7 Convex Generalized Flows

In this chapter, we give insights into the structural properties and the complexity of an extension of the generalized maximum flow problem in which the outflow of an edge is a strictly increasing convex function of its inflow. In contrast to the traditional generalized maximum flow problem, which is solvable in polynomial time as shown in Section 2.4, we show that the problem becomes $\mathcal{NP}$-hard to solve and approximate in this novel setting. Nevertheless, we show that a flow decomposition similar to the one for traditional generalized flows is possible and present (exponential-time) exact algorithms for computing optimal flows on acyclic, series-parallel, and extension-parallel graphs as well as optimal preflows on general graphs. We also identify a polynomially solvable special case and show that the problem is solvable in pseudo-polynomial time when restricting to integral flows on series-parallel graphs.
This chapter is based on joint work with Sven O. Krumke and Clemens Thielen (Holzhauser et al., 2015b).

7.1 Introduction

As it was shown in Section 2.4, the traditional generalized flow problem may be used in order to model real world scenarios such as the loss of water in a broken pipe or the conversion of money between currencies. However, the fixed ratio between the outflow and the inflow of an edge that comes with traditional generalized flows may be insufficient in several applications. In this chapter, we investigate an extension of generalized flows from linear outflow functions $g_e(x_e) = \gamma_e \cdot x_e$ to general strictly increasing continuous convex outflow functions g_e. These more general outflow functions enable us to model processes in which the effectiveness increases with the load. This happens, e.g., in various trading applications where better rates are obtained if larger amounts are traded. By identifying the possible goods with nodes and introducing edges to represent trading options, this effect can be modeled more realistically with the help of convex outflow functions. For two nodes representing the goods A and B, a flow of maximum flow value between these two nodes then yields a strategy for obtaining the maximum amount of good B out of a existing stock of goods of type A.

7.1.1 Previous Work

The traditional generalized maximum flow problem with linear outflow functions has been studied extensively in the past fifty years and is still a topic of active research. In 1977, Truemper (1977) discovered an analogy between generalized maximum flows and traditional minimum cost flows and noted that many of the combinatorial algorithms for the generalized maximum flow problem known at this time were in fact pseudo-polynomial time variations of well-known algorithms for standard minimum cost flow problems. Although the generalized maximum flow problem can be expressed as a linear program and solved in polynomial time, e.g., by interior point methods, it took until 1991 that Goldberg et al. (1991) developed the first (weakly) polynomial-time combinatorial algorithm for the generalized maximum flow problem. For series-parallel graphs, a strongly polynomial-time algorithm was presented by Krumke and Zeck (2013). The first strongly polynomial-time algorithm for general graphs was recently given by Végh (2013).

Ahlfeld et al. (1987) and Tseng and Bertsekas (2000) studied extensions of generalized flows in which the objective function is replaced by a nonlinear function and a minimum cost flow is sought. Nevertheless, the outflow functions are assumed to be linear in both papers. Nonlinear outflow functions in the generalized maximum flow problem have first been studied by Truemper (1978) and later by Shigeno (2006). In both papers, a generalization to (increasing and continuous) *concave* outflow functions is suggested and optimality criteria are presented. By exploiting several analogies to the case of linear outflow functions, Végh (2012) obtained an efficient combinatorial algorithm for this problem. This algorithm, however, makes heavy use of the concavity of the outflow functions, so it cannot be used for the case of convex outflow functions studied here. To the best of our knowledge, (general) convex outflow functions in the generalized maximum flow problem have not been studied so far.

7.1.2 Chapter Outline

After the introduction of the necessary assumptions and definitions in Section 7.2, we derive useful lemmas in Section 7.3 and show that a flow decomposition similar to the one for the traditional generalized flow problem is possible in the case of *convex* generalized flows as well. In Section 7.4, we consider the complexity and approximability of the convex generalized flow problem and show that the problem is both $\mathcal{NP}$-hard to solve and approximate. Afterwards, in Section 7.5, we present algorithms for the problem on graph classes with decreasing complexity. We present an algorithm that computes a maximum convex generalized preflow on general graphs in $\mathcal{O}(3^m \cdot m)$ time. Although such a maximum convex generalized preflow cannot be turned into a feasible flow on general graphs without further assumptions, it will be possible to derive a maximum convex generalized flow within the same time bound when restricting to acyclic graphs. Moreover, we show that we can improve this running time to $\mathcal{O}(2.707^m \cdot (m + n^2))$ in the case of series-parallel graphs and to $\mathcal{O}(2.404^m \cdot (m + n^2))$ time on extension-parallel graphs. Furthermore, we identify a special case of extension-parallel graphs for which the problem becomes solvable in polynomial time. Finally, in Section 7.6, we consider a variant of the problem in which the flows are restricted to be integral and present a pseudo-polynomial time algorithm for the problem on series-parallel graphs. An overview of the results of this chapter is given in Table 7.1 on page 191.

7.2 Preliminaries

We start by defining the convex generalized maximum flow problem in a directed graph $G = (V, E)$ with positive *edge capacities* $u_e > 0$ and *outflow functions* $g_e \colon [0, u_e] \to \mathbb{R}_{\geqslant 0}$ on the edges $e \in E$, and distinguished *source* $s \in V$ and *sink* $t \in V$. We assume that the outflow functions g_e fulfill the following property:

Assumption 7.1: The outflow functions g_e are strictly increasing continuous convex functions fulfilling $g_e(0) = 0$ for all $e \in E$. $\lhd$

Note that, by standard results from analysis, Assumption 7.1 implies that the inverse functions g_e^{-1} are well-defined and continuous as well (cf., e.g., (Rudin, 1964)).

Definition 7.2 (Inflow, outflow, excess):
For any function $x : E \to \mathbb{R}_{\geqslant 0}$, the *inflow of an edge* $e \in E$ is given by $x_e := x(e)$ and the *outflow of edge* e is given by $g_e(x_e)$. Similarly, the *inflow (outflow) of a path* P equals the inflow (outflow) of the first (last) edge on P. For a node $v \in V$, the *inflow of* v is defined as $\sum_{e \in \delta^-(v)} g_e(x_e)$ and the *outflow of* v is given by $\sum_{e \in \delta^+(v)} x_e$. The *excess* of a node $v \in V$ with respect to x is given as $\text{excess}_x(v) := \sum_{e \in \delta^-(v)} g_e(x_e) - \sum_{e \in \delta^+(v)} x_e$. $\lhd$

As in a traditional generalized flow, the outflow of an edge may differ from its inflow. Whereas the ratio between the outflow and the inflow of an edge is constant in traditional generalized flows, this ratio is now a non-decreasing function of the inflow.

Definition 7.3 (Pseudoflow, preflow, flow, flow value, maximum flow):
A function $x : E \to \mathbb{R}_{\geqslant 0}$ is called a (feasible) *convex generalized pseudoflow* (or just *pseudoflow*) if $x_e \leqslant u_e$ for all $e \in E$. If, in addition, $\text{excess}_x(v) \geqslant 0$ for all $v \in V \setminus \{s, t\}$, it is called a (feasible) *preflow*. If $\text{excess}_x(v) = 0$ for all $v \in V \setminus \{s, t\}$, the function is called

a (feasible) *convex generalized flow* (or just *flow*). The *flow value* of a (pre-)flow x is given by $\text{val}(x) := \text{excess}_x(t)$. A (pre-)flow of maximum flow value is called a *maximum (pre-)flow*. $\triangleleft$

Note that a preflow is basically a flow that may send "too much" flow to some nodes such that flow conservation is not fulfilled. In some situations, such a relaxed flow may be much easier to compute than a feasible flow. In fact, while any preflow can be turned into a flow within polynomial time on acyclic graphs, such a transformation is uncomputable on general graphs as it will be shown in Section 7.5.1 and Section 7.5.2.

Using the above definitions, the convex generalized maximum flow problem is defined as follows:

Definition 7.4 (Convex Generalized Maximum Flow Problem (CGMFP)):

INSTANCE:	Directed graph $G = (V, E)$ with source $s \in V$, sink $t \in V$, and non-negative capacities u_e and outflow functions g_e on the edges $e \in E$.
TASK:	Determine a maximum flow.

$\triangleleft$

In addition to Assumption 7.1, we make the following assumptions on the structure of the underlying graph:

Assumption 7.5: For every node $v \in V \setminus \{s, t\}$, it holds that $\delta^+(v) \neq \emptyset$ and $\delta^-(v) \neq \emptyset$. $\triangleleft$

Assumption 7.6: 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. $\triangleleft$

Assumption 7.5 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 7.6 yields no restriction since the corresponding connected components that do not contain the sink t do not contribute to the flow value and can be deleted as well. Note that, for any instance of CGMFP, 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.

An important special case of CGMFP considered throughout this chapter is the case of *quadratic outflow functions* of the form $g_e(x_e) = \alpha_e \cdot x_e^2$ for some positive constants $\alpha_e > 0$.

The problem CGMFP can be formulated as the following nonlinear program:

$$\begin{aligned}
\max \; & \sum_{e\in\delta^-(t)} g_e(x_e) - \sum_{e\in\delta^+(t)} x_e \\
\text{s.t.} \; & \sum_{e\in\delta^-(v)} g_e(x_e) - \sum_{e\in\delta^+(v)} x_e = 0, && \text{for all } v \in V\setminus\{s,t\}, \\
& 0 \leqslant x_e \leqslant u_e, && \text{for all } e \in E.
\end{aligned}$$

Note that this model is not solvable (in polynomial time) by standard solution methods for convex programs since a convex function is maximized (instead of minimized) over a set defined by convex and non-affine equality constraints (so the set of feasible solutions is non-convex in general). In the rest of this chapter, we will concentrate on combinatorial algorithms for the problem.

