6 Generalized Processing Networks

We turn our considerations to a generalization of the maximum flow problem in which each edge $e = (v, w) \in E$ is assigned with a so called *flow ratio* $\alpha_e \in [0, 1]$ that imposes an upper bound on the fraction of the total outgoing flow at v that may be routed through the edge e. This model embodies a generalization of the maximum flow problem in *processing networks* (Koene, 1982), in which the corresponding flow ratios specify the *exact* fraction of flow rather than only an upper bound. We show that a flow decomposition similar to the one for traditional network flows is possible and can be computed in strongly polynomial time. Moreover, we prove that the problem is at least as hard to solve as any packing LP but that there also exists a fully polynomial-time approximation scheme for the maximum flow problem in these generalized processing networks if the underlying graph is acyclic. For the case of series-parallel graphs, we provide two exact algorithms with strongly polynomial running time. Finally, we study the case of *integral* flows and show that the problem becomes $\mathcal{NP}$-hard to solve and approximate in this case.
This chapter is based on joint work with Sven O. Krumke and Clemens Thielen (Holzhauser et al., 2016c).

6.1 Introduction

Traditional flows in networks that were introduced in Section 2.4 and extended in the previous chapters embody a useful tool to model the transshipment of commodities from nodes with supply to nodes with demand. However, in order to model advanced issues such as the production of goods in a manufacturing process, the considered network flow problems are not powerful enough since they lack the possibility to model the splitting of flow at nodes by specific ratios. *Processing networks* (cf. (Koene, 1982)) generalize traditional flow problems by the introduction of *processing nodes* that involve additional *flow ratios* $\alpha_e \in [0, 1]$ for their outgoing edges e. The flow on such an outgoing edge e is required to equal a fraction α_e of the total flow on the outgoing edges of the processing node. In order to maintain flow conservation, these flow ratios of all outgoing edges of each processing node are required to sum up to one.

© Springer Fachmedien Wiesbaden GmbH 2016
M. Holzhauser, *Generalized Network Improvement and Packing Problems*, DOI 10.1007/978-3-658-16812-4_6

In this chapter, we investigate a generalization of processing networks in which a flow ratio $\alpha_e \in [0,1]$ is assigned to *every* edge e. The flow ratios are required to sum up to *at least* one at every node with outgoing edges and only impose an *upper bound* on the ratio of flow on the corresponding edges. Clearly, this extended model subsumes both the maximum flow problem in processing networks and the maximum flow problem in traditional networks but also allows to model more advanced situations. We provide several structural results about flows in such networks and present both approximation and exact algorithms for the maximum flow problem in several special cases of these networks.

The possible applications of our model are manifold. The most natural one is the modeling of distillation processes (e.g., in refineries), in which raw materials are split into intermediate and end products in specific ratios. However, in contrast to traditional processing networks, we are now able to model possible variations in these ratios that are only bounded by specific technical limitations. Similarly, by inverting the direction of each edge, we can model manufacturing processes of goods in which the composition ratios of the basic commodities may vary up to specific upper bounds.

6.1.1 Previous Work

Research on the topic of processing networks has a long history under several different names. To the best of our knowledge, first work was done by Schaefer (1978) who introduced the maximum flow problem in processing networks and a first algorithm for the problem. In particular, he considered the case that there are two kind of nodes: ordinary nodes as in traditional network flow problems and special nodes for which each of the outgoing edges has an assigned value $\alpha_e \in (0,1)$ that determines the fraction of flow that is routed through the corresponding edge e. In order to maintain flow conservation, these values are required to sum up to one at each special node. Schaefer presented a (super-polynomial-time) algorithm that generalizes the augmenting path algorithm for the traditional maximum flow problem (cf. (Ahuja et al., 1993)). However, the author refrained from giving an exact running time analysis and a complete description on how to handle several special cases that might occur in the course of his algorithm. In the 1980s, Koene (1980) considered the maximum flow problem in a processing network where the only special node (called *processing node*) coincides with the source node s of the network. For this special case, he presented a polynomial-time exact algorithm.

In his PhD-thesis, Koene (1982) later generalized the problem in three ways: Besides the introduction of a third kind of nodes (representing so called *blending processes* that

assign proportionality values to the *incoming* edges of a node) and the introduction of *gains* on edges similar to the generalized flow problem (cf. Section 2.4), he considered the more general *minimum cost flow* variant of the problem. He showed that every linear program can be transformed into an instance of this minimum cost flow problem and developed a customized variant of the simplex method in order to solve the problem. This simplex method was later improved by Chen and Engquist (1986) and Chang et al. (1989).

Many years later, in 2003, research on processing networks was revived by Fang and Qi (2003) under the name *manufacturing network flows* in which they derived the algebraic foundations for a network simplex method. In the following years, Lu et al. (2006), Lu et al. (2009), Venkateshan et al. (2008), and Wang and Lin (2009) extended this foundation, partially under the name *minimum distribution cost flow problem*, with the introduction of explicit graph operations that are used in a network simplex algorithm. Moreover, Wang and Lin (2009) showed that the maximum flow problem in a processing network with both processing and blending nodes is at least as hard as the maximum generalized flow problem as introduced in Section 2.4.

The maximum flow variant of the problem was again investigated by Sheu et al. (2006) and Huang (2011). In the former paper, the authors provide a similar algorithm to the very early procedure introduced by Schaefer (1978) with super-exponential running time but neither give a proof of correctness nor handle every special case that may occur. In Huang (2011), the author presents a network simplex method for the problem without processing nodes in combination with computational results.

The case that the corresponding factors do not sum up to one at some nodes was considered in Lu et al. (2006). However, the authors do not assume flow conservation to hold at these nodes and are, thus, able to define preprocessing procedures in order to remove such nodes. To the best of our knowledge, the more general case that is considered in this chapter, in which these factors only provide *upper bounds* on the flow while flow conservation is maintained at each node has not been investigated so far.

6.1.2 Chapter Outline

After defining the *maximum flow problem in generalized processing networks* and the necessary notation in Section 6.2, we show in Section 6.3 that there is a flow decomposition theorem similar to the one for traditional flows (cf. (Ahuja et al., 1993)) and that such a flow decomposition can be computed more efficiently in the case of acyclic graphs. In Section 6.4, we consider the complexity and approximability of the problem. In

particular, we show that the problem of finding a maximum flow in a generalized processing network is solvable in weakly polynomial-time on the one hand, but at least as hard to solve as any packing LP on the other hand. Moreover, we present an FPTAS for the problem on acyclic graphs that is based on the generalized packing framework introduced in Section 3.3. To the best of our knowledge, this comprises the first approximation algorithm for the maximum flow problem in processing networks. In Section 6.5, we turn our focus to the case of series-parallel graphs and present two different approaches on how to solve the problem exactly in strongly polynomial time. The first of these approaches is an analogue to the augmenting path algorithm for the traditional maximum flow problem (cf. (Ahuja et al., 1993)) and achieves a running time of $\mathcal{O}(m^2)$ while the second approach exhaustedly uses the inherent structure of series-parallel graphs in order to repeatedly shrink series-parallel subcomponents into single edges, which results in an algorithm with an improved running time of $\mathcal{O}(m \cdot (n + \log m))$. Finally, in Section 6.6, we briefly investigate the case of flows that are required to be integral on every edge. As it turns out, the problem with integral flows becomes strongly $\mathcal{NP}$-complete to solve and to approximate even on bipartite acyclic graphs and weakly $\mathcal{NP}$-complete to solve and to approximate on series-parallel graphs. An overview of the results of this chapter is given in Table 6.1 and Table 6.2 on page 152.

6.2 Preliminaries

We start by defining the maximum flow problem in a directed graph $G = (V, E)$ with *edge capacities* $u_e \in \mathbb{N}$ and *flow ratios* $\alpha_e \in (0, 1]$ on the edges $e \in E$. Let $s \in V$ and $t \in V$ denote a distinguished *source* and *sink* of the network, respectively.

Definition 6.1 (Flow, flow value, maximum flow, static/dynamic capacity constraint):
A function $x\colon E \to \mathbb{R}_{\geqslant 0}$ is called a *feasible flow in a generalized processing network* or just *flow* if $\text{excess}_x(v) := \sum_{e \in \delta^-(v)} x_e - \sum_{e \in \delta^+(v)} x_e = 0$ for each $v \in V \setminus \{s, t\}$ and both $x_e := x(e) \leqslant u_e$ (called the *static capacity constraint for* e) and $x_e \leqslant \alpha_e \cdot \sum_{e' \in \delta^+(v)} x_{e'}$ (called the *dynamic capacity constraint for* e) for each $e = (v, w) \in E$. The *flow value* of a flow x is given by $\text{val}(x) := \text{excess}_x(t)$. A flow x of maximum flow value is called a *maximum flow in a generalized processing network* or just *maximum flow*. ◁

The above definition allows us to define the *maximum flow problem in a generalized processing network*:

Definition 6.2 (Maximum flow problem in a generalized processing network (MFGPN)):

INSTANCE: A directed graph $G = (V, E)$ with source $s \in V$, sink $t \in V$, capacities $u_e \in \mathbb{N}$, and flow ratios $\alpha_e \in (0, 1]$ on the edges $e \in E$ such that $\sum_{e \in \delta^+(v)} \alpha_e \geqslant 1$ for each $v \in V$ with $\delta^+(v) \neq \emptyset$.

TASK: Determine a maximum flow in G. ◁

Note that we have required that $\sum_{e \in \delta^+(v)} \alpha_e \geqslant 1$ for each $v \in V$ with $\delta^+(v) \neq \emptyset$ in Definition 6.2. However, this does not yield any restriction since flow conservation holds at nodes with $\sum_{e \in \delta^+(v)} \alpha_e \in (0, 1)$ only if the flow on the outgoing edges is zero. Consequently, we can find and remove such nodes in a preprocessing step in $\mathcal{O}(n + m)$ time. This fact is held down in the following assumption:

Assumption 6.3: For every node $v \in V$ with $\delta^+(v) \neq \emptyset$, the flow ratios of its outgoing edges fulfill $\sum_{e \in \delta^+(v)} \alpha_e \geqslant 1$. ◁

In addition to Assumption 6.3, we make the following assumptions on the structure of the underlying graph:

Assumption 6.4: For every node $v \in V \setminus \{s, t\}$, it holds that $\delta^+(v) \neq \emptyset$ and $\delta^-(v) \neq \emptyset$. ◁

Assumption 6.5: For every node $v \in V \setminus \{s, t\}$, there is at least one directed path from s to v or from v to t. ◁

Assumption 6.4 does not impose any restriction on the underlying model since the inflow and outflow of every node $v \in V \setminus \{s, t\}$ with $\delta^+(v) = \emptyset$ or $\delta^-(v) = \emptyset$ must equal zero due to flow conservation at v, which implies that the incident edges can be deleted in a preprocessing step. Similarly, Assumption 6.5 yields no restriction since the corresponding connected components do not contribute to the flow value in any flow and can be deleted as well. Note that, for any instance of MFGPN, both assumptions can be established in $\mathcal{O}(n + m)$ time by performing a depth-first search and repeatedly deleting single nodes and edges. The resulting graph is connected, such that we can assume that $n \in \mathcal{O}(m)$ in the following.

Using the above definitions, we can formulate the maximum flow problem in a generalized processing network as a linear program as follows:

$$\max \sum_{e \in \delta^-(t)} x_e - \sum_{e \in \delta^+(t)} x_e \tag{6.1a}$$

$$\text{s.t.} \sum_{e \in \delta^-(v)} x_e - \sum_{e \in \delta^+(v)} x_e = 0 \qquad \text{for all } v \in V \setminus \{s, t\}, \tag{6.1b}$$

$$x_e \leqslant \alpha_e \cdot \sum_{e' \in \delta^+(v)} x_{e'} \qquad \text{for all } e = (v, w) \in E, \tag{6.1c}$$

$$0 \leqslant x_e \leqslant u_e \qquad \text{for all } e \in E. \tag{6.1d}$$

Note that this formulation as a linear program only differs in equation (6.1c) from the linear programming formulation of the traditional maximum flow problem given in equations (2.1) on page 14. However, the known combinatorial algorithms for the traditional maximum flow problem cannot be applied directly to MFGPN. Instead, we need to make use of new approaches and generalizations of existing results. The following definitions build the basis for the theoretical framework that will be used in the remainder of this chapter.

Definition 6.6 (Types of edges):
Let x be a feasible flow in a generalized processing network. An edge $e = (v, w) \in E$ is said to be of *type* u if $x_e = u_e$. Similarly, if $x_e < u_e$ and $x_e = \alpha_e \cdot \sum_{e' \in \delta^+(v)} x_{e'}$, the edge is said to be of *type* α. ◁

Definition 6.7 ((Basic) Flow distribution scheme):
A function $\beta \colon E \to [0, 1]$ with $\beta_e := \beta(e) \leqslant \alpha_e$ for each $e \in E$ is called a *flow distribution scheme* if, for each $v \in V \setminus \{t\}$, it holds that $\sum_{e \in \delta^+(v)} \beta_e = 1$. Furthermore, if there is at most one edge $e \in \delta^+(v)$ with $\beta_e \notin \{0, \alpha_e\}$ at each node $v \in V \setminus \{t\}$, the function is called a *basic flow distribution scheme*. ◁

Intuitively, each flow distribution scheme determines how flow that arrives at some node $v \in V$ is sent through the outgoing edges without violating the dynamic capacity constraints or the flow conservation constraints. Thus, each flow distribution scheme together with a sufficiently small flow value $\mathrm{val}(x)$ determines a feasible flow x. Note that the concept of basic flow distribution schemes is a generalization of the notion of s-t-paths in traditional network flow problems since we obtain such a path for the case that $\alpha_e = 1$ for each $e \in E$. Moreover, note that the fraction $\frac{x_e}{\mathrm{val}(x)}$ is constant for each edge $e \in E$ and every flow x determined by a given flow distribution scheme β, which leads to the following definition:

Definition 6.8 (Flow on flow distribution scheme, weight of edge in flow distribution scheme):
Let β be a flow distribution scheme. A flow x that fulfills $x_e = \beta_e \cdot \sum_{e' \in \delta^+(v)} x_{e'}$ for each $e = (v, w) \in E$ is called a *flow on* β. For a flow x on β with positive flow value $\mathrm{val}(x)$, the fraction $w_\beta(e) := \frac{x_e}{\mathrm{val}(x)} \in [0, 1]$ (which is independent of the choice of x) is called the *weight of* e *in* β. ◁

In particular, note that the weight function w_β also embodies a flow with unit flow value for each flow distribution scheme β.

