615, 1987.

[14] D. Gale. Transient flows in networks. *Michigan Mathematical Journal*, 6(1):59–63, 1959.

[15] B. Hajek and R.G. Ogier. Optimal dynamic routing in communication networks with continous traffic. *Networks*, 14:457–487, 1984.

[16] A. Hall, K. Langkau, and M. Skutella. An FPTAS for quickest multicommodity flows with inflow-dependent transit times. *in: S. Arora, K. Jansen, J.D. Rolim, A. Sahai (Eds.), Approximation, Randomization and Computational Optimization (APPROX'03), Lecture Notes in Computer Science, Springer, Berlin*, 2764:71–82, 2003.

[17] H.W. Hamacher and K. Klamroth. *Lineare Optimierung und Netzwerkoptimierung.* Vieweg Verlag, 2006.

[18] H.W. Hamacher and S.A. Tjandra. Mathematical modelling of evacuation problems: A state of art. *Berichte des Fraunhofer ITWM*, (24), 2001.

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

[20] B. E. Hoppe. *Efficient Dynamic Network Flow Algorithms.* 1995.

[21] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14:205–220, 1967.

[22] B. Klinz and G.J. Woeginger. Minmum-cost dynamic flows: The series-parallel case. *Networks*, 43(3):153–162, 2004.

[23] B. Kotnyek. An annotated overview of dynamic network flows. *INRIA Sophia Antipolis*, 2003.

[24] S.E. Lovetskii and I.I. Melamed. Dynamic flows in networks. *Automation and Remote control*, 48(11):1417–1434, 1987.

[25] E. Minieka. Maximal, lexicographic, and dynamic network flows. *Operations Research*, 21(2):517–527, 1973.

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

[27] S. A. Tjandra. *Dynamic Network Optimization with Application to the Evacuation Problem.* 2003.

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

[29] N. Zadeh. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5:255–266, 1973.