Moreover, note that, since we allow arbitrary strictly increasing continuous convex outflow functions g_e, it is not canonically clear how the functions are given in the input. In addition, neither the inflow nor the outflow of an edge in a maximum convex generalized flow can be assumed to be rational even if all capacities are integral: Consider, e.g., the instance that is depicted in Figure 7.1, in which the unique maximum generalized flow leads to an irrational inflow of $\sqrt[4]{2}$ and outflow of $\sqrt{2}$ on the first edge. Therefore, similar to the computational model used by Végh (2012) for the case

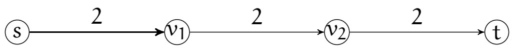


Figure 7.1: An example of a network with integral capacities where the first edge has an irrational inflow and outflow in the unique maximum flow. The labels on the edges represent the capacities. The outflow function of each edge is set to $g_e(x_e) = x_e^2$.

of concave generalized flows, we assume oracle access to the functions g_e and their inverses g_e^{-1} and the running time estimations for our algorithms provide bounds on the number of elementary arithmetic operations and oracle calls. We assume throughout this chapter that we can perform oracle calls returning the value $g_e(x_e)$ for some $x_e \in [0, u_e]$ and the value $g_e^{-1}(y_e)$ for some $y_e \in [0, g_e(u_e)]$ as well as elementary arithmetic operations on real numbers within infinite precision in constant time $\mathcal{O}(1)$. This is similar to standard assumptions in convex and semi-definite programming (cf. (Nesterov and Nemirovskii, 1994; Blum, 1998; Blum et al., 1989; Grötschel et al., 1993)).

7.3 Structural Results

In this section, we give insights into structural properties of convex generalized flows. On the one hand, we show that a flow decomposition theorem similar to the well-known flow decomposition theorem for traditional generalized flows is possible in the case of CGMFP as well. On the other hand, we derive a set of useful lemmas that will be used throughout the rest of this chapter.

The following result allows us to handle paths in the underlying network similarly as single edges:

Lemma 7.7:
Let $P = (e_1, \dots, e_k)$ with $e_i = (v_i, v_{i+1})$, $i \in \{1, \dots, k\}$, be a simple path and x a convex generalized pseudoflow in an acyclic graph $G = (V, E)$. When restricting the flow on each edge $e \notin P$ and the excess at each node $v \notin \{v_1, v_{k+1}\}$ to remain unchanged, the outflow of e_k can be described by a function $\overline{g}$ of the inflow of e_1. This function $\overline{g}$ is continuous, convex, and strictly increasing on the *set of feasible inflows* $[L_x(P), U_x(P)]$ *of* P, i.e., the set of all inflows $y_1 \geqslant 0$ of e_1 such that there exist values $y_2, \dots, y_k \geqslant 0$ for which the function x' with $x'_{e_i} := y_i$ for $i \in \{1, \dots, k\}$ and $x'_e := x_e$ for $e \notin P$ is a feasible pseudoflow with $\text{excess}_{x'}(v_i) = \text{excess}_x(v_i)$ for all $i \in \{2, \dots, k\}$.

Proof: Consider the first two edges $e_1 = (v_1, v_2) \in P$ and $e_2 = (v_2, v_3) \in P$. Let δ be the excess that needs to be generated by the flows on e_1 and e_2 in order to maintain a total excess of $\text{excess}_x(v_2)$ at v_2:

$$\begin{aligned}\delta &:= \text{excess}_x(v_2) - \left(\sum_{e \in \delta^-(v_2) \setminus \{e_1\}} g_e(x_e) - \sum_{e \in \delta^+(v_2) \setminus \{e_2\}} x_e \right) \\ &= g_{e_1}(x_{e_1}) - x_{e_2}. \end{aligned} \tag{7.1}$$

Furthermore, let $[L_{e_i}, U_{e_i}] := [0, u_{e_i}]$ be the interval of feasible inflows of each edge e_i for $i \in \{1, \dots, k\}$. When requiring the excess δ to remain constant, we can describe the outflow of e_2 depending on the inflow of e_1 by a strictly convex and increasing function $\overline{g}$ as well as the set of feasible inflows $[\overline{L}, \overline{U}]$ of the path (e_1, e_2) as follows:

According to equation (7.1), we get that it must hold that $x_{e_2} = g_{e_1}(x_{e_1}) - \delta$. As defined above, the inflow x_{e_2} of edge e_2 must both fulfill $x_{e_2} \geqslant L_{e_2}$ and $x_{e_2} \leqslant U_{e_2}$. Consequently, this is equivalent to the requirement that $x_{e_1} \geqslant g_{e_1}^{-1}(L_{e_2} + \delta)$ and that $x_{e_1} \leqslant g_{e_1}^{-1}(U_{e_2} + \delta)$. Since, furthermore, it must hold that $x_{e_1} \in [L_{e_1}, U_{e_1}]$, we get that $\overline{L} = \max\{L_{e_1}, g_{e_1}^{-1}(L_{e_2} + \delta)\}$ and $\overline{U} = \min\{U_{e_1}, g_{e_1}^{-1}(U_{e_2} + \delta)\}$. For each valid inflow $x \in [\overline{L}, \overline{U}]$, we can then express the outflow of g_{e_2} by $\overline{g}(x) = g_{e_2}(g_{e_1}(x) - \delta)$. Obviously,

$[\overline{L}, \overline{U}] \subseteq [L_{e_1}, U_{e_1}]$ and $[g_{e_1}(\overline{L}) - \delta, g_{e_1}(\overline{U}) - \delta] \subseteq [L_{e_2}, U_{e_2}]$. Thus, both g_{e_1} and g_{e_2} behave strictly convex and increasing for inflows in $[\overline{L}, \overline{U}]$. So, $\overline{g}$ is strictly convex and increasing as well.

By the above procedure, we are able to virtually join the first two edges of the path and to describe the outflow of the new edge by a strictly convex and increasing function. By induction, the claim follows. □

Using the results of Lemma 7.7, we are able to differentiate between full and empty paths – analogously to full and empty single edges:

Definition 7.8 (Full path, empty path):
Let $P = (e_1, \dots, e_k)$ be a simple path and x a convex generalized pseudoflow in $G = (V, E)$. The path P is called *full (with respect to* x*)* if $x_{e_1} = U_x(P)$ and *empty (with respect to* x*)* if $x_{e_1} = L_x(P)$. ◁

Note that a path P is full if and only if there is at least one edge $e \in P$ with $x_e = u_e$. Analogously, we have that $x_e = 0$ for at least one edge $e \in P$ if and only if P is an empty path.

Example 7.9:
Consider the instance of CGMFP that is depicted in Figure 7.2, in which the outflow function and the capacity of each edge $e \in E$ is assumed to be $g_e(x_e) := x_e^2$ and $u_e := 15$, respectively. According to equation (7.1) in the proof of Lemma 7.7, we get that $\delta = 0 - (4 - 5) = 9 - 8 = 1$. It then follows that we can express the outflow of edge e_3 depending on the inflow of edge e_1 by the function $\overline{g}(x) = (x^2 - 1)^2 = x^4 - 2x^2 + 1$, which is convex, continuous, and increasing on the interval $[\overline{L}, \overline{U}]$ with $\overline{L} := \max\left\{0, \sqrt{0+1}\right\} = 1$ and $\overline{U} := \min\left\{15, \sqrt{15+1}\right\} = 4$. Note that for $x_{e_1} := \overline{L} = 1$ the path P is empty and the inflow of edge e_3 is zero. Conversely, for $x_{e_1} := \overline{U} = 4$ the inflow of edge e_3 equals 15 such that the path P is full. ◁

As it turns out, there is a flow decomposition that is similar to the one for traditional generalized flows, which will be shown in the following. To do so, we adapt the corresponding proof for traditional generalized flows with linear outflow functions from Goldberg et al. (1991).

One essential ingredient for the following results is the definition of a *subtraction of flow*. In the case of linear outflow functions (or traditional network flows), it is possible to subtract flow on some path P in the given graph (i.e., to reduce the flow on the edges of P) without influencing the flow on other paths $P' \neq P$. For general convex outflow functions, however, the subtraction becomes more complicated since the removal of flow on some edge e on a path P may reduce the flow value on other

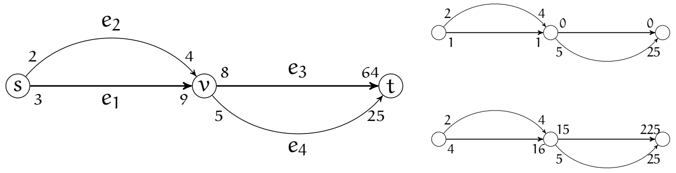


Figure 7.2: An example application of Lemma 7.7 on an instance of CGMFP with outflow functions $g_e := x_e^2$ and capacities $u_e := 15$ for each $e \in E$ (left). The situation for $x_{e_1} = \overline{L} = 1$ ($x_{e_1} = \overline{U} = 4$) is depicted on the upper (lower) right figure. The label on the tail (head) of each edge corresponds to the inflow (outflow) of the corresponding edge.

paths P' with $e \in P'$ as well. Nevertheless, we show that a subtraction similar to the traditional one is possible in our setting as well.

Definition 7.10 (Feasible subtraction):
Let x be a convex generalized pseudoflow in some graph $G = (V, E)$ and let $P = (e_1, \ldots, e_l)$ with $e_i = (v_i, v_{i+1})$, $i \in \{1, \ldots, l\}$, denote a (not necessarily simple) path in G. A function $\overline{x}\colon E \to \mathbb{R}_{\geqslant 0}$ with $\overline{x}_{e_i} > 0$ for each $i \in \{1, \ldots, l\}$ and $\overline{x}_e = 0$ for $e \notin P$ is called a *feasible subtraction on* P if $x_{e_i} - \overline{x}_{e_i} \geqslant 0$ for each $i \in \{1, \ldots, l\}$ and $\text{excess}_{x-\overline{x}}(v_i) = \text{excess}_x(v_i)$ for each $i \in \{2, \ldots, l\}$. $\lhd$

Note that, while the excess at the nodes $v_2, \ldots, v_l$ is required to remain unchanged when reducing the flow on P by a feasible subtraction, the excess at the starting node v_1 and the end node v_{l+1} of the path may change during this procedure in case that $v_1, v_{l+1} \notin \{v_2, \ldots, v_l\}$.