The notion of flow distribution schemes shows the main difference between our model and traditional processing networks: In the latter model, for each flow distribution scheme β, it always holds that $\beta_e = \alpha_e$ for each edge $e = (v, w)$ that leaves a special node v. This implies that the flow on each edge in $\delta^+(v)$ is determined by the flow on e. In our generalized model, however, there are multiple possible (basic) flow distribution schemes at each such node, which prevents us from directly applying the known algorithms for traditional processing networks to the generalized model.

6.3 Structural Results

We start by generalizing existing results for traditional flows to the case of MFGPN. As it turns out, a flow decomposition that is similar to the well-known flow decomposition of traditional flows is possible in the case of MFGPN as well (cf. (Ahuja et al., 1993)). To obtain this result, we need the following lemma:

Lemma 6.9:
A feasible flow on a given flow distribution scheme β that is positive on at least one edge can be determined in $\mathcal{O}(m^3)$ time.

Proof: Let $E_0 := E \cup \{e_0\}$ with $e_0 = (t, s)$ and $\beta_{e_0} := 1$. We show that we can find a non-zero feasible circulation[1] x on β (extended to e_0) in $G_0 = (V, E_0)$ within the given time bound, which clearly shows the claim.

Consider some edge $e = (v, w) \in E_0$. For every feasible circulation x on β, it must hold that $x_e - \beta_e \cdot \sum_{e' \in \delta^-(v)} x_{e'} = 0$ in order to be feasible on β and to fulfill flow conservation. The set of these constraints for each $e \in E$ builds a homogeneous linear equation system of the form $A \cdot x = 0$ over $m + 1$ variables with $m + 1$ constraints. Note that the sum of the coefficients amounts to zero in each row and column. Hence, the rank of the matrix A is at most m and, thus, the dimension of the kernel is at least one. Consequently, there is a non-zero vector x that solves the linear equation system. Without loss of generality, there is at least one edge $e = (v, w)$ with $x_e > 0$ in this vector. Since $x_e - \beta_e \cdot \sum_{e' \in \delta^-(v)} x_{e'} = 0$, it both holds that $\sum_{e' \in \delta^-(v)} x_{e'} > 0$ (i.e. there is at least one edge $e' \in \delta^-(v)$ with $x_{e'} > 0$) and that $x_{e''} \geqslant 0$ for each $e'' \in \delta^+(v)$ (since $x_{e''} - \beta_{e''} \cdot \sum_{e' \in \delta^-(v)} x_{e'} = 0$ as well and $\beta_{e''} \geqslant 0$). Since the underlying graph G_0 is strongly connected according to Assumption 6.4, Assumption 6.5, and due to the

1 A *feasible circulation* x is a feasible flow that fulfills $\text{excess}_x(v) = 0$ for each node $v \in V$.

additional edge e_0, an inductive argument yields that $x_e \geqslant 0$ for each $e \in E_0$. Hence, a suitable multiple of x yields a feasible flow in the underlying network. Since such a vector x can, e.g., be found by the Gaussian elimination procedure in $\mathcal{O}(m^3)$ time, the claim follows. □

Theorem 6.10:
Each flow x can be decomposed into $\kappa \leqslant 2m$ flows $x^{(i)}$ on basic flow distribution schemes $\beta^{(i)}$ for $i \in \{1, \dots, \kappa\}$. Such a decomposition can be found in $\mathcal{O}(m^4)$ time.

Proof: Let x be a feasible flow in $G = (V, E)$ with flow value $\mathrm{val}(x) > 0$. Without loss of generality, we can ignore edges carrying zero flow. For each $v \in V$ with positive outflow, let $(e_1, \dots, e_k)$ denote an ordering of the edges in $\delta^+(v)$ such that $\frac{x_{e_i}}{\alpha_{e_i}} \geqslant \frac{x_{e_j}}{\alpha_{e_j}}$ for $i < j$ (in particular, edges of type α are located at the front of the ordering). If $\sum_{i=1}^{k} \alpha_{e_i} = 1$, we set $\beta_{e_i} := \alpha_{e_i}$ for each $i \in \{1, \dots, k\}$. Else, if $\sum_{i=1}^{k} \alpha_{e_i} > 1$, there is some index $h \leqslant k$ with $\sum_{i=1}^{h-1} \alpha_{e_i} \leqslant 1$ and $\sum_{i=1}^{h} \alpha_{e_i} > 1$. By setting $\beta_{e_i} := \alpha_{e_i}$ for $i \in \{1, \dots, h-1\}$, $\beta_{e_h} := 1 - \sum_{i=1}^{h} \beta_{e_i}$, and $\beta_{e_j} := 0$ for $j \in \{h+1, \dots, k\}$ for each such node $v \in V$, we, thus, obtain a basic flow distribution scheme. Note that, for each $e \in E$, it holds that $\beta_e > 0$ only if $x_e > 0$ and that $\beta_e = \alpha_e$ whenever e is of type α.

Let $\overline{x}$ denote a feasible flow on β, which can be found in $\mathcal{O}(m^3)$ time according to Lemma 6.9. We claim that, for a suitable choice of $\delta > 0$, the flow $x(\delta) := x - \delta \cdot \overline{x}$ remains feasible. Obviously, for each choice of δ, flow conservation remains fulfilled at each node $v \in V$ since

$$\sum_{e \in \delta^-(v)} (x(\delta))_e - \sum_{e \in \delta^+(v)} (x(\delta))_e = \sum_{e \in \delta^-(v)} (x_e - \delta \cdot \overline{x}_e) - \sum_{e \in \delta^+(v)} (x_e - \delta \cdot \overline{x}_e)$$
$$= \left(\sum_{e \in \delta^-(v)} x_e - \sum_{e \in \delta^+(v)} x_e \right) - \delta \cdot \left(\sum_{e \in \delta^-(v)} \overline{x}_e - \sum_{e \in \delta^+(v)} \overline{x}_e \right) = 0 - 0 = 0.$$

For the flow on each edge $e \in E$ to remain non-negative, it must hold that $x((\delta))_e = x_e - \delta \cdot \overline{x_e} \geqslant 0$, i.e., $\delta \leqslant \frac{x_e}{\overline{x}_e}$ for each $e \in E$ with $\overline{x}_e > 0$. Moreover, in order to fulfill the dynamic capacity of each edge $e = (v, w) \in E$, the value δ must fulfill the following constraint:

$$(x(\delta))_e \leqslant \alpha_e \cdot \sum_{e' \in \delta^+(v)} (x(\delta))_{e'}$$
$$\iff \quad (x_e - \delta \cdot \overline{x}_e) \leqslant \alpha_e \cdot \sum_{e' \in \delta^+(v)} (x_{e'} - \delta \cdot \overline{x}_{e'})$$
$$\iff \quad \delta \cdot \left(\alpha_e \cdot \left(\sum_{e' \in \delta^+(v)} \overline{x}_{e'} \right) - \overline{x}_e \right) \leqslant \alpha_e \cdot \left(\sum_{e' \in \delta^+(v)} x_{e'} \right) - x_e. \tag{6.2}$$

Note that both sides of inequality (6.2) are non-negative since $\overline{x}$ and x fulfill the dynamic capacity constraints. For the case that e is of type α in $\overline{x}$ (which is, e.g., true if e is also of type α in x according to the construction of β), inequality (6.2) is fulfilled for every choice of δ since $\alpha_e \cdot (\sum_{e' \in \delta^+(v)} \overline{x}_{e'}) - \overline{x}_e = 0$. Otherwise, (6.2) is equivalent to

$$\delta \leqslant \frac{\alpha_e \cdot \left(\sum_{e' \in \delta^+(v)} x_{e'}\right) - x_e}{\alpha_e \cdot \left(\sum_{e' \in \delta^+(v)} \overline{x}_{e'}\right) - \overline{x}_e}. \tag{6.3}$$

Let δ be the maximum value that fulfills both $\delta \leqslant \frac{x_e}{\overline{x}_e}$ for each e with $\overline{x}_e > 0$ and inequality (6.3) for each $e \in E$ that is not of type α in $\overline{x}$. By the above arguments, it follows that $\delta \cdot \overline{x}$ is a feasible flow on β and that the remaining flow $x(\delta)$ is feasible as well. Moreover, note that the flow on at least one edge in $x(\delta)$ becomes zero (for the case that $\delta = \frac{x_e}{\overline{x}_e}$ for some edge $e \in E$), or at least one edge $e \in E$ that was not of type α in x is of type α in $x(\delta)$. In the latter case, edge e will remain of type α after each of the following iterations of the algorithm according to the definition of β.

Hence, the above procedure executes at most $2m$ iterations while each of these iterations runs in $\mathcal{O}(m^3)$ time according to Lemma 6.9, which shows the claim. □

Note that, on a graph without dynamic capacities (i.e., with $\alpha_e = 1$ for each $e \in E$), each flow on a basic flow distribution scheme β either corresponds to a flow on an s-t-path or on a cycle. Thus, Theorem 6.10 is a generalization of the flow decomposition theorem for traditional flows (cf. (Ahuja et al., 1993)).

We now restrict our considerations to the case of acyclic graphs. In this case, the running time of finding a flow decomposition can be significantly improved compared to Theorem 6.10. Recall that the weight $w_\beta(e)$ of an edge $e \in E$ in a flow distribution scheme β is independent of the choice of the underlying flow x according to Definition 6.8.

Lemma 6.11:
The weights $w_\beta(e)$ of all edges $e \in E$ in a given flow distribution scheme β can be determined in $\mathcal{O}(m)$ time on acyclic graphs.

Proof: Let $(v_1 = s, v_2, \ldots, v_{n-1}, v_n = t)$ denote a topological sorting of the nodes, which can be determined in $\mathcal{O}(m)$ time (cf., e.g., Cormen et al. (2009)). For $i = 1$, the weight of each edge $e \in \delta^+(v_i)$ is directly given by β, i.e., we get that $w_\beta(e) := \beta_e$. Now assume that we know the weights for all edges in $\delta^+(v_j)$ for $j \in \{1, \ldots, i\}$ and consider the subsequent node v_{i+1} in the ordering. In each flow x on β with flow value $\mathrm{val}(x)$, the amount of flow that reaches v_{i+1} is given by $F := \sum_{e \in \delta^-(v_{i+1})} x_e =$

$\sum_{e\in\delta^-(v_{i+1})} \mathrm{val}(x)\cdot w_\beta(e)$. For each $e \in \delta^+(v_{i+1})$, the flow on x_e is then given by $x_e = \beta_e \cdot F$, i.e., the weight of e amounts to

$$w_\beta(e) = \frac{x_e}{\mathrm{val}(x)} = \frac{\beta_e \cdot F}{\mathrm{val}(x)} = \beta_e \cdot \sum_{e'\in\delta^-(v_{i+1})} w_\beta(e').$$

Repeating the above procedure for each node $v_i \in V$, the weight of each edge in β can be determined in $\mathcal{O}(m)$ time, which shows the claim. □

Corollary 6.12:
A feasible flow on a given flow distribution scheme β that is positive on at least one edge can be determined in $\mathcal{O}(m)$ time on acyclic graphs.

Proof: According to Lemma 6.11, we can determine the weights $w_\beta(e)$ in β of all edges $e \in E$ in $\mathcal{O}(m)$ time. Note that a flow of value F on β results in a flow of value $w_\beta(e)\cdot F$ on each edge $e \in E$. Hence, for any F with $0 < F \leqslant \min\{\frac{u_e}{w_\beta(e)} : e \in E \text{ and } w_\beta(e) > 0\}$, the flow x with $x_e := F \cdot w_\beta(e)$ for each $e \in E$ is a feasible flow with positive flow value F, which shows the claim. □

Using the result of Corollary 6.12 in the proof of Theorem 6.10, we immediately get the following result:

Theorem 6.13:
On acyclic graphs, each flow x can be decomposed into at most $2m$ flows on basic flow distribution schemes in $\mathcal{O}(m^2)$ time. □

6.4 Complexity and Approximability

In this section, we consider the complexity and approximability of the maximum flow problem in generalized processing networks. Although MFGPN is solvable in polynomial time, it turns out to be much harder to solve than the maximum flow problem in traditional networks. Nevertheless, for the case of acyclic graphs, we will be able to derive an FPTAS for the problem that runs in strongly polynomial time.

6.4.1 Complexity

Note that the linear program (6.1a) – (6.1d) can be solved in (weakly) polynomial time by known techniques such as interior point methods (cf. (Schrijver, 1998)). In

particular, using the procedure by Vaidya (1989) that was described in Section 4.2, we get the following weakly polynomial running time for MFGPN:

Theorem 6.14:
MFGPN is solvable in weakly polynomial time $\mathcal{O}\left(m^{3.5} \log M\right)$ if $M \geqslant \max_{e \in E} u_e$ and each flow ratio is a rational number with numerator and denominator at most M. □

However, as in the previous chapters of this thesis, we are in particular interested in combinatorial algorithms for the treated problems that exploit the discrete structure of the underlying network. As for the case of traditional flows, the flow decomposition theorem that was derived in Section 6.3 is only a structural result and does not immediately yield an algorithm that solves the problem of finding an optimal solution. In fact, it turns out that the problem MFGPN seems to be much more complicated than the traditional maximum flow problem since every packing LP of the form $\max\left\{c^T x : Ax \leqslant b, x \geqslant 0\right\}$ for positive rational vectors c and b and a matrix A with non-negative entries can be reduced to MFGPN in linear time. This result was first published by Schaefer (1978). Since it seems to be widely unnoticed in present literature, we present a short proof in the following.