Now consider a pseudoflow x and a simple path $P = (e_1, \ldots, e_l)$ with $e_i = (v_i, v_{i+1})$ and $x_{e_i} > 0$ for each $i \in \{1, \ldots, l\}$. According to Lemma 7.7, there is a flow x' with $x'_e = x_e$ for each $e \notin P$ and with $x'_{e_1} = L_x(P)$ as well as $\text{excess}_{x'}(v_i) = \text{excess}_x(v_i)$ for each $i \in \{2, \ldots, l\}$ such that the path P is empty. Clearly, the function $\overline{x} := x - x'$ is then a feasible subtraction on P fulfilling $\overline{x}_e = 0$ for $e \notin P$ and $\overline{x}_{e_i} > 0$ for each $i \in \{1, \ldots, l\}$. We call this subtraction $\overline{x}$ the *largest subtraction* on P in the following. Moreover, for a cycle C, we can consider C as a simple path with starting node and end node v for some node $v \in C$ and apply the above definition of a largest subtraction. However, the largest subtraction on C as well as the excess that is generated at the starting node v when reducing the flow on P by the largest subtraction may depend on the choice of the starting node.

Similar to traditional flows and generalized flows with linear outflow functions, a convex generalized pseudoflow x does not only decompose into s-t-paths, but also into cycles. As in the case of generalized flows with linear outflow functions, we distinguish three classes of cycles: flow generating cycles, flow absorbing cycles, and flow conserving cycles. The intuition is that a cycle is flow generating (flow absorbing) if removing the flow on the cycle yields a negative (positive) excess at some node on the cycle. If no excess is generated, the cycle is flow conserving.

Definition 7.11 (Flow generating cycle, flow absorbing cycle, flow conserving cycle): Let C be a simple cycle in G, $v \in C$ a node on C, and x a convex generalized pseudoflow in G. The pair (C, v) is called a *flow generating cycle* (*flow absorbing cycle*) with respect to x if $\text{excess}_{x-\overline{x}}(v) < \text{excess}_x(v)$ ($\text{excess}_{x-\overline{x}}(v) > \text{excess}_x(v)$), where $\overline{x}$ denotes the largest subtraction on C when C is considered as a path with starting node (and end node) v. If $\text{excess}_{x-\overline{x}}(v) = \text{excess}_x(v)$, the pair (C, v) is called a *flow conserving cycle*. ◁

Note that, for generalized flows with linear outflow functions, Definition 7.11 can easily be seen to coincide with the standard definitions of flow generating and flow absorbing cycles independent of the choice of the starting node.

In the following, for some convex generalized pseudoflow x, let $\mathcal{D}(x)$ denote the set of nodes with *demand*, i.e., negative excess, and $\mathcal{S}(x)$ denote the set of nodes with *supply*, i.e., positive excess. Analogously to the elementary pseudoflows studied by Gondran and Minoux (1984) and Goldberg et al. (1991), we distinguish five types of *elementary subtractions*, where the type is determined by the graph induced by the set of edges on which the elementary subtraction is positive:

Definition 7.12 (Types of elementary subtractions):

Type I	The largest subtraction on a simple path from a node in $\mathcal{D}(x)$ to a node in $\mathcal{S}(x)$.
Type II	The largest subtraction on a simple path composed of a flow generating cycle (C, v) and a simple path from v to a node $w \in \mathcal{S}(x)$.
Type III	The largest subtraction on a simple path composed of a flow absorbing cycle (C, v) and a simple path from a node $w \in \mathcal{D}(x)$ to v.
Type IV	The largest subtraction on a flow conserving cycle (C, v).
Type V	The largest subtraction on a simple path composed of a flow generating cycle (C_1, v_1) and a flow absorbing cycle (C_2, v_2) that are connected by a simple path from v_1 to v_2.

◁

Note that, by definition of feasible subtractions and flow conserving cycles, the excess may only change at the mentioned nodes in $\mathcal{D}(x)$ and $\mathcal{S}(x)$ when subtracting an elementary subtraction from the current pseudoflow x.

We are now ready to adopt the decomposition theorem for linear outflow functions from Goldberg et al. (1991) to the case of convex outflow functions.

Theorem 7.13 (Decomposition theorem for convex generalized pseudoflows):
A convex generalized pseudoflow x in a graph G can be decomposed into a sequence $(\overline{x}^{(1)}, \dots, \overline{x}^{(k)})$ of $k \leqslant m$ elementary subtractions $\overline{x}^{(j)}$ such that $x_e = \sum_{j=1}^{k} \overline{x}_e^{(j)}$ for each $e \in E$.

Proof: We prove the claim by induction on the number p of edges with positive flow. If $p = 0$, then $x = 0$ and the claim trivially holds. Otherwise, let G' be the graph obtained from G by removing all edges with zero flow.

If G' is acyclic, we can find a simple path P from some node $v \in \mathcal{D}(x)$ to some node $w \in \mathcal{S}(x)$ with positive flow on each edge and the largest subtraction $\overline{x}$ on P is an elementary subtraction of Type I. Furthermore, as described above, $x - \overline{x}$ is a feasible convex generalized pseudoflow with at most $p - 1$ edges with positive flow. Thus, the claim follows by induction.

If G' is not acyclic, let $C = (v_1, \dots, v_{l+1} = v_1)$ be a simple cycle in G'. We consider C as a simple path with starting node and end node v_1. As above, if we consider the largest subtraction $\overline{x}$ on C, then $x - \overline{x}$ is a feasible generalized pseudoflow that is zero on at least one edge in C. While the excess is maintained at each node v_i, $i \in \{2, \dots, l\}$, the excess at v_1 may change by some amount $\delta \in \mathbb{R}$. If $\delta = 0$, the removal of flow on the cycle did not affect the remaining pseudoflow and $\overline{x}$ is an elementary subtraction of Type IV.

If $\delta < 0$, the pair (C, v_1) was a flow generating cycle. Since $x - \overline{x}$ is a valid generalized pseudoflow and has at most $p - 1$ edges with positive flow, we can apply the induction hypothesis and decompose $x - \overline{x}$. Since there was a demand at v_1, there are elementary subtractions of Type I and III in the decomposition of $x - \overline{x}$ that are responsible for the demand. These subtractions together with some appropriate fractions of $\overline{x}$ consequently correspond to elementary subtractions of Type II and Type V in G'. If $\delta > 0$, the pair (C, v_1) was a flow absorbing cycle and, by the same arguments as before, we obtain elementary subtractions of Type III and Type V.

In all three cases, the remaining pseudoflow is feasible again and contains at most $p - 1$ edges with positive flow. Hence, the claim follows by induction. $\square$

Note that, since the largest subtraction on a path or cycle depends on the current pseudoflow x, each elementary subtraction $\overline{x}^{(j)}$ will only be an elementary subtraction with respect to the pseudoflow obtained after all previous elementary subtractions $\overline{x}^{(1)}, \dots, \overline{x}^{(j-1)}$ have already been subtracted from x. In particular, the order of the elementary subtractions within the sequence $(\overline{x}^{(1)}, \dots, \overline{x}^{(k)})$ is important.

Also note that, in case that the pseudoflow x itself is a *flow*, the components in the decomposition for generalized pseudoflows with linear outflow functions obtained in Goldberg et al. (1991) are generalized flows again. Even if the pseudoflow x itself is a flow, however, each elementary subtraction in Theorem 7.13 will only be a feasible convex generalized flow with respect to different (strictly increasing, continuous, and convex) outflow functions given by $\overline{g}_e(x_e) := g_e(c_e + x_e) - g_e(c_e)$ with c_e denoting the remaining flow on edge e after the subtraction.

Finally, note that the computation of a flow decomposition using the recursive procedure presented in the proof of Theorem 7.13 is computationally intractable in general: For example, assume that a largest feasible subtraction $\overline{x}$ on a flow generating cycle (C, v_1) is being removed from the current pseudoflow x in some iteration of the procedure, which changes the excess at node v_1 by $\delta < 0$. A recursive application of the procedure to $x - \overline{x}$ yields elementary subtractions $\overline{x}^{(1)}, \dots, \overline{x}^{(h)}$ of Type I or III that are responsible for the demand at v_1. However, for each such elementary subtraction $\overline{x}^{(j)}$ that generates an excess of $\delta^{(j)} < 0$ at v_1, we then need to determine the appropriate fraction of $\overline{x}$ that generates an excess of exactly $-\delta^{(j)}$ at v_1, i.e., we need to create a specific amount of flow on the cycle C. This is computationally intractable as we will see in the following section.

Example 7.14:
Figure 7.3 shows an exemplary flow and a possible decomposition into a sequence $(\overline{x}^{(1)}, \overline{x}^{(2)})$ of two elementary subtractions according to Theorem 7.13. In this example, all outflow functions are quadratic functions of the form $g_e(x_e) = \alpha_e \cdot x_e^2$ with positive constants $\alpha_e > 0$. Besides the source, which provides one unit of flow, the left-hand cycle is flow generating and creates two units of flow and the right-hand cycle is flow absorbing and consumes two units of flow (independently of the choice of the starting node). The rest of the flow is delivered to the sink.

In the example, there are several ways how to start the decomposition. Assume that we start by extracting an elementary subtraction $\overline{x}^{(1)}$ of Type I that is depicted on the lower left of Figure 7.3. While the inflow at node v_1 is reduced by one unit, the inflow at v_2 sinks by 9 units. The elementary subtraction $\overline{x}^{(1)}$ is a feasible convex generalized flow when changing the outflow function of the edge $e = (v_1, v_2)$ to $\overline{g}_e(x_e) = g_e(4 + x_e) - g_e(4) = (4 + x_e)^2 - 16 = x_e^2 + 8 \cdot x_e$. In this case, we are done since the remaining flow is an elementary subtraction $\overline{x}^{(2)}$ of Type V (shown on the right side of Figure 7.3). $\triangleleft$

We end this section with a useful property of paths in acyclic graphs:

Lemma 7.15:
Let x be a convex generalized flow in an acyclic graph $G = (V, E)$ and let P_1, P_2 be two

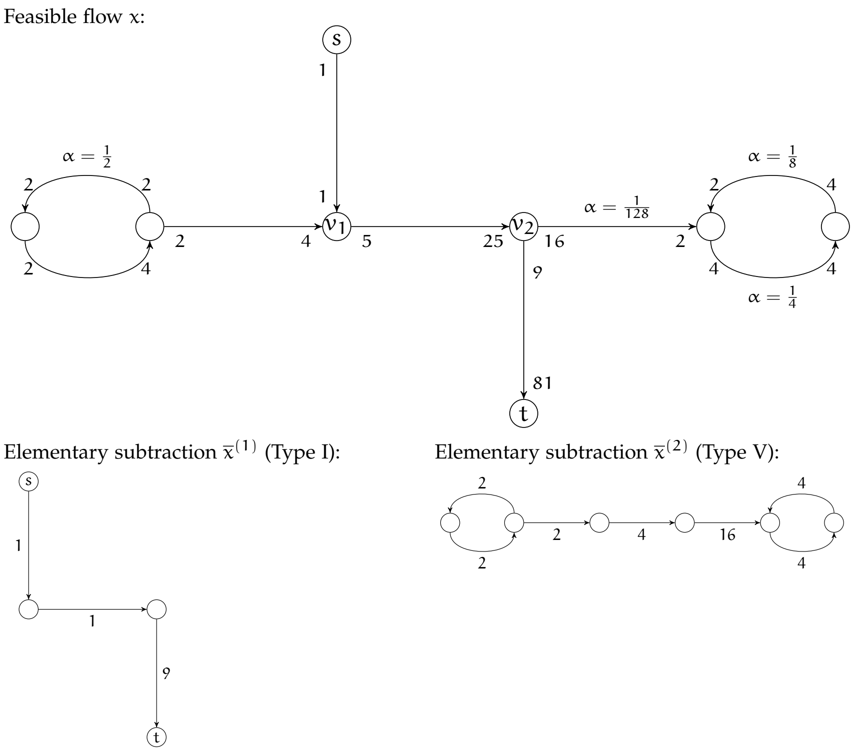


Figure 7.3: A sample flow and a possible decomposition. In the upper figure, the tail (head) of each edge is labeled with the corresponding inflow (outflow). The constant α_e in the outflow function $g_e(x_e) = \alpha_e \cdot x_e^2$ is equal to one if not given explicitly. In the lower figures, each edge is labeled with the amount of flow subtracted. The given flow can be decomposed into an elementary subtraction $\bar{x}^{(1)}$ of Type I (bottom left) and an elementary subtraction $\bar{x}^{(2)}$ of Type V (bottom right).

edge-disjoint v_1-v_2-paths for some nodes $v_1, v_2 \in V$. If both paths are neither full nor empty, then there exists a convex generalized flow x' with $\text{val}(x') = \text{val}(x)$ for which at least one of the paths P_1 and P_2 is full or empty.

Proof: By Lemma 7.7, we can describe the outflow of each P_i, $i \in \{1, 2\}$, by a convex function g_i of its inflow x_i. Writing $\varepsilon := x_1 + x_2$, the total amount of flow arriving at v_2 via P_1 and P_2 is then given as

$$f(x_1) := g_1(x_1) + g_2(\varepsilon - x_1)$$

for

$$x_1 \in [L, U] := [L_x(P_1), U_x(P_1)] \cap [\varepsilon - U_x(P_2), \varepsilon - L_x(P_2)].$$

Since both g_1 and g_2 are convex and increasing, so is f. Thus, the maximum of f on $[L, U]$ is obtained at either L or U and, hence, at a boundary of $[L_x(P_1), U_x(P_1)]$ or $[\varepsilon - U_x(P_2), \varepsilon - L_x(P_2)]$, which means that we can obtain an excess of at least $\text{excess}_x(v_2)$ at v_2 by choosing $x_1 = L$ or $x_1 = U$. By definition of L, U, and ε, this corresponds to turning P_1 or P_2 into a full or empty path.

In case that the excess of v_2 was increased by the above procedure, we can regain the old excess at v_2 by reducing x_1 or x_2 appropriately: If $L_{x'}(P_i) < x'_i < U_{x'}(P_i)$ for the new pseudoflow x' and some $i \in \{1, 2\}$, we can reduce x'_i until either $x'_i = L_{x'}(P_i)$ or the excess at v_2 attains its old value. In the former case, the path P_i becomes empty and we can reduce the flow on the other path until the excess at v_2 attains its old value. If one path P_i is full and the other path is empty, we can simply reduce x'_i until the excess at v_2 attains its old value.

Note that this procedure for regaining the old excess at v_2 may create a positive excess at v_1. This excess can be eliminated by reducing the flow on some of the paths that transport flow from the source s to v_1 in a similar way as above. Hence, only the excess of the source s is changed in the overall procedure. In particular, we obtain a feasible flow with the same flow value as x, which proves the claim. □

Example 7.16:
Figure 7.4 shows an application of Lemma 7.15 on a small acyclic graph. None of the two paths (edges) between v_1 and v_2 is full or empty, but the flow is optimal since the single edge leading to the sink t is filled to its capacity. According to the proof of Lemma 7.15, we can redistribute the flow on the lower path to the upper path, which creates an outflow of value 16 and, thus, an excess of 8 at v_2. We then reduce the inflow of the upper path to $\sqrt{8}$ in order to regain flow conservation at v_2. This, in turn, leads to an excess of $4 - \sqrt{8} > 0$ at v_1, which can be resolved by reducing the inflow of the edge (s, v_1) from 2 to $\sqrt[4]{8} < 2$. ◁

7.4 Complexity and Approximability

In this section, we consider the complexity and approximability of the convex generalized flow problem. As it turns out – in contrast to traditional generalized flows and generalized flows with concave outflow functions – the problem is $\mathcal{NP}$-hard to solve and approximate in general, which will be shown in Section 7.4.1 and 7.4.2, respectively.

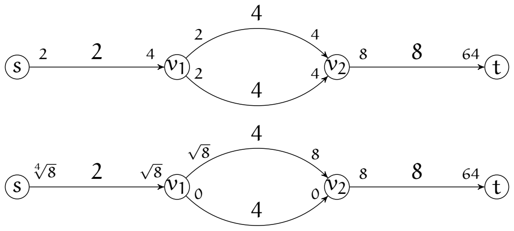


Figure 7.4: A sample application of Lemma 7.15. The number above each edge denotes the capacity of the corresponding edge. The outflow function of every edge e is $g_e(x_e) = x_e^2$. The upper scenario shows an optimal solution in which none of the two paths (edges) from v_1 to v_2 is full or empty. By applying the steps from the proof of Lemma 7.15, we obtain another optimal solution in which the lower path between v_1 and v_2 is empty.

As noted above, we assume the underlying computational model to cohere with the Blum-Shub-Smale model in which we can perform arithmetic operations on irrational numbers within infinite precision in constant time. For this model, an independent theory of $\mathcal{NP}$-completeness has been developed and the connection between this theory and the traditional theory based on the RAM model is unclear. For example, the well-known *traveling salesman problem*, which is $\mathcal{NP}$-complete in the standard RAM model, is not known to be $\mathcal{NP}$-complete in the BSS model. Nevertheless, we want to stress that we use the BSS model only for the sake of simplicity. It can be easily seen that the upcoming algorithms and complexity results are also valid when restricting our considerations to outflow functions that map rational numbers to rational numbers. Hence, the complexity results presented in this section will be based on the traditional theory of $\mathcal{NP}$-completeness in the RAM model. We refer the reader to (Blum, 1998) for further details on the BSS model and the connection to the RAM model.

7.4.1 Complexity

We start by proving that the convex generalized maximum flow problem is strongly $\mathcal{NP}$-hard to solve on general graphs.

Theorem 7.17:
CGMFP is strongly $\mathcal{NP}$-hard to solve, even if all outflow functions are quadratic outflow functions of the form $g_e(x_e) = x_e^2$, the capacities are integral, and the graph is bipartite and acyclic.

Proof: We use a reduction from the ExactCoverBy3Sets problem, which is known to be strongly $\mathcal{NP}$-complete (cf. (Garey and Johnson, 1979, Problem SP2)):

Instance:	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 network for CGMFP as follows:

We introduce a single source s and sink t as well as a node s', which is reachable from s via a single edge with capacity $3q$. For each subset $C_i \in \mathcal{C}$, $i \in \{1, \ldots, k\}$, we insert a node v_i and an edge between s' and v_i with capacity 3^2q. Furthermore, we introduce a node v'_j for each $j \in X$, which is reachable from every v_i with $j \in C_i$ via an edge with capacity 3^3q^2. Finally, we connect each v'_j to the sink t by an edge with capacity 3^6q^4. All outflow functions are set to $g_e(x_e) := x_e^2$. The resulting network for the set $X = \{1, \ldots, 9\}$ and the collection $\mathcal{C} = \{\{1,2,4\},\{2,3,4\},\{3,5,8\},\{4,6,7\},\{6,7,9\}\}$ is shown in Figure 7.5.