Theorem 6.15 (Schaefer (1978)):
Every packing LP can be solved by computing a maximum flow in a generalized processing network. This network can be constructed from the given packing LP in linear time.

Proof: Let $\max\left\{c^T x : A^T x \leqslant b, x \geqslant 0\right\}$ be a packing LP with $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{m \times n}$, $c_j > 0$ for $j \in \{1, \ldots, n\}$, $b_i > 0$ for $i \in \{1, \ldots, m\}$, and $a_{ij} \geqslant 0$ for $j \in \{1, \ldots, n\}, i \in \{1, \ldots, m\}$.

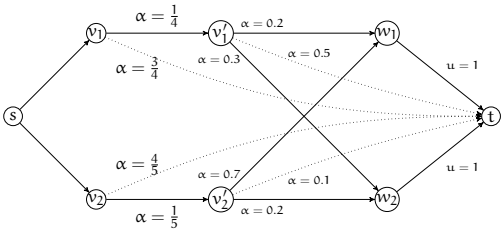
Without loss of generality, we may assume that $b_i = 1$ for each $i \in \{1, \ldots, m\}$ and that $c_j \geqslant 1$ for each $j \in \{1, \ldots, n\}$ since we can otherwise scale the corresponding row or the objective function, respectively, by appropriate factors. Similarly, we may assume that $\sum_{i=1}^{m} a_{ij} \leqslant 1$ for each $j \in \{1, \ldots, n\}$: Otherwise, for $1 < q := \max\left\{\sum_{i=1}^{m} a_{ij} : j \in \{1, \ldots, n\}\right\}$, we could use the equivalent LP formulation $\max\{\frac{1}{q} c^T x' : \frac{1}{q} A^T x' \leqslant 1, x' \geqslant 0\}$ and afterwards substitute $x := \frac{1}{q} \cdot x'$.

We construct an instance of MFGPN as follows: Aside from a source s and sink t, we insert two nodes v_j and v'_j for each $j \in \{1, \ldots, n\}$ and a node w_i for each $i \in \{1, \ldots, m\}$. We connect s with each node v_j and insert an edge with capacity 1 between each node w_i and the sink t. Moreover, we insert an edge between v_j and v'_j with flow ratio $\frac{1}{c_j}$ and an edge that heads from v_j to t with flow ratio $1 - \frac{1}{c_j}$. Finally, we add an edge between each v'_j and each w_i with flow ratio a_{ij} and one edge between each v'_j and t with flow ratio $1 - \sum_{i=1}^{m} a_{ij}$. If not mentioned explicitly, the flow ratio of

each edge is 1 and the capacity is infinite. An example of a packing LP and the corresponding network is depicted in Figure 6.1.

$$\begin{aligned} \max\ & 4x_1 + 5x_2 \\ \text{s.t.}\ & 0.2x_1 + 0.7x_2 \leqslant 1 \\ & 0.3x_1 + 0.2x_2 \leqslant 1 \\ & x_1, x_2 \geqslant 0 \end{aligned}$$

(a) Packing LP



(b) Instance of MFGPN

Figure 6.1: An example packing LP (left) and the corresponding instance of MFGPN (right). If not depicted, the flow ratio of each edge is 1 and the capacity is infinite.

It is now easy to see that the flow value of a maximum flow in the constructed network equals the optimum value of the given (transformed) packing LP instance: For $j \in \{1, \ldots, n\}$, the flow on the edge between v_j and v_j' can be interpreted as the value of variable x_j. Since this edge has a flow ratio of $\frac{1}{c_j}$, it is necessary to send $c_j \cdot x_j$ units of flow from s to v_j in order to obtain a flow of value x_j between v_j and v_j', which corresponds to the contribution of x_j to the objective function value of the LP. Moreover, for $i \in \{1, \ldots, m\}$, the flow on the edge between w_i and t can be interpreted as the left-hand side of the constraint $\sum_{j=1}^{n} a_{ij} \cdot x_j \leqslant 1$ and the capacity of the edge enforces the constraint to be fulfilled. Finally, the interconnections between the nodes v_j' and w_i model the effect that the corresponding variables x_j have on the value of the left-hand side of each constraint i.

Hence, by solving the constructed instance of MFGPN and interpreting the flow values on each edge between v_j and v_j' as the value of variable x_j, we obtain an optimal solution of the given packing LP. Since the above transformations and construction of the network work in linear time, the claim follows. □

6.4.2 Approximability

The results of the previous subsection imply that, even on acyclic graphs, the maximum flow problem in generalized processing networks is much more complicated than the maximum flow problem in traditional networks, so strongly polynomial-time combinatorial algorithms may not necessarily exist for MFGPN. Nevertheless, as

it will be shown in the following, we can use the special structure of acyclic graphs in order to obtain an FPTAS for finding a maximum flow in an acyclic generalized processing network by incorporating the generalized packing framework that was introduced in Section 3.3. Even more, this FPTAS can be implemented to run in strongly polynomial time, in contrast to interior point methods. To this end, we need the following result:

Lemma 6.16:
Let y denote a function that assigns a positive weight $y_e := y(e) > 0$ to each edge $e \in E$. A basic flow distribution scheme β that minimizes the total weight $\sum_{e \in E} w_\beta(e) \cdot y_e$ can be found in $\mathcal{O}(m)$ time on acyclic graphs.

Proof: Let $(v_1, \ldots, v_n)$ denote a topological sorting of the node set V, which can be found in $\mathcal{O}(m)$ time. For each $i \in \{1, \ldots, n\}$, let $G^{(i)} := (V^{(i)}, E^{(i)})$ with $V^{(i)} := \{v_i, \ldots, v_n\}$ and $E^{(i)} := \{e = (v_j, v_l) \in E : j, l \geqslant i\}$ denote the subgraph induced by $\{v_i, \ldots, v_n\}$. Moreover, let $w(i)$ denote the minimum total weight of a basic flow distribution scheme in $G^{(i)}$ with respect to y.

Clearly, since $G^{(n)}$ contains no edge at all, it holds that $w(n) = 0$. Now assume that we want to determine the value of $w(i)$ for some $i \in \{1, \ldots, n-1\}$ and that the values $w(i+1), \ldots, w(n)$ are already known. In order to find a (not necessarily basic) flow distribution scheme β, we need to assign values $\beta_e \in [0, 1]$ to each $e \in \delta^+(v_i)$ such that $\sum_{e \in \delta^+(v_i)} \beta_e = 1$. A value of β_e for some edge $e = (v_i, v_l) \in \delta^+(v_i)$ increases the total weight $w(i)$ by $\beta_e \cdot y_e + \beta_e \cdot w(l)$ since $w_\beta(e) = \beta_e$ in $G^{(i)}$ and since a fraction β_e of the total flow must be sent through $G^{(l)}$. Thus, the minimum total weight in $G^{(i)}$ is given by the following linear program:

$$
\begin{aligned}
w(i) = \min \quad & \sum_{e=(v_i, v_l) \in \delta^+(v_i)} \beta_e \cdot (y_e + w(l)) \\
\text{s.t.} \quad & \sum_{e \in \delta^+(v_i)} \beta_e = 1, \\
& 0 \leqslant \beta_e \leqslant \alpha_e \qquad \text{for all } e \in E.
\end{aligned}
$$

Similarly to the fractional knapsack problem (cf. Kellerer et al. (2004)), it is easy to see that an optimal solution to this fractional packing problem can be determined by the following procedure: If $\sum_{e \in \delta^+(v_i)} \alpha_e = 1$, the only feasible solution is given by $\beta_e = \alpha_e$ for each $e \in \delta^+(v_i)$. Otherwise, if $\sum_{e \in \delta^+(v_i)} \alpha_e > 1$, let $(e_1, \ldots, e_k)$ denote a sorting of the outgoing edges of v_i in non-decreasing order of their coefficients $y_e + w(l)$. Let l be the unique index such that $\sum_{j=1}^{l-1} \alpha_e \leqslant 1$ and $\sum_{j=1}^{l} \alpha_e > 1$. By setting $\beta_{e_j} := \alpha_{e_j}$ for $j \in \{1, \ldots, l-1\}$, $\beta_{e_l} = 1 - \sum_{j=1}^{l-1} \beta_{e_j}$, and $\beta_{e_j} = 0$ for $j \in \{l+1, \ldots, k\}$, we then get an optimal solution. Similar to the fractional knapsack problem, we can find this index l

in $\mathcal{O}(k)$ time by using weighted medians (cf. (Korte and Vygen, 2002)). Note that, in this solution, it holds that $\beta_e \notin \{0, \alpha_e\}$ for at most one edge, i.e., the optimal solution is a basic flow distribution scheme. □

Note that the above algorithm is strongly combinatorial according to the definition that was given in Section 3.1. This leads to the following theorem:

Theorem 6.17:
There is an FPTAS for the maximum flow problem in acyclic generalized processing networks that runs in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m\right)$ time.

Proof: The proof is composed of three results that have already been shown before: According to Theorem 6.10 (and Theorem 6.13), each flow x in a generalized processing network can be decomposed into at most $2m$ flows on basic flow distribution schemes, i.e., each flow x lies in the cone C that is generated by the (possibly exponential-size, but finite) set $S := \{w_\beta : \beta \text{ is a flow distribution scheme}\}$ of flows with unit flow value on basic flow distribution schemes. Moreover, since we can rewrite the objective function of the maximum flow problem in generalized processing networks as $\max \sum_{e \in E} c_e \cdot x_e$ with $c_e := 1$ for $e \in \delta^-(t)$ and $c_e := 0$ for $e \in E \setminus \delta^-(t)$, it holds that $\sum_{e \in E} c_e \cdot w_\beta(e) = 1$ for all flows $w_\beta \in S$. Hence, we obtain the following equivalent formulation of MFGPN:

$$\begin{aligned} \max \sum_{e \in E} c_e \cdot x_e & \\ \text{s.t. } x_e \leqslant u_e & \qquad \text{for all } e \in E, \\ x \in C. & \end{aligned}$$

The constraint matrix of this formulation only contains $N = m$ non-zero entries since both the flow conservation constraints and the dynamic capacity constraints are modeled by the containment in the cone C. Since, for a given cost vector y, we can determine a flow distribution scheme β that minimizes the total weight $\sum_{e \in E} w_\beta(e) \cdot y_e$ in $\mathcal{O}(m)$ time according to Lemma 6.16, the claim immediately follows by Theorem 3.5. □

Note that Theorem 6.17 can be easily generalized to the *minimum-cost flow problem in a generalized processing network*, in which the objective function is replaced by a general linear cost function of the form $\min \sum_{e \in E} c_e \cdot x_e$. By the same arguments that were used in the proof of Theorem 6.17, we get an FPTAS for this much more general problem running in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^3 \log m\right)$ time by using Theorem 3.10. Moreover, note that we can solve budget-constrained versions of both the maximum and the minimum cost flow problem in a generalized processing network within the same running times as the unconstrained versions according to Theorem 3.5 and Theorem 3.10, respectively.

6.5 Series-Parallel Graphs

In this section, we investigate the maximum flow problem for generalized processing networks on series-parallel graphs. Since each series-parallel graph is acyclic, in particular, the positive results from Section 6.4 apply here as well. However, we are now able to derive two algorithms that compute a maximum flow in a series-parallel generalized processing network in (strongly) polynomial time. In the following three results, we investigate a special case of the problem which will be used as a building block for the two upcoming polynomial-time procedures in Section 6.5.1 and Section 6.5.2. To this end, let $E(v,w) := \delta^+(v) \cap \delta^-(w)$ denote the set of all edges between the two nodes $v, w \in V$.

Lemma 6.18:
Let v and w be two nodes such that all edges $\{e_1, \dots, e_k\}$ that leave v are parallel edges heading to w, i.e., $\delta^+(v) = E(v,w)$, and assume that the edges are ordered such that $\frac{u_{e_i}}{\alpha_{e_i}} \leqslant \frac{u_{e_j}}{\alpha_{e_j}}$ for $i < j$. Then the maximum flow between v and w fulfills the property that there exists an index $h \in \{1, \dots, k\}$ such that all edges e_i with $1 \leqslant i \leqslant h$ are of type u and all edges e_j with $h+1 \leqslant j \leqslant k$ are of type α. This index h can be computed in $\mathcal{O}(k)$ time.

Proof: Let x be any maximum flow between v and w. Clearly, we may assume that each edge is either of type u or of type α in x since the total flow value could else be further improved. In the following, we show that we can label several edges with type α in an iterative process from right to left (i.e., from higher indices to lower indices) until we find the desired index h, which allows us to assign the remaining edges to type u and stop the procedure.