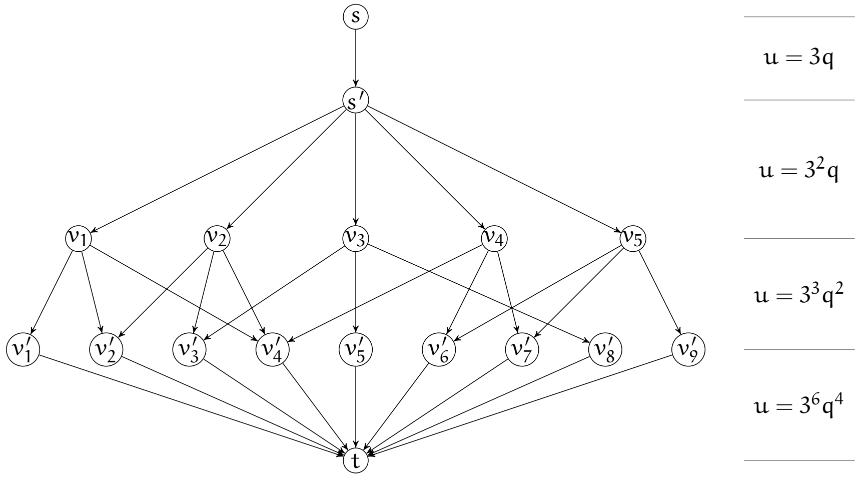


Figure 7.5: The resulting network for a given instance of ExactCoverBy3Sets with $q = 3$, $X = \{1, \ldots, 9\}$ and $\mathcal{C} = \{\{1,2,4\},\{2,3,4\},\{3,5,8\},\{4,6,7\},\{6,7,9\}\}$. On the right hand side, the capacities of the edges in each level of the graph are depicted. The outflow function of every edge is given by $g_e(x_e) := x_e^2$.

We now show that there exists a convex generalized flow of value at least $3^{13}q^9$ if and only if the given instance of ExactCoverBy3Sets is a Yes-instance.

Suppose that there is a convex generalized flow x in the constructed network with flow value $\mathrm{val}(x) \geqslant 3^{13}q^9$. The maximum amount of flow that may arrive at s' is 3^2q^2 since the capacity of the edge between s and s' is $3q$ and the outflow function is given as $g(x_e) = x_e^2$. Furthermore, we claim that the total amount of flow arriving at the nodes v_i, $i \in \{1,\dots,k\}$, is at most 3^4q^3 and this value is achieved if and only if the inflow of 3^2q^2 at s' is distributed equally on exactly q of the edges (s', v_i). To see this, consider a set $I \subseteq \{1,\dots,k\}$ with $|I| = q$. By sending $x_i := 3^2q$ units of flow to v_i, $i \in I$, and $x_i := 0$ units to the remaining v_i, $i \in \{1,\dots,k\} \setminus I$, flow conservation at node s' is fulfilled and, in total, $\sum_{i \in I} g(x_i) = q \cdot (3^2q)^2 = 3^4q^3$ units of flow reach the nodes v_i. Conversely, consider a flow x' that uses more than q of the edges (s', v_i). Then, there are at least two nodes v_{i_1} and v_{i_2} such that $x'_{i_1}, x'_{i_2} \in (0, 3^2q)$. Without loss of generality, we can assume that $x'_{i_1} \geqslant x'_{i_2}$. This flow cannot yield the highest possible amount of flow arriving at the nodes v_i since increasing x'_{i_1} and decreasing x'_{i_2} by some positive amount $\varepsilon \leqslant \min\{3^2q - x'_{i_1}, x'_{i_2}\}$ leads to a strictly higher amount of flow arriving at the nodes v_{i_1}, v_{i_2}:

$$\begin{aligned}(x'_{i_1} + \varepsilon)^2 + (x'_{i_2} - \varepsilon)^2 &= (x'_{i_1})^2 + (x'_{i_2})^2 + \underbrace{2\varepsilon(x'_{i_1} - x'_{i_2})}_{\geqslant 0} + \underbrace{2\varepsilon^2}_{>0} \\ &> (x'_{i_1})^2 + (x'_{i_2})^2.\end{aligned}$$

Hence, the total amount of flow arriving at each of the nodes v_i, $i \in \{1,\dots,k\}$, is at most 3^4q^3. Additionally, this is only the case if exactly q out of k nodes v_i are used which in turn produce outflows of value 3^4q^2 each. Since the sum of the capacities of the three edges leaving each v_i is $3 \cdot 3^3q^2 = 3^4q^2$, every such edge must have an inflow of value 3^3q^2 and produce an outflow of value 3^6q^4 in this situation. On the other hand, note that each of the $3q$ edges leading to the sink must receive the maximum inflow of value 3^6q^4 since this is the only possibility how to obtain the claimed flow value of $3^{13}q^9 = 3q \cdot (3^6q^4)^2$.

In summary, the total flow arriving at the nodes v'_j, $j \in X$, is at most $3q \cdot 3^6q^4 = 3^7q^5$ which, in turn, is the minimum flow needed to achieve the flow value $\mathrm{val}(x)$ at t. Since this flow can only be achieved by selecting q out of the k nodes v_i whose sets of adjacent nodes v'_j are pairwise disjoint and cover all nodes v'_j, we obtain a solution to ExactCoverBy3Sets by identifying the used nodes v_i with the given sets $C_i \in \mathcal{C}$.

Conversely assume that there exists a solution $\mathcal{C}' \subseteq \mathcal{C}$ for the given instance of ExactCoverBy3Sets. By sending $3q$ units of flow to s' and 3^2q units of flow to each of the nodes v_i corresponding to the subsets $C_i \in \mathcal{C}'$, we achieve q inflows of value $(3^2q)^2 = 3^4q^2$ each at the nodes v_i. The flow can further be distributed to the nodes v'_j, $j \in X$, in packages of 3^3q^2 each. Thus, we get an inflow of value $(3^3q^2)^2 = 3^6q^4$ at every node v'_j. Since each element $j \in X$ is contained in exactly one of the sets $C_i \in \mathcal{C}'$, each of these packages can further be sent to t producing outflows of $(3^6q^4)^2 = 3^{12}q^8$ each. Consequently, since there are $3q$ edges leading to the sink, we achieve a flow value of $3q \cdot 3^{12}q^8 = 3^{13}q^9$. □

Theorem 7.17 shows that, unless $\mathcal{P} = \mathcal{NP}$, one cannot expect to find an algorithm that solves the problem exactly and runs in polynomial time. The reason why the theorem only claims $\mathcal{NP}$-hardness instead of $\mathcal{NP}$-completeness (in the standard Turing machine model) is that the problem CGMFP need not always have rational solutions (of polynomial size). In the Blum-Shub-Smale model (Blum et al., 1989), however, CGMFP is readily seen to be in $\mathcal{NP}$ since the feasibility and the flow value of a given convex generalized flow can then be checked easily in (oracle) polynomial time. Nevertheless, we want to stress that, in the presented reduction, a Yes-instance of CGMFP contains only integral numbers and can, thus, be verified in polynomial time using the standard Turing machine model.

Theorem 7.18:
CGMFP on extension-parallel graphs is weakly $\mathcal{NP}$-hard to solve, even if all outflow functions are quadratic functions of the form $g_e(x_e) = \alpha_e \cdot x_e^2$ with integral constants $\alpha_e > 0$ and all capacities are integral.

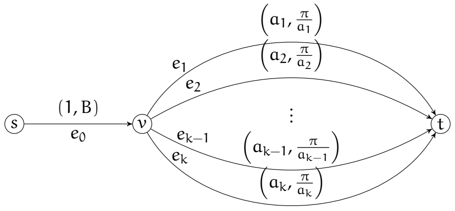
Proof: We use a reduction from the weakly $\mathcal{NP}$-complete SubsetSum problem, which is defined as follows (cf. (Garey and Johnson, 1979, Problem SP13)):

Instance:	Finite set $A = \{a_1, \ldots, a_k\}$ of k positive integers and a positive integer B.
Question:	Is there a subset $I \subseteq \{1, \ldots, k\}$ such that $\sum_{i \in I} a_i = B$?

Given an instance of SubsetSum, we construct a network for CGMFP by introducing three nodes s, v, and t. Between s and v, we insert an edge e_0 with capacity 1 and outflow function $g_{e_0}(x_{e_0}) := B \cdot x_{e_0}^2$. Additionally, we introduce an edge e_i between v and t with capacity a_i and outflow function $g_{e_i}(x_{e_i}) := \frac{\pi}{a_i} \cdot x_{e_i}^2$ for each $i \in \{1, \ldots, k\}$, where $\pi := \prod_{j=1}^{k} a_j$. Note that the factors $\alpha_{e_i} := \frac{\pi}{a_i} = \prod_{j \neq i} a_j$ are integral and the resulting graph is extension-parallel. The constructed network is shown in Figure 7.6.

We now show that there exists a convex generalized flow of value at least $B \cdot \pi$ if and only if the given instance of SubsetSum is a Yes-instance.

Suppose that there is a convex generalized flow x of value $\text{val}(x) \geqslant B \cdot \pi$ in the constructed network. Note that, for each edge e_i between v and t with capacity a_i and inflow x_{e_i}, the outflow is given by $\frac{\pi}{a_i} \cdot x_{e_i}^2$, which evaluates to $\pi \cdot x_{e_i}$ if $x_{e_i} = a_i$ and to some smaller multiple of x_{e_i} if $x_{e_i} < a_i$. Hence, since the maximum possible outflow



$(1, B)$
e_0
s
v
t
e_1
e_2
e_{k-1}
e_k
$\left(a_1, \frac{\pi}{a_1}\right)$
$\left(a_2, \frac{\pi}{a_2}\right)$
$\left(a_{k-1}, \frac{\pi}{a_{k-1}}\right)$
$\left(a_k, \frac{\pi}{a_k}\right)$

Figure 7.6: The constructed network for the given instance of SUBSETSUM. The label on each edge e denotes the capacity u_e and the factor α_e, respectively.

of e_0 is B, the flow value of $\mathrm{val}(x) \geqslant B \cdot \pi$ implies that B units of flow must arrive at v and each edge among $e_1, \ldots, e_k$ that has positive inflow satisfies $x_{e_i} = a_i$. Hence, the B units of flow arriving at v are distributed to some edges e_i, $i \in I$, for some subset $I \subseteq \{1, \ldots, k\}$ that satisfies $\sum_{i \in I} a_i = B$, i.e, the given instance of SUBSETSUM is a YES-instance.