Let e_i be some edge that has not yet been labeled such that all edges e_j with $i+1 \leqslant j \leqslant k$ are of type α (initially, choose $i := k$). Since these edges e_j are of type α, a fixed fraction $\alpha^{(i)} := \sum_{j=i+1}^{k} \alpha_{e_j}$ of the total outflow of v flows through the edges e_j, $i+1 \leqslant j \leqslant k$. Thus, the maximum flow value $F := \mathrm{val}(x) = \sum_{j=1}^{k} x_{e_j}$ is determined by the flow values on the edges $e_1, \dots, e_i$ as

$$F = \sum_{j=1}^{k} x_{e_j} = \sum_{j=1}^{i} x_{e_j} + \sum_{j=i+1}^{k} x_{e_j} = \sum_{j=1}^{i} x_{e_j} + \alpha^{(i)} \cdot F$$

$$\iff F = \frac{1}{1-\alpha^{(i)}} \cdot \sum_{j=1}^{i} x_{e_j}.$$

First consider the case that $\alpha_{e_i} < (1-\alpha^{(i)}) \cdot \frac{u_{e_i}}{\sum_{j=1}^{i} u_{e_j}}$. In this case, edge e_i cannot be of type u since this would imply that

$$\alpha_{e_i} \cdot F = \alpha_{e_i} \cdot \frac{1}{1-\alpha^{(i)}} \cdot \sum_{j=1}^{i} x_{e_j} \leqslant \alpha_{e_i} \cdot \frac{1}{1-\alpha^{(i)}} \cdot \sum_{j=1}^{i} u_{e_j} < u_{e_i} = x_{e_i},$$

so the dynamic capacity of edge e_i would be violated. Thus, edge e_i and, hence, all edges e_j with $j \in \{i, \dots, k\}$ are of type α.

Now consider the case that $\alpha_{e_i} \geqslant (1-\alpha^{(i)}) \cdot \frac{u_{e_i}}{\sum_{j=1}^{i} u_{e_j}}$. By setting $x'_{e_j} := u_{e_j}$ for $j \in \{1, \dots, i\}$ and $x'_{e_l} := x_{e_l}$ for $l \in \{i+1, \dots, k\}$, the dynamic capacity constraint of each edge e_j is fulfilled for x' since

$$\begin{aligned}\alpha_{e_j} \cdot F = \alpha_{e_j} \cdot \frac{1}{1-\alpha^{(i)}} \cdot \sum_{l=1}^{i} x_{e_l} &= \frac{\alpha_{e_j}}{u_{e_j}} \cdot u_{e_j} \cdot \frac{1}{1-\alpha^{(i)}} \cdot \sum_{l=1}^{i} u_{e_l} \\ &\geqslant \frac{\alpha_{e_i}}{u_{e_i}} \cdot u_{e_j} \cdot \frac{1}{1-\alpha^{(i)}} \cdot \sum_{l=1}^{i} u_{e_l} \geqslant u_{e_j} = x'_{e_j}.\end{aligned}$$

Thus, by maximality of x, we must have $x_{e_j} = u_{e_j}$ for $j \in \{1, \dots, i\}$. In total, by setting $h := i$, we can label each edge e_j with $j \in \{1, \dots, h\}$ with type u and each edge e_l for $l \in \{h+1, \dots, k\}$ with type α, which shows the claim. □

Corollary 6.19:
Let v and w be two nodes such that all edges $\{e_1, \dots, e_k\}$ that leave v are parallel edges heading to w, i.e., $\delta^+(v) = E(v, w)$, and assume that the edges are ordered such that $\frac{u_{e_i}}{\alpha_{e_i}} \leqslant \frac{u_{e_j}}{\alpha_{e_j}}$ for $i < j$. The maximum flow between v and w can be found in $\mathcal{O}(k)$ time.

Proof: According to Lemma 6.18, there exists a maximum flow x and an index h such that each edge e_i with $1 \leqslant i \leqslant h$ is of type u and each edge e_j with $h+1 \leqslant j \leqslant k$ is of type α in x and this index h can be computed in $\mathcal{O}(k)$ time. The flow x_{e_i} on the edges e_i is consequently given by $x_{e_i} := u_{e_i}$ for $1 \leqslant i \leqslant h$. Since a fixed fraction $\alpha^{(h)} = \sum_{j=h+1}^{k} \alpha_{e_j}$ of the total flow is sent along the edges e_j for $h+1 \leqslant j \leqslant k$, the total flow value is given by $F := \frac{1}{1-\alpha^{(h)}} \cdot \sum_{i=1}^{h} u_{e_i}$ and the flow on each edge e_j amounts to $x_{e_j} := \alpha_{e_j} \cdot F$. □

As seen in Lemma 6.18, there is some index h such that all edges e_i with $1 \leqslant i \leqslant h$ are of type u and the remaining edges are of type α in a maximum flow. In the following lemma, we show that the converse is true as well. In particular, this shows that the maximum flow is unique. Note that this lemma considers the more general case, in which the edges in $\delta^+(v)$ are not assumed to be necessarily parallel:

Lemma 6.20:
Let v be a node such that, for some given feasible flow x, at least one of the outgoing edges $\delta^+(v)$ of v is of type u while the rest of the edges is of type α. Then the flow on the edges in $\delta^+(v)$ is unique and maximum.

Proof: As in Lemma 6.18, assume that the edges $\{e_1, \dots, e_k\}$ are ordered such that $\frac{u_{e_i}}{\alpha_{e_i}} \leqslant \frac{u_{e_j}}{\alpha_{e_j}}$ for $i < j$. Since all edges in $\delta^+(v)$ are either of type u or of type α in the flow x, the same arguments that were used in the proof of Lemma 6.18 show that there is some index $h \in \{1, \dots, k\}$ with $\alpha_{e_h} \geqslant (1 - \alpha^{(h)}) \cdot \frac{u_{e_h}}{\sum_{l=1}^{h} u_{e_l}}$ such that none of the edges e_j for $j \in \{h+1, \dots, k\}$ can be of type u and is, thus, of type α. Furthermore, since $\frac{u_{e_i}}{\alpha_{e_i}} \leqslant \frac{u_{e_h}}{\alpha_{e_h}}$ for each $i \in \{1, \dots, h\}$, we get that

$$\alpha_{e_i} \geqslant \frac{u_{e_i}}{u_{e_h}} \cdot \alpha_{e_h} \geqslant \frac{u_{e_i}}{u_{e_h}} \cdot (1 - \alpha^{(h)}) \cdot \frac{u_{e_h}}{\sum_{l=1}^{h} u_{e_l}} = (1 - \alpha^{(h)}) \cdot \frac{u_{e_i}}{\sum_{l=1}^{h} u_{e_l}}.$$

Let $I_\alpha := \{i \in \{1, \dots, h\} : e_i \text{ is of type } \alpha\}$ and $I_u := \{i \in \{1, \dots, h\} : e_i \text{ is of type } u\}$, where $I_u \neq \emptyset$ by assumption. The flow value F out of node v is then given by

$$F = \sum_{i=1}^{k} x_{e_i} = \sum_{j=h+1}^{k} \alpha_{e_j} \cdot F + \sum_{i \in I_\alpha} \alpha_{e_i} \cdot F + \sum_{i \in I_u} u_{e_i},$$

which is equivalent to

$$\begin{aligned}
F &= \frac{1}{1 - \alpha^{(h)} - \sum_{i \in I_\alpha} \alpha_{e_i}} \cdot \sum_{i \in I_u} u_{e_i} \\
&\geqslant \frac{1}{1 - \alpha^{(h)} - \sum_{i \in I_\alpha} (1 - \alpha^{(h)}) \cdot \frac{u_{e_i}}{\sum_{l=1}^{h} u_{e_l}}} \cdot \sum_{i \in I_u} u_{e_i} \\
&= \frac{1}{(1 - \alpha^{(h)}) \cdot \left(1 - \frac{\sum_{i \in I_\alpha} u_{e_i}}{\sum_{l=1}^{h} u_{e_l}}\right)} \cdot \sum_{i \in I_u} u_{e_i} \\
&= \frac{1}{(1 - \alpha^{(h)}) \cdot \frac{\sum_{i \in I_u} u_{e_i}}{\sum_{l=1}^{h} u_{e_l}}} \cdot \sum_{i \in I_u} u_{e_i} = \frac{1}{1 - \alpha^{(h)}} \cdot \sum_{l=1}^{h} u_{e_l}.
\end{aligned}$$

Note that this lower bound on the flow value equals the flow value that is given by setting $x_{e_i} := u_{e_i}$ for each $i \in \{1, \dots, h\}$. Thus, since the given flow is feasible, each of the edges e_i for $i \in \{1, \dots, h\}$ must be of type u, which is clearly maximum and shows the claim. □

6.5.1 Augmenting on Flow Distribution Schemes

In this subsection, we describe an algorithm that repeatedly sends flow on flow distribution schemes with positive residual capacity, i.e., on which a given flow can be increased without violating any capacity constraint. This algorithm is similar to the well-known augmenting path algorithm for the traditional maximum flow problem, but generalizes the procedure from augmentations on single paths to augmentations on (basic) flow distribution schemes. As it turns out, it is possible to augment flow in a greedy manner without the need to use some form of residual network in order to revert prior decisions, which allows us to obtain a strongly polynomial-time algorithm for the problem on series-parallel graphs that runs in $\mathcal{O}(m^2)$ time. Note that a similar result is known for the traditional maximum flow and minimum cost flow problem in series-parallel graphs (Bein et al., 1985).

The result will be established in four steps: We first show that we can find a suitable starting solution efficiently (Lemma 6.21). We then define a measurement α_x that allows us to evaluate easily if or if not there is an augmenting flow distribution scheme (Lemma 6.22). In a next step, we prove that the procedure terminates within $2m$ augmentations (Corollary 6.28). Finally, we show that the resulting flow is in fact maximal (Theorem 6.31).

As a starting flow for our algorithm, we use a flow that is positive on each edge. Such a flow can be found in linear time even on general acyclic graphs, as the following lemma shows:

Lemma 6.21:
Let G be an acyclic graph. In $\mathcal{O}(m)$ time, we can compute a feasible flow x that is positive on each edge and that fulfills the property that, for each $v \in V \setminus \{t\}$, either all or no edges in $\delta^+(v)$ are of type α and no edge is of type u.

Proof: For each $e = (v, w) \in E$, let $\beta_e := \frac{\alpha_e}{\sum_{e' \in \delta^+(v)} \alpha_{e'}}$. Clearly, β is a feasible flow distribution scheme that assigns a positive value β_e to each edge. Moreover, for each $v \in V \setminus \{t\}$, if $\sum_{e' \in \delta^+(v)} \alpha_{e'} = 1$, then $\beta_e = \alpha_e$ for all $e \in \delta^+(v)$ (so all edges in $\delta^+(v)$ are of type α in any flow on β). Otherwise, $\sum_{e' \in \delta^+(v)} \alpha_{e'} > 1$ and we obtain that $\beta_e < \alpha_e$ for each $e \in \delta^+(v)$ (so no edge in $\delta^+(v)$ is of type α in any flow on β). According to Lemma 6.11, we can compute the corresponding (positive) weights $w_\beta(e)$ of all edges in $\mathcal{O}(m)$ time. Using these weights, we obtain a feasible flow x with the desired properties by setting $x_e := w_\beta(e) \cdot F$ for $0 < F < \min\left\{\frac{u_e}{w_\beta(e)} : e \in E\right\}$. □

After sending a small amount of flow that is positive on each edge as described in Lemma 6.21, we may assume in the following that the values α_e of all outgoing edges

of type α sum up to at most one at each node, which may not be true for the zero-flow[2].

In the following, for a given series-parallel graph G and a feasible flow x in G, we call a flow distribution scheme β *augmenting* if we can increase x by adding some flow x' on β of positive flow value without violating any static or dynamic capacity constraint. Clearly, if we can find an augmenting flow distribution scheme β, the flow x cannot be optimal since we are able to increase the flow value by sending flow on β. As we will see in the following, the reverse is true as well, i.e., as soon as there is no further augmenting flow distribution scheme, the flow is maximum.

Given a flow x, we start by defining a function $\alpha_x(G')$ that measures the maximum fraction of augmenting flow that can be sent through each subgraph G' corresponding to a node in the decomposition tree of the given series-parallel network G:

$$\alpha_x(G' = e) = \begin{cases} 0, & \text{if } e \text{ is of type } u \text{ in } x, \\ \alpha_e, & \text{if } e \text{ is of type } \alpha \text{ in } x, \\ 1, & \text{else.} \end{cases} \tag{6.4a}$$

$$\alpha_x(G' = G_1 \mid G_2) = \min\{1, \alpha_x(G_1) + \alpha_x(G_2)\}. \tag{6.4b}$$

$$\alpha_x(G' = G_1 \circ G_2) = \begin{cases} \alpha_x(G_1), & \text{if } \alpha_x(G_2) = 1, \\ 0, & \text{else.} \end{cases} \tag{6.4c}$$

Clearly, all values $\alpha_x(G')$ can be computed in $\mathcal{O}(m)$ time using a dynamic programming scheme on the decomposition tree of G. The following lemma and the resulting corollary show that we can use $\alpha_x(G)$ in order to decide whether an augmenting flow distribution scheme exists:

Lemma 6.22:
For a given flow x in a series-parallel graph G, it holds that $\alpha_x(G') = q$ for a given subgraph G' corresponding to a node in the composition tree of G and for some $q \in [0, 1]$ if and only if q is the maximum value in $[0, 1]$ such that a fraction q of a sufficiently small amount of additional flow that arrives at the source node of G' can be sent through G'.

Proof: We show the claim in a bottom-up manner by induction on the decomposition tree of G. Consider a leaf of the tree that corresponds to an edge $e = (v, w)$. Obviously, if e is of type u, the flow on e cannot be increased. If the edge is of type α, at most a

2 Clearly, this does not imply that the values α_e of *all* outgoing edges of some node v sum up to at most one, in contrast to the case of traditional processing networks.

fraction α_e of additional flow that arrives at v can be sent through e without violating the dynamic capacity constraint of the edge. Otherwise, if neither of the two capacities of e is reached, we can send all of the additional flow that arrives at v to w using edge e until e becomes of type u or type α. This behavior is modeled by equation (6.4a).