Conversely assume that there exists a solution $I \subseteq \{1, \ldots, k\}$ of the given instance of SUBSETSUM, i.e., $\sum_{i \in I} a_i = B$. By sending $x_{e_0} := 1$ units of flow along e_0 (which amounts to an inflow of value B at v), $x_{e_i} := a_i$ units of flow over the edges e_i for $i \in I$, and $x_{e_j} := 0$ units along the edges e_j for $j \in \{1, \ldots, k\} \setminus I$, we obtain flow conservation at v and get a feasible convex generalized flow x of value

$$\begin{aligned} \mathrm{val}(x) &= \sum_{i \in \{1,\ldots,k\}} g_{e_i}(x_{e_i}) = \sum_{i \in I} g_{e_i}(a_i) = \sum_{i \in I} \frac{\pi}{a_i} \cdot a_i^2 = \sum_{i \in I} \pi \cdot a_i \\ &= B \cdot \pi. \end{aligned}$$

□

7.4.2 Approximability

Since the convex generalized flow problem is strongly $\mathcal{NP}$-hard to solve on general graphs, as it was shown in the preceding section, one might still hope for efficient approximation algorithms for the problem. However, as it turns out, the problem is $\mathcal{NP}$-hard to approximate as well, even on simple graph classes:

Theorem 7.19:
CGMFP is $\mathcal{NP}$-hard to approximate within constant factors, even on extension-parallel graphs.

Proof: For $\varepsilon \in (0, 1)$, suppose that there was an $(1-\varepsilon)$-approximation algorithm for CGMFP that computes a feasible flow x with $(1-\varepsilon) \cdot \mathrm{val}(x^*) \leqslant \mathrm{val}(x) \leqslant \mathrm{val}(x^*)$ in polynomial time, where x^* is a maximum convex generalized flow. We show that this $(1-\varepsilon)$-approximation algorithm allows us to decide if any instance of SubsetSum is a Yes-instance, which is a contradiction unless $\mathcal{P} = \mathcal{NP}$.

Let $(\{a_1, \dots, a_k\}, B)$ denote an instance of SubsetSum. Without loss of generality, we may assume that $a_i \geqslant 2$ for each $i \in \{1, \dots, k\}$ since we can otherwise multiply each of the values a_i and B by two. Similar to the proof of Theorem 7.18, we construct a network for CGMFP by introducing four nodes s, v, w, and t. Between s and v, we insert an edge e_0 with capacity 1 and outflow function $g_{e_0}(x_{e_0}) := B \cdot x_{e_0}$. For each $i \in \{1, \dots, k\}$, we introduce an edge e_i between v and w with capacity a_i and outflow function

$$g_{e_i}(x_{e_i}) := \begin{cases} \frac{x_{e_i}}{a_i - 1} \cdot \frac{1}{2}, & \text{if } x_{e_i} \leqslant a_i - 1, \\ (x_{e_i} - (a_i - 1)) \cdot (a_i - \frac{1}{2}) + \frac{1}{2}, & \text{else.} \end{cases}$$

Note that the functions g_{e_i} are continuous, increasing, and convex for each $i \in \{1, \dots, k\}$: Let $g^{(1)}_{e_i}(x_{e_i}) := \frac{x_{e_i}}{a_i - 1} \cdot \frac{1}{2}$ and $g^{(2)}_{e_i}(x_{e_i}) := (x_{e_i} - (a_i - 1)) \cdot (a_i - \frac{1}{2}) + \frac{1}{2}$ denote the two linear segments of g_{e_i}. It holds that $g^{(1)}_{e_i}(a_i - 1) = \frac{1}{2} = g^{(2)}_{e_i}(a_i - 1)$, which shows continuity of g_{e_i}. Moreover, it holds that

$$0 < \frac{d}{dx_{e_i}} g^{(1)}_{e_i}(x_{e_i}) = \frac{1}{2(a_i - 1)} < 1 < a_i - \frac{1}{2} = \frac{d}{dx_{e_i}} g^{(2)}_{e_i}(x_{e_i}),$$

which shows both convexity and monotonicity of g_{e_i}.

Moreover, we insert an edge $\bar{e}$ between w and t with capacity B and outflow function

$$g_{\bar{e}}(x_{\bar{e}}) := \begin{cases} \frac{x_{\bar{e}}}{B - \frac{1}{2}} \cdot (1-\varepsilon)\frac{B}{4}, & \text{if } x_{\bar{e}} \leqslant B - \frac{1}{2}, \\ (x_{\bar{e}} - (B - \frac{1}{2})) \cdot (1+\varepsilon)\frac{B}{2} + (1-\varepsilon)\frac{B}{4}, & \text{else.} \end{cases}$$

Similar to the functions g_{e_i}, the function $g_{\bar{e}}$ is continuous, increasing, and convex as well: For $g^{(1)}_{\bar{e}}(x_{\bar{e}}) := \frac{x_{\bar{e}}}{B - \frac{1}{2}} \cdot (1-\varepsilon)\frac{B}{4}$ and $g^{(2)}_{\bar{e}}(x_{\bar{e}}) := (x_{\bar{e}} - (B - \frac{1}{2})) \cdot (1+\varepsilon)\frac{B}{2} + (1-\varepsilon)\frac{B}{4}$, it holds that $g^{(1)}_{\bar{e}}(B - \frac{1}{2}) = (1-\varepsilon)\frac{B}{4} = g^{(2)}_{\bar{e}}(B - \frac{1}{2})$, which shows continuity of $g_{\bar{e}}$. Furthermore, since

$$0 < \frac{d}{dx_{\bar{e}}} g^{(1)}_{\bar{e}}(x_{\bar{e}}) = \frac{(1-\varepsilon) \cdot \frac{B}{4}}{B - \frac{1}{2}} = \frac{(1-\varepsilon)}{2B - 1} \cdot \frac{B}{2} < \frac{B}{2} < (1+\varepsilon)\frac{B}{2} = \frac{d}{dx_{\bar{e}}} g^{(2)}_{\bar{e}}(x_{\bar{e}}),$$

the function $g_{\bar{e}}$ is increasing and convex as well.

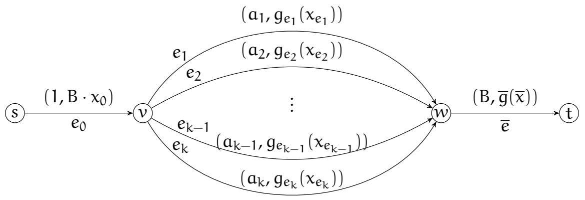

$(a_1, g_{e_1}(x_{e_1}))$
$(a_2, g_{e_2}(x_{e_2}))$
e_1
e_2
$(1, B \cdot x_0)$
e_0
s
v
$\vdots$
w
$(B, \overline{g}(\overline{x}))$
$\overline{e}$
t
e_{k-1}
e_k
$(a_{k-1}, g_{e_{k-1}}(x_{e_{k-1}}))$
$(a_k, g_{e_k}(x_{e_k}))$

Figure 7.7: The constructed network for the given instance of SubsetSum. The label on each edge e denotes the capacity u_e and the outflow function g_e, respectively.

The resulting network is depicted in Figure 7.7. We now show that the flow value of a maximum convex generalized flow equals B if there is a solution to the given instance of SubsetSum and less than $(1 - \varepsilon) \cdot B$ else.

Let $I \subseteq \{1, \dots, k\}$ be a solution to the given instance of SubsetSum, i.e., $\sum_{i \in I} a_i = B$. By sending $x_{e_0} := 1$ unit of flow through e_0, $x_{e_i} := a_i$ units of flow through edge e_i for $i \in I$, $x_{e_j} := 0$ units of flow through edge e_j, $j \in \{1, \dots, k\} \setminus I$, and $x_{\overline{e}} := B$ units of flow through $\overline{e}$, we get a feasible convex generalized flow x with flow value $\text{val}(x) = B$, which is clearly maximum.

Now suppose that there is no solution to the given instance of SubsetSum and let $I := \{i \in \{1, \dots, k\} : x^*_{e_i} > 0\}$, where x^* is a maximum convex generalized flow in the constructed instance of CGMFP. Clearly, if $\sum_{i \in I} a_i \leqslant B - 1$, the flow arriving at node w amounts to at most $\sum_{i \in I} g_{e_i}(a_i) = \sum_{i \in I} a_i \leqslant B - 1$, which causes a flow value of less than $(1 - \varepsilon) \cdot \frac{B}{4} < (1 - \varepsilon) \cdot B$ after passing edge $\overline{e}$ due to the definition of $g_{\overline{e}}$. If $\sum_{i \in I} a_i \geqslant B + 1$, there is exactly one $i \in I$ with $0 < x^*_{e_i} < a_i$ without loss of generality according to Lemma 7.15 since there are only at most B units of flow arriving at node v. Moreover, since all of the values a_i are integral, it holds that $x^*_{e_i} \leqslant a_i - 1$ such that $g_i(x^*_{e_i}) \leqslant \frac{1}{2}$. Then, however, the amount of flow that arrives at node w is given by $\sum_{j \in I} x^*_j \leqslant \sum_{j \in I \setminus \{i\}} a_j + \frac{1}{2} \leqslant B - \frac{1}{2}$, which implies that the maximum flow value $\text{val}(x^*)$ is less than $(1 - \varepsilon) \cdot \frac{B}{4} < (1 - \varepsilon) \cdot B$.

Hence, the flow value of a maximum convex generalized flow in the constructed network is B if there is a solution to the given instance of SubsetSum and less than $(1 - \varepsilon) \cdot B$ else. Thus, any $(1 - \varepsilon)$-approximation algorithm returns a solution x with flow value $\text{val}(x) \geqslant (1 - \varepsilon) \cdot B$ if and only if the underlying instance of SubsetSum is a Yes-instance, which proves the theorem. □

7.5 Exact Algorithms

As it was shown in the preceding section, the problem CGMFP is both $\mathcal{NP}$-hard to solve and to approximate. Thus, unless $\mathcal{P} = \mathcal{NP}$, we will not be able to solve the problem in polynomial time. However, we are able to derive exponential-time exact algorithms that compute maximal generalized (pre-)flows. In this section, we consider different graph classes with decreasing structural complexity. We will be able to derive more efficient algorithms with decreasing complexity of the underlying graph. Moreover, in Section 7.5.5, we introduce a special case of extension-parallel graphs for which a maximum convex generalized flow can be computed in polynomial time.