Now assume that G' with source s' is the parallel composition of the two series-parallel components G_1 and G_2, which can carry at most a fraction $\alpha_x(G_1) =: q_1$ and $\alpha_x(G_2) =: q_2$ of additional flow at s' by the induction hypothesis, respectively. Since flow that it sent through G' splits into two fractions that pass G_1 and G_2, respectively, the fraction of additional flow that can flow through G' amounts to at most $\min\{1, q_1 + q_2\}$, as given by equation (6.4b).

Finally, let G' with source s' be the series composition of two components G_1 and G_2 with common node v. Assume that G' can carry a positive fraction $q > 0$ of additional flow that arrives at s'. This is possible if and only if G_1 can carry a fraction of q (i.e., if $\alpha_x(G_1) \geqslant q$) and, since all the edges in $\delta^+(v)$ are already contained in G_2, the component G_2 can carry all the flow that arrives at v (i.e., if $\alpha_x(G_2) = 1$). Thus, according to equation (6.4c), $\alpha_x(G') = q$ in this case. Similarly, it holds that $\alpha_x(G') = 0$ if and only if $\alpha_x(G_1) = 0$ or $\alpha_x(G_2) < 1$, which is true if and only if G_1 cannot carry flow at all or G_2 cannot carry all of the flow that arrives at v, i.e., we cannot send a fraction of additional flow that arrives at s' through G'. □

Corollary 6.23:
There exists an augmenting flow distribution scheme in a series-parallel graph G with a given flow x if and only if $\alpha_x(G) = 1$. □

In the following, for a fixed decomposition tree of the series-parallel graph G, we define a total order $\prec$ on the edges with $e \prec e'$ if and only if there is an inner node in the decomposition tree such that e is reachable via the left and e' is reachable via the right child of the inner node. For example, in Figure 2.1 on page 11, it holds that $e_1 \prec e_3$ and that $e_4 \prec e_6$, but not that $e_4 \prec e_2$.

Lemma 6.24:
For a given flow x in a series-parallel graph G satisfying $\alpha_x(G) = 1$, it is possible to find an augmenting (basic) flow distribution scheme in $\mathcal{O}(m)$ time.

Proof: We show that we can recursively compute a *partial* augmenting basic flow distribution scheme $\beta(G', q)$ for each series-parallel graph G' corresponding to a node in the decomposition tree of G with source s' in $\mathcal{O}(m)$ time. This flow distribution scheme describes how a fraction q with $0 \leqslant q \leqslant \alpha_x(G')$ of the flow that leaves s' is routed through G'. Evaluating $\beta(G, 1)$ then gives the desired flow distribution scheme in $\mathcal{O}(m)$ time.

For a component consisting of a single edge $e = (s', t')$, the only possible partial flow distribution scheme that fulfills the required properties is given by $\beta_e := q$. Now let G' be the series composition of two series-parallel graphs G_1 and G_2. Similar to the definition of α_x in equation (6.4c), we need to evaluate both $\beta(G_1, q)$ and $\beta(G_2, 1)$ in order to distribute a fraction of q of the flow arriving at s' through G_1 and G_2. Finally, let G' be the parallel composition of the two series-parallel graphs G_1 and G_2 where G_1 is the left child of the inner node that corresponds to G' in the decomposition tree of G. We distribute the fraction q to both components by evaluating $\beta(G_1, q_1)$ and $\beta(G_2, q_2)$ with $q_1 := \min\{q, \alpha_x(G_1)\}$ and $q_2 := q - q_1$. It is now easy to see that starting the recursive procedure from the root of the decomposition tree by evaluating $\beta(G, 1)$ yields the desired augmenting flow distribution scheme in $\mathcal{O}(m)$ time. □

In summary, for a given flow x in a series-parallel graph G, we are able to determine in $\mathcal{O}(m)$ time if the flow x can be improved by sending additional flow on some flow distribution scheme. If so, we can obtain such a flow distribution scheme within the same running-time. It remains to show that we can incorporate these results into a general procedure that yields a maximum flow within $\mathcal{O}(m)$ iterations.

In the following, for a feasible flow x, we refer to an edge e as *dead* if e is of type u or if there is a series-parallel subgraph G' that corresponds to an inner node in the decomposition tree, contains edge e, and satisfies $\alpha_x(G') = 0$. Note that a dead edge remains dead after an augmentation over any flow distribution scheme as described in the proof of Lemma 6.24.

Lemma 6.25:
For a flow x in a series-parallel graph G' with source s' that satisfies $\alpha_x(G') < 1$, let β denote the flow distribution scheme that is obtained by evaluating $\beta(G', \alpha_x(G'))$, where the function $\beta(\cdot, \cdot)$ is defined as in the proof of Lemma 6.24. It then holds that $\beta_e = \alpha_e$ for each $e \in \delta^+(s')$ that is of type α, and that $\beta_e = 0$ for each dead edge $e \in \delta^+(s')$. Moreover, there is no $e \in \delta^+(s')$ that is neither of type α nor dead.

Proof: If G' consists of a single edge $e = (s', t')$, the claim clearly follows according to the definition of β. Now let G' denote a series composition of two series-parallel subgraphs G_1 and G_2. If $\alpha_x(G') = 0$, it holds that all edges in G_1 and, thus, all edges in $\delta^+(s')$ are dead by definition. Since we obtain β_e for each $e \in \delta^+(s')$ by evaluating $\beta(G_1, \alpha_x(G')) = \beta(G_1, 0)$, the claim then follows. Otherwise, if $\alpha_x(G') \in (0, 1)$, it holds that $\alpha_x(G_1) = \alpha_x(G')$ by equation (6.4c) such that the values β_e for all edges in $\delta^+(s')$ are obtained by evaluating $\beta(G_1, \alpha_x(G_1))$ and the claim follows by induction. Finally, if G' is the parallel composition of two series-parallel subgraphs G_1 and G_2, it must hold that $\alpha_x(G_1) + \alpha_x(G_2) < 1$ and, thus, that $\alpha_x(G_1) < 1$ and $\alpha_x(G_2) < 1$. The

claim then follows by induction since the values β_e for all $e \in \delta^+(s')$ are obtained by evaluating $\beta(G_1, \alpha_x(G_1))$ and $\beta(G_2, \alpha_x(G_2))$. □

Lemma 6.26:
Let x denote a flow in a series-parallel graph G' that satisfies $\alpha_x(G') = 1$ and let β be an augmenting basic flow distribution scheme as described in the proof of Lemma 6.24. If $\beta_e > 0$ for some edge $e = (v, w)$, then, for each $e' \in \delta^+(v)$ with $e' \prec e$, it either holds that e' is dead and $\beta_{e'} = 0$ or that e' is of type α and $\beta_{e'} = \alpha_{e'}$.

Proof: Let e' and e with $e' \prec e$ be defined as above. Clearly, there is an inner node in the decomposition tree that corresponds to a parallel composition of two series-parallel subgraphs G_1 and G_2 with e' in G_1 and e in G_2. Let $0 < q \leqslant 1$ denote the fraction of additional flow that was distributed in two quantities $q_1 := \min\{q, \alpha_x(G_1)\}$ and $q_2 := q - q_1$ among G_1 and G_2, respectively, in the construction process of β as described in the proof of Lemma 6.24. Since $\beta_e > 0$, it holds that $q_2 > 0$ and, hence, that $\alpha_x(G_1) = q_1 < 1$. The claim then follows by Lemma 6.25. □

Our augmenting flow distribution scheme algorithm for computing a maximum flow in a series-parallel generalized processing network works as follows: After computing the initial flow x as described in the proof of Lemma 6.21, we compute the value $\alpha_x(G)$ at the beginning of each iteration in $\mathcal{O}(m)$ time. If this value is less than one, we terminate and return the current flow x (we will show later that the flow is then optimum). Otherwise, we compute an augmenting basic flow distribution scheme β as described in the proof of Lemma 6.24. According to Lemma 6.11, we can determine the weight $w_\beta(e)$ of each edge $e \in E$ in $\mathcal{O}(m)$ time. The maximum amount of flow that can be sent on β is then given by the largest value $\delta > 0$ such that the flow $x(\delta)$ with $(x(\delta))_e := x_e + \delta \cdot w_\beta(e)$ is feasible. To be more precise, the value of δ is given by $\delta := \min\{\delta_1, \delta_2\}$ with

$$\begin{aligned}\delta_1 &:= \max\{\delta : (x(\delta))_e \leqslant u_e \ \ \forall e \in E\} \\ &= \min\left\{\frac{u_e - x_e}{w_\beta(e)} : e \in E \text{ with } w_\beta(e) > 0\right\}\end{aligned}$$

and

$$\begin{aligned}\delta_2 &:= \max\left\{\delta : (x(\delta))_e \leqslant \alpha_e \cdot \sum_{e' \in \delta^+(v)} (x(\delta))_{e'} \ \ \forall e = (v, w) \in E\right\} \\ &= \max\left\{\delta : x_e + \delta \cdot w_\beta(e) \leqslant \alpha_e \cdot \sum_{e' \in \delta^+(v)} (x_{e'} + \delta \cdot w_\beta(e')) \ \forall e = (v, w)\right\}\end{aligned}$$

$$= \max \left\{ \delta : \delta \cdot \left(w_\beta(e) - \alpha_e \cdot \sum_{e' \in \delta^+(v)} w_\beta(e') \right) \leqslant \left(\alpha_e \cdot \sum_{e' \in \delta^+(v)} x_{e'} \right) - x_e \ \ \forall e = (v,w) \in E \right\}$$

$$= \min \left\{ \frac{\left(\alpha_e \cdot \sum_{e' \in \delta^+(v)} x_{e'} \right) - x_e}{w_\beta(e) - \alpha_e \cdot \sum_{e' \in \delta^+(v)} w_\beta(e')} : e = (v,w) \in E \text{ with } \beta_e > \alpha_e \right\},$$

where the last equality follows from the fact that only those edges $e = (v,w) \in E$ restrict δ for which $w_\beta(e) - \alpha_e \cdot \sum_{e' \in \delta^+(v)} w_\beta(e') > 0$, which is true if and only if $\beta_e > \alpha_e$ since $w_\beta(e)$ is proportional to β_e and $\sum_{e' \in \delta^+(v)} \beta_{e'} = 1$.

Note that, for the value of δ determined above, at least one edge becomes of type α or of type u that was not of this type in x. We show in the following that at the same time it cannot happen that an edge that was dead or of type α before will be neither dead nor of type α after the augmentation. If we were not able to guarantee this, our procedure would not be guaranteed to terminate within a finite number of augmentations.

Lemma 6.27:
Let x' denote the flow that is obtained after augmenting a flow x with $\alpha_x(G) = 1$ over a flow distribution scheme β as described above and let $\mathcal{D}(x)$ ($\mathcal{D}(x')$) and $\mathcal{A}(x)$ ($\mathcal{A}(x')$) denote the set of dead edges and α-edges in x (x'), respectively. It then holds that $\mathcal{D}(x') \cup \mathcal{A}(x') \supsetneq \mathcal{D}(x) \cup \mathcal{A}(x)$ or that $\mathcal{D}(x') \cup \mathcal{A}(x') = \mathcal{D}(x) \cup \mathcal{A}(x)$ and $\mathcal{D}(x') \supsetneq \mathcal{D}(x)$.

Proof: First, consider some node $v \in V \setminus \{t\}$ and let $e \in \delta^+(v)$ denote the unique edge with $\beta_e > 0$ and $e' \prec e$ for each $e' \in \delta^+(v) \setminus \{e\}$ with $\beta_{e'} > 0$. According to Lemma 6.26, it holds that every such edge e' is either dead or of type α and remains so after an augmentation of value $\delta' < \delta$, where $\delta := \min\{\delta_1, \delta_2\}$ is defined as above. Moreover, if edge e was of type α as well before the augmentation, it holds that $\beta_e = 1 - \sum_{e' \in \delta^+(v) : e' \prec e} \beta_{e'} = 1 - \sum_{e' \in \delta^+(v) : e' \prec e, e' \in \mathcal{A}(x)} \alpha_{e'} = \alpha_e$ and e remains of type α. So, as long as we send less than δ units of flow on β, each edge remains its type.

By sending δ units of flow over β as described above, one of the following cases applies: If $\delta = \delta_1$, some edge becomes of type u in x' that was either of type α or of no type in x. In the first case, it holds that $\mathcal{D}(x') \cup \mathcal{A}(x') = \mathcal{D}(x) \cup \mathcal{A}(x)$ and $\mathcal{D}(x') \supsetneq \mathcal{D}(x)$, while in the second case we get that $\mathcal{D}(x') \cup \mathcal{A}(x') \supsetneq \mathcal{D}(x) \cup \mathcal{A}(x)$. On the other side, if $\delta = \delta_2$, some edge that was of no type in x becomes of type α in x', which implies that $\mathcal{D}(x') \cup \mathcal{A}(x') \supsetneq \mathcal{D}(x) \cup \mathcal{A}(x)$. This shows the claim. □

Corollary 6.28:
The augmenting flow distribution scheme algorithm terminates after $\mathcal{O}(m)$ augmentations and runs in $\mathcal{O}(m^2)$ time.

Proof: The claim directly follows from the discussion before and from the combination of Lemma 6.27 and the fact that $|\mathcal{D}(x)| \leqslant m$ and $|\mathcal{A}(x)| \leqslant m$ for any flow x. □

It remains to show that the computed flow is maximum. To this end, we need the following lemma:

Lemma 6.29:
Let G be a series-parallel graph and x be a flow in G that is positive on each edge $e \in E$. Moreover, assume that $\alpha_x(G) < 1$. Then there is an s-t-cut (S, T) such that

1. each edge in $\delta^+(S)$ is either of type α or of type u,
2. for each node $v \in V$ with $\emptyset \neq \delta^+(v) \subseteq \delta^+(S)$, it holds that at least one edge in $\delta^+(v)$ is of type u, and
3. $\delta^-(S) = \emptyset$.

Proof: Let x be a flow in a series-parallel graph G that fulfills $\alpha_x(G) < 1$. Consider the following function c_x defined on series-parallel subgraphs G' corresponding to nodes in the decomposition tree of G:

$$
\begin{aligned}
c_x(G' = e) &= \{e\},\\
c_x(G' = G_1 \mid G_2) &= c_x(G_1) \cup c_x(G_2),\\
c_x(G' = G_1 \circ G_2) &= \begin{cases} c_x(G_2) & \text{if } \alpha_x(G_2) < 1,\\ c_x(G_1) & \text{else.} \end{cases}
\end{aligned}
$$

We now show that the set $c_x(G)$ contains exactly the edges in an s-t-cut that fulfills the required properties. First, suppose that G consists of a single edge e. Since we are looking for an s-t-cut of G, the only possible cut is given by $\{e\} = c_x(G)$. Note that, since $\alpha_x(G) < 1$, edge e must be either of type u or of type α. Now assume that G is the series composition of two series-parallel graphs G_1 and G_2. Since $\alpha_x(G) < 1$, it holds that either $\alpha_x(G_2) < 1$ or that $\alpha_x(G_2) = 1$ but $\alpha_x(G_1) < 1$ (cf. equation (6.4c)). In the first (second) case, we set $c_x(G) := c_x(G_2)$ ($c_x(G) := c_x(G_1)$) and proceed recursively. Finally, suppose that G is the parallel composition of the two series-parallel graphs G_1 and G_2. Since each s-t-cut of G must pass both G_1 and G_2, we set $c_x(G) := c_x(G_1) \cup c_x(G_2)$.

Note that all the series-parallel subgraphs G' in the decomposition tree of G that contain edges in $c_x(G)$ fulfill $\alpha_x(G') < 1$. Hence, by evaluating $c_x(G)$ as described above, we obtain an s-t-cut (S, T) that only consists of edges of type α and type u, which shows claim (1). Now suppose that there is a node $v \in V$ with $\delta^+(v) \subseteq \delta^+(S)$ such that each edge in $\delta^+(v)$ is of type α. Since $x_e > 0$ for each $e \in E$, this implies that $\sum_{e \in \delta^+(v)} \alpha_e = 1$. Let G' denote the inclusionwise minimal series-parallel subgraph of G that contains all edges in $\delta^+(v)$ and corresponds to an inner node in the decomposition tree of G. For every series composition $G'' = G_1 \circ G_2$ that is contained in the decomposition tree of G', since $\delta^+(v) \subseteq \delta^+(S)$, it must hold that $\alpha_x(G_2) = 1$ and $\alpha_x(G_1) < 1$ and, thus, that $\alpha_x(G'') = \alpha_x(G_1)$. But then, according to the definition of c_x and α_x, it finally holds that $\alpha_x(G') = \sum_{e \in \delta^+(v)} \alpha_e = 1$, which contradicts the fact that $\alpha_x(G') < 1$ for all series-parallel subgraphs G' corresponding to an inner node in the decomposition tree of G.

Finally, the third claim follows from the fact that the graph G is acyclic and that, for each s-t-path P in G, the set $c_x(G)$ only contains one edge in P by construction. Hence, the cut that is implied by $c_x(G)$ fulfills all of the required properties and the claim of the lemma follows. □

Lemma 6.30:
Let G be a series-parallel graph and let (S, T) denote an s-t-cut in G. Then there exists a node $v \in S$ with $\delta^+(v) \subseteq \delta^+(S)$.

Proof: Let $S' \subseteq S$ denote the set of all nodes in S that are reachable from s via a (possibly empty) directed path using nodes in S only. Since $s \in S'$, the set is nonempty. For a given topological sorting of the nodes in G, let $v \in S'$ denote the node in S' with the highest index in the topological sorting. Since $v \neq t$, it holds that $\delta^+(v) \neq \emptyset$. However, for each $e = (v, w) \in \delta^+(v)$, if e was not contained in $\delta^+(S)$, it would hold that $w \in S$ and, thus, that $w \in S'$. However, this would contradict the definition of v, which shows the claim. □

We are now ready to prove the main theorem of this subsection:

Theorem 6.31:
The augmenting flow distribution scheme algorithm computes a maximum flow in a series-parallel generalized processing network in $\mathcal{O}(m^2)$ time.

Proof: The claimed running time follows from the above arguments. It remains to show that the computed flow x is maximum.

First, consider some arbitrary flow x' and the s-t-cut (S, T) obtained from applying Lemma 6.29 to the flow x computed by the algorithm. Using that $\text{excess}_{x'}(v) = 0$ for each $v \in V \setminus \{s, t\}$ according to equation (6.1b), we get that

$$\begin{aligned}\text{val}(x') = \text{excess}_x(t) = \sum_{v \in T} \text{excess}_{x'}(v) = \sum_{v \in T} \left(\sum_{e \in \delta^-(v)} x'_e - \sum_{e \in \delta^+(v)} x'_e \right) \\ = \sum_{v \in \delta^-(T)} x'_e - \sum_{v \in \delta^+(T)} x'_e = \sum_{v \in \delta^+(S)} x'_e - \sum_{v \in \delta^-(S)} x'_e \leqslant \sum_{v \in \delta^+(S)} x'_e,\end{aligned}$$

i.e., the flow value of each flow x' is bounded by the total flow value on edges that head from S to T. We now show that the flow x that is computed by the augmenting flow distribution scheme algorithm fulfills $\text{val}(x) = \sum_{v \in \delta^+(S)} x_e$ and maximizes this value among all feasible flows, which shows the claim.

The first claim follows directly from the construction of the cut since there are no edges in $\delta^-(S)$. Now assume that there is a feasible flow x' with $\text{val}(x') > \text{val}(x)$. Clearly, there must be at least one edge in $\delta^+(S)$ for which $x'_e > x_e$. According to Lemma 6.30, there is at least one node $v \in S$ with $\delta^+(v) \subseteq \delta^+(S)$. Moreover, according to Lemma 6.29, it holds that each of the edges in $\delta^+(v)$ is of type α or of type u in x and that at least one of these edges is of type u, which, according to Lemma 6.20, implies that the flow leaving v is maximum and unique and, thus, the flow on each of the edges in $\delta^+(v)$ is maximum. Thus, it holds that $x'_e \leqslant x_e$ for each $e \in \delta^+(v)$. If $v = s$, it additionally holds that $\text{val}(x') = \sum_{e \in \delta^+(s)} x'_e \leqslant \sum_{e \in \delta^+(s)} x_e = \text{val}(x)$ and the claim of the theorem follows. Otherwise, v results from merging the sink of G_1 with the source of G_2 in a series composition of two series-parallel subgraphs G_1 and G_2. Let $\overline{G}$ denote the graph that results from G by replacing $G_1 \circ G_2$ with a single edge $\overline{e}$ with capacity $u_{\overline{e}} := \sum_{e \in \delta^+(v)} x_e$. Clearly, the flow $\overline{x}$ with $\overline{x}_{\overline{e}} := u_{\overline{e}}$ and $\overline{x}_e := x_e$ for each $e \in E(\overline{G}) \setminus \{\overline{e}\}$ is feasible in $\overline{G}$. Since there is a flow x' in G with $\text{val}(x') > \text{val}(x)$ and $x'_e \leqslant x_e$ for each $e \in \delta^+(v)$, there must be a flow $\overline{x}'$ with $\text{val}(\overline{x}') > \text{val}(\overline{x})$ in $\overline{G}$. However, since the new edge $\overline{e}$ is of type u in $\overline{x}$ and is contained in the cut $(\overline{S}, \overline{T})$ with $\overline{S} := S \cap V(\overline{G})$ and $\overline{T} := T \cap V(\overline{G})$, we can repeat the above arguments until $S = \{s\}$ and $\delta^+(s) = \delta^+(S)$. Thus, there is no flow x' in G with $\text{val}(x') > \text{val}(x)$ and the claim follows. □

6.5.2 A Faster Approach

Lemma 6.18 and Corollary 6.19 already give insights about the behavior of a special case of series-parallel graphs, namely the case of parallel edges between two nodes v and w. We now show how to incorporate the behavior of other edges in $\delta^+(v)$ that do

not reach w into the above algorithm. Until the end of the following discussion, we again assume that the edges in $E(v,w) = \{e_1, \dots, e_k\}$ are ordered such that $\frac{u_{e_i}}{\alpha_{e_i}} \leqslant \frac{u_{e_j}}{\alpha_{e_j}}$ for $i < j$.

Consider the set $E(v,w) := \{e_1, \dots, e_k\}$ of parallel edges between two nodes v and w and assume that the total flow on the remaining edges $e \in \delta^+(v) \setminus E(v,w)$ that leave v is given by x_0. For a fixed value of x_0, we can find the maximum flow on the edges in $E(v,w)$ by using the algorithm described in the proof of Corollary 6.19 as follows: By introducing an artificial edge e_0 between v and w with capacity $u_{e_0} := x_0$ and $\alpha_{e_0} := 1$ and evaluating the algorithm on the graph $G' := (\{v,w\}, E(v,w) \cup \{e_0\})$, edge e_0 will eventually be labeled as type u since $\alpha_{e_0} = 1$ (so the second case must hold in the proof of Lemma 6.18 when we reach edge e_0) and will, thus, carry the desired amount of flow x_0 while $\sum_{e \in E(v,w)} x_e$ is maximum according to Corollary 6.19.

Now let x_0 be of variable value. Note that, for each $e_i \in E(v,w)$, there is some (not necessarily positive) value $b(e_i)$ such that edge e_i is declared to be of type α for $x_0 < b(e_i)$ and is labeled as type u for $x_0 \geqslant b(e_i)$. We call this value $b(e_i)$ the *breakpoint of edge* e_i in the following. Clearly, according to the proof of Lemma 6.18, it holds that

$$\begin{aligned} x_0 = b(e_i) \qquad &\iff \qquad \alpha_{e_i} = \left(1 - \alpha^{(i)}\right) \cdot \frac{u_{e_i}}{x_0 + \sum_{j=1}^{i} u_{e_j}} \\ &\iff \qquad x_0 = \left(1 - \alpha^{(i)}\right) \cdot \frac{u_{e_i}}{\alpha_{e_i}} - \sum_{j=1}^{i} u_{e_j} \end{aligned} \tag{6.5}$$

with $\alpha^{(i)} = \sum_{j=i+1}^{k} \alpha_{e_j}$ as before. Using the sorting of the edges in $E(v,w)$, we get the following result:

Lemma 6.32:
Consider a set of edges $E(v,w) = \{e_1, \dots, e_k\}$ with $\frac{u_{e_i}}{\alpha_{e_i}} \leqslant \frac{u_{e_j}}{\alpha_{e_j}}$ for $i < j$ between two nodes $v, w \in V$. For $i \in \{1, \dots, k-1\}$, it holds that $b(e_i) \leqslant b(e_{i+1})$.

Proof: Using the definitions of $b(e_i)$ and $b(e_{i+1})$ and the ordering of the edges in $E(v,w)$, we get that

$$\begin{aligned} b(e_i) &= \left(1 - \alpha^{(i)}\right) \cdot \frac{u_{e_i}}{\alpha_{e_i}} - \sum_{j=1}^{i} u_{e_j} \leqslant \left(1 - \alpha^{(i)}\right) \cdot \frac{u_{e_{i+1}}}{\alpha_{e_{i+1}}} - \sum_{j=1}^{i} u_{e_j} \\ &= \left(1 - \alpha_{e_{i+1}} - \alpha^{(i+1)}\right) \cdot \frac{u_{e_{i+1}}}{\alpha_{e_{i+1}}} - \sum_{j=1}^{i} u_{e_j} \\ &= \left(1 - \alpha^{(i+1)}\right) \cdot \frac{u_{e_{i+1}}}{\alpha_{e_{i+1}}} - u_{e_{i+1}} - \sum_{j=1}^{i} u_{e_j} \end{aligned}$$

$$= \left(1 - \alpha^{(i+1)}\right) \cdot \frac{u_{e_{i+1}}}{\alpha_{e_{i+1}}} - \sum_{j=1}^{i+1} u_{e_j} = b(e_{i+1}),$$

which shows the claim. □

Lemma 6.33:
Let $f_{(v,w)}(x_0)$ denote the maximum flow that can be sent through the parallel edges $E(v,w) = \{e_1, \dots, e_k\}$ depending on the total flow x_0 on the remaining edges in $\delta^+(v) \setminus E(v,w)$. The function $f_{(v,w)}$ is continuous, non-decreasing, concave, and piecewise linear with breakpoints contained in $\{b(e_i) : i \in \{1, \dots, k\}\}$ and non-negative slopes between two adjacent breakpoints. Moreover, it holds that $f_{(v,w)}(x_0) > 0$ for each $x_0 \geqslant 0$. The function can be determined in $\mathcal{O}(k)$ time.