7.5.1 General Graphs

In this section, we present an exponential-time algorithm that computes a maximum preflow on general graphs in $\mathcal{O}(3^m \cdot m)$ time. We start by proving several auxiliary results.

The proof of the following well-known fact is provided for the sake of completeness:

Lemma 7.20:
A full-dimensional polytope in $\mathbb{R}^n$ has at least $n+1$ facets.

Proof: Let $\{x \in \mathbb{R}^n : Ax \leqslant b\}$ be an non-redundant formulation of a full-dimensional polytope $P \subseteq \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$. Since P is full-dimensional and bounded, it contains at least two distinct extreme points $x^{(1)}$ and $x^{(2)}$. Each of these extreme points can be determined by setting a set of n inequalities to equality. Thus, since $x^{(1)} \neq x^{(2)}$, the formulation has at least $n+1$ inequalities. It is well known that, in such a non-redundant formulation, there is a one-to-one correspondence between the inequalities and the facets of P, see e.g. (Schrijver, 1998). □

Consider a partition (L, T, U) of the edge set E into three sets L, T, and U, where T forms a spanning tree of the graph G. Similar to the network simplex algorithm for the traditional minimum cost flow problem (cf. Section 4.3), we refer to this partition (L, T, U) as a *basis structure* in the following. We refer to any convex generalized preflow x fulfilling $x_e = 0$ for each $e \in L$ and $x_e = u_e$ for each $e \in U$ as a *preflow corresponding to the basis structure*. As in the case of traditional and budget-constrained minimum cost flows, we can restrict our considerations to such preflows:

Proposition 7.21:
Let x be a convex generalized preflow in a graph $G = (V, E)$. Then there exists

a convex generalized preflow x' corresponding to a basis structure (L, T, U) with $\mathrm{val}(x') \geqslant \mathrm{val}(x)$.

Proof: For the given convex generalized preflow x, consider the partition of the edge set given as $L := \{e \in E : x_e = 0\}$, $U := \{e \in E : x_e = u_e\}$, and $T := \{e \in E : x_e \in (0, u_e)\}$. If the subgraph that is induced by the edges in T does not contain any cycle, the claim clearly follows since we can add edges from L or U to T until the edges in T form a spanning tree of G.

Now let $C = (e_1, \dots, e_k)$ be a (possibly undirected) cycle in G such that $e_i \in T$ and $x_{e_i} \in (0, u_{e_i})$ for each $i \in \{1, \dots, k\}$. In the following, we refer to such a cycle as a T-*cycle*. We show that there also exists a preflow x' with $\mathrm{val}(x') \geqslant \mathrm{val}(x)$ in which the flow on C is rerouted in a way such that at least one edge on C belongs to L or U. By a repeated application of these arguments, the claim then follows.

Let $(P_1, \dots, P_\kappa)$ denote the partition of C into maximal directed subpaths. We replace each of the subpaths P_i by a single edge $\overline{e}_i$. The outflow of each such edge $\overline{e}_i$ can then be described by a convex function $\overline{g}_i$ of its inflow according to Lemma 7.7. If $\kappa = 1$, i.e., C is a directed cycle, we can set the flow entering the cycle at the starting node v of P_1 to $U_x(P_1)$ or $L_x(P_1)$ depending on whether (C, v) is a flow generating cycle or a flow absorbing cycle for x. Since the excess then increases at v and remains constant at every other node, the claim follows.

Now let C be an undirected cycle. By construction, in the resulting (undirected) cycle $\overline{C} = (\overline{e}_1, \dots, \overline{e}_\kappa)$, the directions of the edges alternate. The situation before and after this procedure is depicted in Figure 7.8.

Note that the number κ of nodes and edges on $\overline{C}$ is even since the direction of the edges changes at each node. Moreover, we may assume without loss of generality that $\kappa \geqslant 4$ since, for the case $\kappa = 2$, the cycle consists of two parallel paths and we can proceed as in the proof of Lemma 7.15 in order to make one of the paths full or empty while only generating positive excess at the end node. Furthermore, each node on $\overline{C}$ has either two incoming edges or two outgoing edges in $\overline{C}$ and we assume that the edges and nodes are labeled as in Figure 7.8, i.e., nodes with odd index have two outgoing edges and nodes with even index have two incoming edges. Furthermore, edges with odd index j start from $\overline{v}_j$ and head to $\overline{v}_{j+1}$, while edges with even index l head from $\overline{v}_{l+1}$ to $\overline{v}_l$. Let $\overline{g}_i$ be the function describing the outflow of $\overline{e}_i$ for $i \in \{1, \dots, \kappa\}$, which is convex according to the above explanations. Furthermore, let s_j denote the sum of the inflows of the edges $\overline{e}_{j-1}$ and $\overline{e}_j$ in the given preflow x for $j \in \{1, 3, 5, \dots, \kappa - 1\}$, i.e., the flow leaving node $\overline{v}_j$ along the edges of the cycle $\overline{C}$. Similarly, let d_l denote the flow arriving at $\overline{v}_l$ via $\overline{e}_{l-1}$ and $\overline{e}_l$ in x, $l \in \{2, 4, \dots, \kappa\}$. Note that, due to notational convenience, we avoid using modulo-functions, i.e., whenever

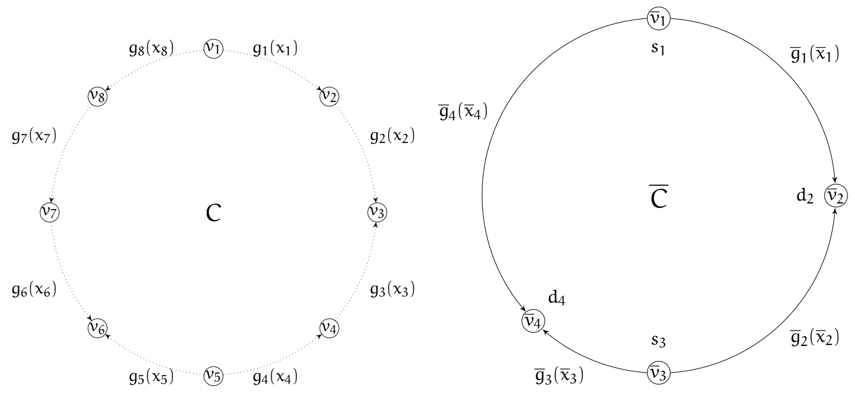


Figure 7.8: Replacing each maximal directed subpath of a cycle by a single edge yields a cycle in which the directions of the edges alternate. The left figure shows the situation before, the right figure after the procedure. The new outflow functions $\overline{g}_i$ are convex on the set of feasible inflows.

an index evaluates to 0 or $\kappa+1$, it should be κ or 1, respectively. Moreover, we will always denote odd indices by j and even indices by l.

Our aim is to distribute the flows s_j leaving the odd nodes $\overline{v}_j$ onto $\overline{e}_{j-1}$ and $\overline{e}_j$ in a way such that the inflow of each even node $\overline{v}_l$ is at least d_l, while the flow on at least one edge $\overline{e}_i$ lies in $\{L_x(P_i), U_x(P_i)\}$ (which means that at least one edge $e \in P_i$ will have $x'_e = 0$ or $x'_e = u_e$ as remarked above after Definition 7.8). To do so, we formulate a system of nonlinear inequalities in the inflows y_j of the odd edges $\overline{e}_j$. The inflow y_l of each even edge $\overline{e}_l$ can then be expressed as $y_l = s_{l+1} - y_{l+1}$. We then have the boundary conditions $y_j \in [L_x(P_j), U_x(P_j)]$ and $s_j - y_j \in [L_x(P_{j-1}), U_x(P_{j-1})]$ or, equivalently,

$$y_j \in [L_j, U_j] := [L_x(P_j), U_x(P_j)] \cap [s_j - U_x(P_{j-1}), s_j - L_x(P_{j-1})].$$

Hence, we want to find a solution to the following system of nonlinear inequalities for which $y_j \in \{L_j, U_j\}$ for at least one $j \in \{1, 3, \dots, \kappa-1\}$:

$$\begin{aligned}
\overline{g}_1(y_1) + \overline{g}_2(s_3 - y_3) &\geqslant d_2, \\
\overline{g}_3(y_3) + \overline{g}_4(s_5 - y_5) &\geqslant d_4, \\
&\vdots \\
\overline{g}_{\kappa-1}(y_{\kappa-1}) + \overline{g}_\kappa(s_1 - y_1) &\geqslant d_\kappa, \\
y_j &\in [L_j, U_j], j \in \{1, 3, \dots, \kappa-1\}.
\end{aligned}$$

According to the definitions of s_j and d_j, choosing y_j to be the inflow of the path P_j in the original preflow x yields a solution $\overline{y}$ of the system that fulfills all inequalities with equality. In the following, let $S := \{(y_1, y_3, \ldots, y_{\kappa-1}) : \overline{g}_j(y_j) + \overline{g}_{j+1}(s_{j+2} - y_{j+2}) \geqslant d_{j+1}$ for $j = 1, 3, \ldots, \kappa - 1\}$ be the set of vectors satisfying the inequalities and $D := [L_1, U_1] \times [L_3, U_3] \times \ldots \times [L_{\kappa-1}, U_{\kappa-1}]$ the set of vectors of allowed inflows. Since $\overline{C}$ was a T-cycle, the solution $\overline{y}$ mentioned above lies in $S \cap D^\circ$ and we are done if we can show that there also exists a solution in $S \cap \partial D$.[1]