Proof: First consider the case that $x_0 = 0$. According to Lemma 6.18 and 6.20, there is some index h such that, in every maximum flow x between v and w, we have $x_{e_i} = u_{e_i}$ for $1 \leqslant i \leqslant h$ and $x_{e_j} = \alpha_{e_j} \cdot F$ for $h+1 \leqslant j \leqslant k$, where $F = x_0 + \sum_{i=1}^{k} x_{e_i}$ is the total outflow of node v. If $h = k$, the claim clearly follows since, in this case, $f_{(v,w)}$ is a constant function. Else, the partitioning of the edges into the types u and α remains valid until the first breakpoint is reached. According to Lemma 6.32, this breakpoint is given by $b(e_{h+1})$. Note that, for $x_0 \in [0, b(e_{h+1}))$, the maximum flow through the edges in $E(v,w)$ is given by

$$\begin{aligned} f_{(v,w)}(x_0) &= \sum_{i=1}^{k} x_{e_i} = \sum_{i=1}^{h} u_{e_i} + \sum_{j=h+1}^{k} \alpha_{e_j} \cdot F \\ &= \sum_{i=1}^{h} u_{e_i} + \left(f_{(v,w)}(x_0) + x_0\right) \cdot \alpha^{(h)}, \end{aligned}$$

so

$$f_{(v,w)}(x_0) = \frac{1}{1 - \alpha^{(h)}} \cdot \left(\sum_{i=1}^{h} u_{e_i} + x_0 \cdot \alpha^{(h)} \right) > 0, \tag{6.6}$$

which is an increasing linear function of x_0 with slope $\frac{\alpha^{(h)}}{1-\alpha^{(h)}} > 0$. Again, if $h+1 = k$, the claim follows.

Else, as x_0 reaches the value $b(e_{h+1})$, edge e_{h+1} turns from type α to type u and, accordingly, the function $f_{(v,w)}(x_0)$ behaves as a linear function of x_0 until the next breakpoint $b(e_{h+2})$ is reached and so on. The slope of $f_{(v,w)}$ on $[b(e_{h+1}), b(e_{h+2}))$ evaluates to $\frac{\alpha^{(h+1)}}{1-\alpha^{(h+1)}} \leqslant \frac{\alpha^{(h)}}{1-\alpha^{(h)}}$, so the slopes of the linear segments do not increase with x_0.

It remains to show the continuity of $f_{(v,w)}$, which – in combination with the above arguments – in turn yields that the function is concave and non-decreasing. Consider the intersection of the above two adjacent linear segments of $f_{(v,w)}$:

$$\frac{1}{1-\alpha^{(h)}} \cdot \left(\sum_{i=1}^{h} u_{e_i} + x_0 \cdot \alpha^{(h)} \right) = \frac{1}{1-\alpha^{(h+1)}} \cdot \left(\sum_{i=1}^{h+1} u_{e_i} + x_0 \cdot \alpha^{(h+1)} \right).$$

Multiplying by $(1-\alpha^{(h)})$ and $(1-\alpha^{(h+1)})$, we get that

$$\left(1-\alpha^{(h+1)}\right) \cdot \left(\sum_{i=1}^{h} u_{e_i} + x_0 \cdot \alpha^{(h)} \right) = \left(1-\alpha^{(h)}\right) \cdot \left(\sum_{i=1}^{h+1} u_{e_i} + x_0 \cdot \alpha^{(h+1)} \right). \quad (6.7)$$

The left-hand side of equation (6.7) evaluates to

$$\left(1-\alpha^{(h+1)}\right) \cdot \left(\sum_{i=1}^{h} u_{e_i} + x_0 \cdot \alpha^{(h+1)} \right) + \left(1-\alpha^{(h+1)}\right) \cdot x_0 \cdot \alpha_{e_{h+1}},$$

while the right-hand side of equation (6.7) can be rearranged into

$$\left(1-\alpha^{(h+1)}\right) \cdot \left(\sum_{i=1}^{h+1} u_{e_i} + x_0 \cdot \alpha^{(h+1)} \right) - \alpha_{e_{h+1}} \cdot \left(\sum_{i=1}^{h+1} u_{e_i} + x_0 \cdot \alpha^{(h+1)} \right).$$

Hence, subtracting $\left(1-\alpha^{(h+1)}\right) \cdot \left(\sum_{i=1}^{h} u_{e_i} + x_0 \cdot \alpha^{(h+1)}\right)$ from both sides of equation (6.7) and dividing by $\alpha_{e_{h+1}}$ yields

$$\begin{aligned} & \left(1-\alpha^{(h+1)}\right) \cdot x_0 = \left(1-\alpha^{(h+1)}\right) \cdot \frac{u_{e_{h+1}}}{\alpha_{e_{h+1}}} - \left(\sum_{i=1}^{h+1} u_{e_i} + x_0 \cdot \alpha^{(h+1)} \right) \\ \iff \quad & x_0 = \left(1-\alpha^{(h+1)}\right) \cdot \frac{u_{e_{h+1}}}{\alpha_{e_{(h+1)}}} - \sum_{i=1}^{h+1} u_{e_i}, \end{aligned}$$

i.e., both line segments intersect at $x_0 = b(e_{h+1})$. Repeating the above arguments until the last breakpoint is reached then yields the claim. □

Lemma 6.34:
Consider three nodes $v, w, z \in V$ with parallel edges $E(v,w) = \{e_1, \ldots, e_k\}$ between v and w and $\delta^+(w) = \{\overline{e}\}$, where $\overline{e}$ is heading to z. The set of edges $E(v,w) \cup \{\overline{e}\}$ can be replaced by $i \leqslant k$ parallel edges $\{e'_1, \ldots, e'_i\}$ from v to z such that, for each flow value x_0 on the edges in $\delta^+(v) \setminus E(v,w)$, the maximum amount of flow that can arrive at z through the edges $\{e'_1, \ldots, e'_i\}$ equals the maximum amount of flow that can reach z through $E(v,w)$ and $\overline{e}$. This transformation can be performed in $\mathcal{O}(k)$ time.

Proof: Let $g_{(v,z)}(x_0)$ denote the maximum amount of flow that can be sent from v to z using the edges in $E(v,w)$ and the edge $\overline{e}$ depending on the total flow value x_0

on the edges in $\delta^+(v) \setminus E(v,w)$. Clearly, due to the structure of the subgraph that is considered, $g_{(v,z)}(x_0) = \min\{u_{\overline{e}}, f_{(v,w)}(x_0)\}$, where $f_{(v,w)}$ is defined as in Lemma 6.33. If $u_{\overline{e}} \geqslant \max_{x_0 \geqslant 0} f_{(v,w)}(x_0)$, the claim follows by deleting the edge $\overline{e}$ and merging the nodes w and z.

Else, if $u_{\overline{e}} < \max_{x_0 \geqslant 0} f_{(v,w)}(x_0)$, it either holds that $u_{\overline{e}} < f_{(v,w)}(0)$ or there must be two adjacent breakpoints $b(e_{i-1})$ and $b(e_i)$ with $f_{(v,w)}(b(e_{i-1})) \leqslant u_{\overline{e}} < f_{(v,w)}(b(e_i))$ (which are uniquely defined since $f_{(v,w)}$ is non-decreasing and continuous according to Lemma 6.33). In the first case, it clearly holds that $g_{(v,z)}(x_0) = u_{\overline{e}}$ and the claim follows by deleting the edges in $E(v,w)$ and merging the nodes v and w.

Now assume that $f_{(v,w)}(b(e_{i-1})) \leqslant u_{\overline{e}} < f_{(v,w)}(b(e_i))$ for some $i \in \{2,\dots,k\}$ and let $\overline{x}$ denote the (uniquely defined) flow value such that $f_{(v,w)}(\overline{x}) = u_{\overline{e}}$. Note that all of the edges e_j with $i \leqslant j \leqslant k$ remain of type α within the interval $[0,\overline{x}]$ since the respective breakpoints are not yet reached, i.e., a constant fraction $\alpha^{(i-1)} = \sum_{j=i}^{k} \alpha_{e_j}$ of $f_{(v,w)}(x_0) = g_{(v,z)}(x_0)$ flows through the edges $e_i,\dots,e_k$ as long as $x_0 \in [0,\overline{x}]$. Hence, we can replace the edges $e_i,\dots,e_k$ by a single edge e'_i with $\alpha_{e'_i} := \alpha^{(i-1)}$ without changing the behavior of $f_{(v,w)}$ and $g_{(v,z)}$ in $[0,\overline{x}]$. Since e'_i is the only edge of type α within the interval $[b(e_{i-1}),\overline{x}]$, we achieve that $f_{(v,w)}(x_0) = u_{\overline{e}} = g_{(v,z)}(x_0)$ for $x_0 \geqslant \overline{x}$ by setting $u_{e'_i} := u_{\overline{e}} - \sum_{j=1}^{i-1} u_{e_j}$.

Thus, after the transformation, it holds that $f_{(v,w)}(x_0) = g_{(v,z)}(x_0)$ for each $x_0 \geqslant 0$, i.e., edge $\overline{e}$ does not influence the flow value anymore. By deleting the edge $\overline{e}$ and merging the nodes w and z, the claim then follows. □

Note that the edges in $E'(v,w) := \{e'_1,\dots,e'_i\}$ remain ordered by $\frac{u_{e'_j}}{\alpha_{e'_j}}$ after the transformation performed in the proof of Lemma 6.34: For the case that $i = 1$, the claim clearly holds. Otherwise, by construction, it holds that $u_{\overline{e}} \geqslant f_{(v,w)}(b(e_{i-1}))$, so

$$\begin{aligned}
\frac{u_{e'_i}}{\alpha_{e'_i}} = \frac{u_{\overline{e}} - \sum_{j=1}^{i-1} u_{e_j}}{\alpha^{(i-1)}} &\geqslant \frac{f_{(v,w)}(b(e_{i-1})) - \sum_{j=1}^{i-1} u_{e_j}}{\alpha^{(i-1)}} \\
&\overset{(6.6)}{=} \frac{\frac{1}{1-\alpha^{(i-1)}} \cdot \left(\sum_{j=1}^{i-1} u_{e_j} + b(e_{i-1}) \cdot \alpha^{(i-1)}\right) - \sum_{j=1}^{i-1} u_{e_j}}{\alpha^{(i-1)}} \\
&= \frac{\left(\frac{1}{1-\alpha^{(i-1)}} - 1\right) \cdot \sum_{j=1}^{i-1} u_{e_j} + b(e_{i-1}) \cdot \frac{\alpha^{(i-1)}}{1-\alpha^{(i-1)}}}{\alpha^{(i-1)}} \\
&= \frac{\frac{\alpha^{(i-1)}}{1-\alpha^{(i-1)}} \cdot \sum_{j=1}^{i-1} u_{e_j} + b(e_{i-1}) \cdot \frac{\alpha^{(i-1)}}{1-\alpha^{(i-1)}}}{\alpha^{(i-1)}} = \frac{\sum_{j=1}^{i-1} u_{e_j} + b(e_{i-1})}{1-\alpha^{(i-1)}} \\
&\overset{(6.5)}{=} \frac{\sum_{j=1}^{i-1} u_{e_j} + \left(1-\alpha^{(i-1)}\right) \cdot \frac{u_{e_{i-1}}}{\alpha_{e_{i-1}}} - \sum_{j=1}^{i-1} u_{e_j}}{1-\alpha^{(i-1)}} = \frac{u_{e_{i-1}}}{\alpha_{e_{i-1}}},
\end{aligned}$$

where the last inequality follows from Lemma 6.32. Since the transformation assures that $e'_j = e_j$ for $j \in \{1, \dots, i-1\}$, we get the following observation:

Observation 6.35:
After applying the procedure described in Lemma 6.34 to a set of edges $E(v, w) = \{e_1, \dots, e_k\}$ and $\overline{e} = (w, z)$, the resulting edge set $\{e'_1, \dots, e'_i\}$ is ordered by the values $\frac{u_{e'_j}}{\alpha_{e'_j}}$ again. ◁

Lemma 6.36:
Let v and w be two nodes such that all edges $\{e_1, \dots, e_k\}$ that leave v are parallel edges heading to w and let F denote the maximum flow that can be sent from v to w. Then each flow value F' with $0 \leqslant F' \leqslant F$ can be achieved as well.

Proof: The claim follows by a simple scaling argument: Consider the maximum flow x on the edges $e_1, \dots, e_k$ with flow value F that is, e.g., determined using the algorithm described in the proof of Corollary 6.19. It is easy to see that the flow x' with $x'_{e_i} := \frac{F'}{F} \cdot x_{e_i}$ for each $i \in \{1, \dots, k\}$ is feasible as well and achieves a flow value of F'. □

Theorem 6.37:
A maximum flow in a series-parallel graph $G = (V, E)$ can be computed in $\mathcal{O}(m \cdot (n + \log m))$ time.

Proof: Consider a decomposition tree T of G. By sorting the leaves of T in $\mathcal{O}(m \cdot \log m)$ time, we can assure that $\frac{u_{e_i}}{\alpha_{e_i}} \leqslant \frac{u_{e_j}}{\alpha_{e_j}}$ for $i < j$ in each set of edges $\{e_1, \dots, e_k\}$.

In a second step, we use a breadth-first-search in the decomposition tree starting at the root in order to get a list S of all nodes that correspond to series compositions in $\mathcal{O}(m)$ time. Note that this list is inherently sorted by the depth of the respective nodes in the tree.

Let $v \in T$ denote a node in T of maximum depth that corresponds to a series-parallel graph G' that is the series composition of two series-parallel graphs G_1 and G_2 (note that v can be found in $\mathcal{O}(1)$ time by looking at the tail of S). Due to the maximum depth of v, neither G_1 nor G_2 can contain series compositions, i.e., each of the graphs either consists of a single edge or of parallel edges.

Let k_1 and k_2 denote the number of edges contained in G_1 and G_2, respectively. First, consider the case that G_2 consists of parallel-edges $e_1, \dots, e_{k_2}$ with $k_2 \geqslant 2$. Due to the structure of series-parallel graphs, there are no other edges leaving the source node of G_2, so we can find the maximum amount of flow F_2 that can be sent through G_2 in $\mathcal{O}(k_2)$ time according to Corollary 6.19. Consequently, we can replace the edges in G_2 by a single edge with capacity F_2. This in turn enables us to replace the edges in G_1

and G_2 by at most k_1 edges E' in $\mathcal{O}(k_1)$ time using Lemma 6.34. Note that these edges are ordered according to Observation 6.35, but the set $\delta^+(s_{G'})$, where $s_{G'}$ is the source of G', may not be ordered anymore. However, since both the edges in E' and the edges in $\delta^+(s_{G'}) \setminus E'$ are sorted, we can regain the ordering of $\delta^+(s_{G'})$ in $\mathcal{O}(m)$ time.

The algorithm stops when no series composition is left, i.e., the remaining graph consists of parallel edges only. The maximum flow value can then be determined using the procedure described in the proof of Corollary 6.19. Since the maximum flow in each series composition can be computed in $\mathcal{O}(m)$ time as described above, the claimed running time follows. Note that the procedure only describes the computation of the maximum flow value in G. However, using the procedures described in the proofs of Corollary 6.19, Lemma 6.34, and Lemma 6.36, the flow on each edge can be determined as well within the same running time. □

6.6 Integral Flows

In the traditional maximum flow problem, there always exists an integral optimal solution if the capacities are integral (cf. Section 2.4). It is easy to see that this is no longer valid for the case of flows in processing networks. In particular, if we *add* the requirement that the flow on each edge needs to be integral, the maximum flow problem in a generalized processing network becomes both $\mathcal{NP}$-hard to solve and to approximate as we will see in the following two theorems.

Theorem 6.38:
The problem of finding a maximum *integral* flow in a generalized processing network is strongly $\mathcal{NP}$-complete to solve and $\mathcal{NP}$-hard to approximate within constant factors, even if the graph is acyclic and bipartite.

Proof: We first show the $\mathcal{NP}$-completeness of the problem. Clearly, the problem is contained in $\mathcal{NP}$ since we can check if a given solution candidate (which has a polynomially bounded encoding length) is feasible and has a specific flow value in polynomial time. In order to show $\mathcal{NP}$-hardness, we use a reduction from the Exact-CoverBy3Sets-problem, which is known to be strongly $\mathcal{NP}$-complete (cf. (Garey and Johnson, 1979, Problem SP2)):

Instance: A set X with $3q$ elements and a collection $\mathcal{C} = \{C_1, \ldots, C_k\}$ of 3-element subsets of X.

Question: Does there exist a subcollection $\mathcal{C}' \subseteq \mathcal{C}$ such that every element $j \in X$ is contained in exactly one of the subsets in $\mathcal{C}'$?

Given an instance of ExactCoverBy3Sets, we construct a generalized processing network as follows:

We introduce a source s and a sink t as well as nodes v_i and v'_i for each $C_i \in \mathcal{C}$ and a node w_j for each $j \in X$. For each subset $C_i \in \mathcal{C}$, we insert an edge with capacity 3 between s and v_i and three edges, each with flow ratio of $\frac{1}{3}$ between v_i and v'_i. Furthermore, we introduce an edge between v'_i and w_j if $j \in C_i$ and an edge between each node w_j and the sink t. If not mentioned explicitly, we set $\alpha_e = 1$ and $u_e = 1$ for every other edge $e \in E$. The resulting network for $X = \{1, \ldots, 9\}$ and $\mathcal{C} = \{\{1,2,4\},\{2,3,4\},\{3,5,8\},\{4,6,7\},\{6,7,9\}\}$ is shown in Figure 6.2. It is easy to see that the constructed network is always acyclic and bipartite, as claimed.

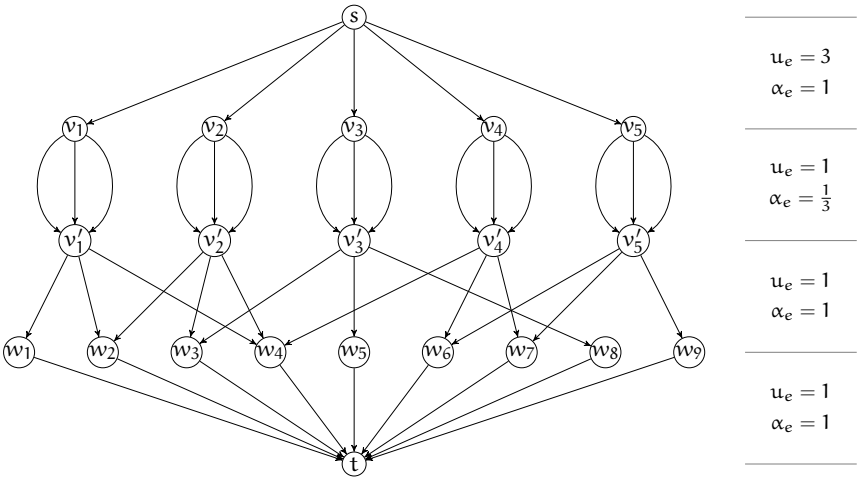


Figure 6.2: The resulting network for a given instance of ExactCoverBy3Sets with $X = \{1, \ldots, 9\}$ and $\mathcal{C} = \{\{1,2,4\},\{2,3,4\},\{3,5,8\},\{4,6,7\},\{6,7,9\}\}$. On the right hand side, the capacities and flow ratios of the edges in each level of the graph are depicted.

We now show that there exists an integral flow x with flow value $\text{val}(x) \geqslant 3q$ if and only if the underlying instance of ExactCoverBy3Sets is a Yes-instance. First assume that there is an integral flow x with flow value $\text{val}(x) \geqslant 3q$. In fact, since there are $3q$ edges leading to the sink, each with a capacity of one, it must hold that $\text{val}(x) = 3q$. Thus, each of these edges must carry one unit of flow. Moreover, note that each of the edges between s and v_i for $i \in \{1, \ldots, k\}$ may either carry zero or three units of flow since there are three edges between each v_i and v'_i that are linked such that they can only carry the same (integral) amount of flow at the same time. Hence, the amount of flow that arrives at the nodes $v'_1, \ldots, v'_k$ equals three units for exactly q of these

nodes and zero for the remaining $k - q$ nodes. Let $K \subseteq \{1, \dots, k\}$ denote the set of the indices of those nodes v'_i for which three units of flow arrive. Since each of the $3q$ edges between w_j and t, for $j \in \{1, \dots, 3q\}$, carries one unit of flow as described above, it, thus, follows that the adjacent nodes to the nodes v'_i for $i \in K$ are disjoint, which in turn implies that the corresponding sets C_i are mutually disjoint and cover X. Hence, $\mathcal{C}' := \{C_i : i \in K\}$ is a solution to the underlying instance of ExactCoverBy3Sets.

Now suppose that $\mathcal{C}'$ is a solution to a given instance of ExactCoverBy3Sets. For each $C_i \in \mathcal{C}'$, we send three units of flow from s to v_i and one unit on each of the three edges between v_i and v'_i. Since the sets in $\mathcal{C}'$ do not intersect, we can send one unit of flow to each of the nodes w_j for $j \in \{1, \dots, 3q\}$ and finally to the sink. The resulting flow is feasible, integral, and has a flow value of $\operatorname{val}(x) = 3q$, which shows the claim.

In order to show the $\mathcal{NP}$-hardness of approximation, we add an artificial sink t' to the above network and connect t and t' by $3q$ parallel edges with capacity 1 and flow ratio $\frac{1}{3q}$ each. By similar arguments as above, all of these edges carry the same amount of flow in each feasible flow x such that, due to integrality, the amount of flow that can reach the new sink t' is either zero or $3q$. Thus, if there was an α-approximation algorithm for $\alpha \in (1, \infty)$ that computes a solution x' with $\frac{1}{\alpha} \cdot \operatorname{val}(x^*) \leqslant \operatorname{val}(x') \leqslant \operatorname{val}(x^*)$ where x^* denotes an optimal solution, we can decide whether the underlying instance of ExactCoverBy3Sets is a Yes-instance by checking if $\operatorname{val}(x') > 0$, which concludes the proof. □

Theorem 6.39:
The problem of finding a maximum *integral* flow in a generalized processing network is weakly $\mathcal{NP}$-complete to solve and $\mathcal{NP}$-hard to approximate within constant factors on series-parallel graphs.

Proof: For the reduction, we use the SubsetSum-problem, which is defined as follows (cf. (Garey and Johnson, 1979, Problem SP13)):

Instance: Finite set $\{a_1, \dots, a_k\}$ of k positive integers and a positive integer A.

Question: Is there a subset $I \subseteq \{1, \dots, k\}$ such that $\sum_{i \in I} a_i = A$?

Given an instance of SubsetSum, we construct a series-parallel generalized processing network as follows:

We insert three nodes s, v_0, and t. Between v_0 and t, we introduce two parallel edges, one of them with capacity 1 and flow ratio $\frac{1}{A}$ and the other one with capacity $A - 1$ and flow ratio $\frac{A-1}{A}$. Moreover, for each $i \in \{1, \dots, k\}$, we insert an additional node v_i, an edge between s and v_i with capacity a_i and flow ratio 1, and two parallel edges

between v_i and v_0, one of them with capacity 1 and flow ratio $\frac{1}{a_i}$ and the other one with capacity $a_i - 1$ and flow ratio $\frac{a_i - 1}{a_i}$. The resulting graph is depicted in Figure 6.3.

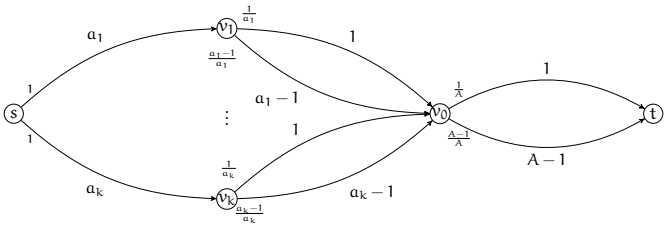


Figure 6.3: The resulting network for a given instance of SubsetSum. The edges are labeled with the corresponding capacities while the edge tails are labeled with the flow ratios.

Note that, due to integrality, the flow that reaches the sink is either zero or A, since the flow on the edge with capacity 1, which amounts to either zero or one, only makes up a fraction $\frac{1}{A}$ of the flow that reaches t. Accordingly, for $i \in \{1, \ldots, k\}$, the amount of flow that reaches v_0 via v_i may either be zero or a_i. By identifying those nodes v_i that receive a_i units of flow from s with those elements that are contained in I, the claims follow by similar arguments as in the proof of Theorem 6.38. □

6.7 Conclusion

In this chapter, we generalized the well-known maximum flow problem in processing networks from flow ratios that determine the *exact* fraction of flow routed through an edge to flow ratios that determine an *upper bound* on this fraction. We were able to generalize the notion of paths as a central unit in traditional network flows to the concept of flow distribution schemes and could show that a flow decomposition similar to the one for traditional network flows is possible on general graphs. Although it was easy to see that the problem is solvable in weakly polynomial time, we could show that the problem is at least as hard to solve as any packing LP. Nevertheless, for acyclic graphs, we presented a (strongly polynomial-time) FPTAS with a running time of $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m\right)$ that also embodies the first approximation scheme for the maximum flow problem in processing networks. Moreover, for the case of series-parallel graphs, we presented two very different approaches on how to solve the problem exactly. The first of these approaches generalizes the idea of the augmenting path algorithm for the traditional maximum flow problem and resulted in

an algorithm with a running time of $\mathcal{O}(m^2)$. The second approach used the structure of series-parallel graphs in a more sophisticated way in order to improve this running time based on a successive shrinking of subgraphs to $\mathcal{O}(m \cdot (n + \log m))$. Finally, we investigated the case of integral flows and showed that the problem becomes strongly $\mathcal{NP}$-hard to solve and approximate on bipartite acyclic graphs and weakly $\mathcal{NP}$-hard to solve and approximate on series-parallel graphs. A complete overview of the results for the continuous and the integral version of MFGPN is given in Table 6.1 and 6.2, respectively.

General Graphs	Acyclic Graphs	Series-Parallel Graphs
Theorem 6.10: Decomposable in $\mathcal{O}(m^4)$ time	**Theorem 6.13**: Decomposable in $\mathcal{O}(m^2)$ time	$\longrightarrow$
$\longleftarrow$	**Theorem 6.15**: At least as hard to solve as any packing LP	
Theorem 6.14: Solvable in $\mathcal{O}(m^{3.5} \log M)$ time	$\longrightarrow$	**Theorem 6.31**: Solvable in $\mathcal{O}(m^2)$ time **Theorem 6.37**: Solvable in $\mathcal{O}(m \cdot (n + \log m))$ time
	Theorem 6.17: FPTAS in $\mathcal{O}\left(\frac{1}{\varepsilon^2} m^2 \log m\right)$ time	$\longrightarrow$

Table 6.1: The summarized results for the continuous maximum flow problem in generalized processing networks in Chapter 6. Implied results are denotes with gray arrows.

General Graphs	Acyclic Graphs	Series-Parallel Graphs
$\longleftarrow$	**Theorem 6.38**: Strongly $\mathcal{NP}$-complete to solve	**Theorem 6.39**: Weakly $\mathcal{NP}$-complete to solve
$\longleftarrow$	**Theorem 6.38**: $\mathcal{NP}$-hard to approximate	**Theorem 6.39**: $\mathcal{NP}$-hard to approximate

Table 6.2: The summarized results for the integral maximum flow problem in generalized processing networks in Chapter 6. Implied results are denoted with gray arrows.

The introduced generalized model provides many topics for future research. On the one hand, it would be worth investigating if a network simplex algorithm as described

in (Wang and Lin, 2009) could be adopted to the extended model. Although the problem lies in $\mathcal{P}$, there are no polynomial-time combinatorial algorithms for the maximum flow problem both in the traditional and the generalized model yet. On the other hand, the introduced model could be further extended to the case of minimum cost flows. It remains open if some of the algorithms introduced in this chapter can be generalized in order to compute a minimum cost flow in a given acyclic or series-parallel network.