For the sake of a contradiction, assume that there is no solution in $S \cap \partial D$ and, for $j \in \{1, 3, \ldots, \kappa - 1\}$, let

$$C_j := \{(y_1, y_3, \ldots, y_{\kappa-1}) : \overline{g}_j(y_j) + \overline{g}_{j+1}(s_{j+2} - y_{j+2}) < d_{j+1}\}$$

denote the set of points violating inequality j. Since $S = \mathbb{R}^{\kappa/2} \setminus \bigcup_{j \in \{1,3,\ldots,\kappa-1\}} C_j$ and we assumed that there is no solution in $S \cap \partial D$, the boundary ∂D must be contained in $\bigcup_{j \in \{1,3,\ldots,\kappa-1\}} C_j$. Since the functions $\overline{g}_i$ are convex, the sets $D \cap C_j$ are convex as well and we can find a hyperplane $H_j := \{y = (y_1, y_3, \ldots, y_{\kappa-1}) : \omega_j \cdot y = b_j\}$ with $\omega_j \cdot \overline{y} < b_j$ that separates $D \cap C_j$ from $\overline{y}$ for each $j \in \{1, 3, \ldots, \kappa - 1\}$. Thus, the set $P := \{y = (y_1, y_3, \ldots, y_{\kappa-1}) : \omega_j \cdot y \leqslant b_j$ for $j = 1, 3, \ldots, \kappa - 1\}$ is a polyhedron with $\overline{y} \in P$. In fact, since $\overline{y} \in P \cap (S \cap D)$, the polyhedron P must be a polytope enclosing $\overline{y}$ (otherwise, it would be possible to pass $S \cap \partial D$ following an extreme ray of P). Moreover, since $\overline{y}$ lies in the topological interior of P, we have $P^\circ \neq \emptyset$, which shows that P is a full-dimensional polytope in $\mathbb{R}^{\kappa/2}$. According to Lemma 7.20, a polytope of dimension $\frac{\kappa}{2}$ must have at least $\frac{\kappa}{2} + 1$ facets, whereas P was defined by only $\frac{\kappa}{2}$ inequalities, which yields a contradiction. Thus, there exists a solution in $S \cap \partial D$.

Hence, in terms of the original problem, we have now shown that there exists a feasible generalized preflow on G with flow value at least $\text{val}(x)$ for which the flow on at least one edge $e \in C$ is contained in $\{0, u_e\}$ (the flow value can have increased in case that the sink t is one of the nodes $\overline{v}_l$). Note that we only changed the flow on edges on C. Hence, by a repeated application of the above arguments, we finally obtain that there are no T-cycles left, which proves the claim. □

Proposition 7.21 builds the foundation of the following main result of this subsection:

Theorem 7.22:
A maximum convex generalized preflow can be computed in $\mathcal{O}(3^m \cdot m)$ time.

Proof: Since we can restrict our considerations to preflows corresponding to basis structures as shown in Proposition 7.21, the theorem follows if we can show that,

1 Here, D° denotes the topological interior of D and $\partial D := \bar{D} \setminus D^\circ$ the boundary.

given such a basis structure, we can compute feasible flow values on the edges in T that yield a maximum convex generalized preflow (or decide that the partition does not allow a feasible preflow) in $\mathcal{O}(m)$ time. We show that we can proceed similarly as in the traditional network simplex algorithm in order to reconstruct the flow that corresponds to a partition of the edge set. In doing so, we may discard partitions that may lead to feasible preflows. Nevertheless, we never discard partitions that lead to a maximum convex generalized preflow.

Clearly, we can discard partitions in which some node $v \in V$ that is incident only to L-edges and U-edges has a negative excess. Moreover, by Proposition 7.21, we can discard partitions that contain T-cycles and restrict our considerations to partitions in which the edges in T form a spanning tree of G.[2] Since T is a spanning tree, it contains the sink t. We designate t as the root of the tree T and seek to move the excess from each leaf of the tree towards the root in the following.

Each leaf v of the tree T is incident to exactly one edge $e \in T$. Let δ_v denote the excess at v generated by the L- and U-edges incident to v. We try to specify the flow on e in a way such that the excess at v becomes zero and will get transported towards the root node: If e is heading from v to some node w and $\delta_v < 0$, we discard the current partition since it does not allow a feasible preflow (since x_e is required to be non-negative, a negative excess will remain at v). If $\delta_v \geqslant 0$, we set $x_e := \min\{u_e, \delta_v\}$ in order to move the excess at v towards the root. Similarly, if e is heading from some node w to v, we set $x_e := \min\{u_e, g_e^{-1}(-\delta_v)\}$ if $\delta_v \leqslant 0$ in order to satisfy the demand at v and discard the partition if $\delta_v > 0$. In any case, since we have specified the flow on e, we can delete the edge from the tree and continue with the next leaf and so on. Note that this procedure maintains a non-negative excess at each leaf while creating the maximum possible excess at the corresponding adjacent inner node of the tree. Eventually, the flow on each edge $e \in T$ is determined (if possible) while creating the maximum possible excess at the root of the tree.

For each of the 3^m possible partitions (L, T, U) of E, we are able to find a node that is incident to exactly one edge in T efficiently by maintaining values $b(v)$ that correspond to the number of incident T-edges of a node $v \in V$ for which the flow value has not yet been fixed and a queue Q of nodes v with $b(v) = 1$. Whenever the flow value x_e of an edge $e \in T$ with end nodes v and w is fixed, we are able to decrease $b(v)$ and $b(w)$ and update Q in constant time $\mathcal{O}(1)$. Consequently, the reconstruction needs $\mathcal{O}(m)$ time, which proves the claimed total running time of $\mathcal{O}(3^m \cdot m)$. □

2 Note that one can check in $\mathcal{O}(m)$ time whether a given graph contains an undirected cycle by using a depth-first search in the corresponding undirected graph.

Note that the maximum convex generalized preflow that is obtained by the above algorithm cannot be transformed into a maximum convex generalized flow without further assumptions: Suppose that there is a flow generating cycle (C, v) for some preflow x that creates s_v units of flow at v resulting in an excess of $\delta_v \in (0, s_v)$ at v. According to Lemma 7.7, we can describe the outflow of C at v by a convex function $\overline{g}$ of the inflow x_C of C at v that fulfills $\overline{g}(x_C) - x_C = s_v$. In order to get rid of the positive excess at v, we need to solve the equation $\overline{g}(x_C) - x_C = s_v - \delta_v$ in x_C, which is uncomputable in general even for a strictly increasing continuous convex function $\overline{g}$ since we do not have oracle access for the function $\overline{g}(x_C) - x_C$.

However, if we suppose that there is an oracle $\mathcal{A}$ that solves the above kind of equations in $\mathcal{O}(T_{\mathcal{A}})$ time, it is possible to find a maximum convex generalized flow on general graphs in $\mathcal{O}(3^m \cdot nmT_{\mathcal{A}})$ time as follows: For some partition (L, T, U) of the edge set E that implies a feasible preflow, consider a node $v \in V \setminus \{s, t\}$ with positive excess. Starting at $z := v$, we recursively follow some edge $e = (w, z) \in E$ with positive flow x_e and set $z := w$. Eventually, we either reach the source s or some node v' considered before, i.e., we obtain a cycle. In the first case, we find a feasible subtraction of Type I that either removes the excess at v or removes all flow on some edge on the underlying s-v-path. Note that this case may occur up to $\mathcal{O}(m + n) = \mathcal{O}(m)$ times and causes an overhead of $\mathcal{O}(n)$. In the second case, there is a minimum inflow x_P into the path P between v' and v such that the excess at node v and the flow on each edge on the path between v' and v remains non-negative. By incorporating the oracle $\mathcal{A}$, we can then either find a flow on the cycle such that the inflow into the path equals x_P while the excess at v' remains constant or we can reduce the flow on the cycle completely. Again, this case can occur up to $\mathcal{O}(m + n)$ times and causes an overhead of $\mathcal{O}(n \cdot T_{\mathcal{A}})$. This yields a total time requirement of $\mathcal{O}(nm \cdot T_{\mathcal{A}})$ per partition for converting the maximum preflow corresponding to the partition into a flow with the same flow value.

7.5.2 Acyclic Graphs

As shown above, according to Theorem 7.22, we can obtain a maximum convex generalized *pre*flow on general graphs in $\mathcal{O}(3^m \cdot m)$ time. When restricting to acyclic graphs, we are able to turn preflows into flows efficiently, which can be incorporated into the proof of Theorem 7.22 in order to obtain a maximum convex generalized flow within the same time bound. This will be shown in the following corollary:

Corollary 7.23:
A maximum convex generalized flow in an acyclic graph can be computed in $\mathcal{O}(3^m \cdot m)$ time.

Proof: The algorithm used in the proof of Theorem 7.22 considers each possible partition of E into L, T, and U and computes a feasible preflow for this partition if possible. In acyclic graphs, the positive excess that might occur at any node $v \in V \setminus \{s,t\}$ stems from flows on s-v-paths according to Theorem 6.10. Thus, each of the computed preflows can afterwards be turned into a feasible flow by computing feasible subtractions of Type I on those paths that create the excess at the nodes $v \in V \setminus \{s,t\}$ with positive excess. This can be done in linear time $\mathcal{O}(m)$ as follows: Let $(v_1, \ldots, v_n)$ with $v_1 = s$ and $v_n = t$ denote a topological sorting of the nodes in V and let δ_v denote the excess of each node $v \in V$, which is non-negative for each $v \in V \setminus \{s,t\}$. Let i denote the maximum index with $i < n$ such that $\delta_{v_i} > 0$. Note that the positive excess at v_i stems from the ingoing edges $e \in \delta^-(v_i)$ only. Hence, as long as $\delta_{v_i} > 0$, we can reduce the inflow of some $e = (v_j, v_i) \in \delta^-(v_i)$ with $x_e > 0$, decrease δ_{v_i}, and increase δ_{v_j} appropriately until either $x_e = 0$ or $\delta_{v_i} = 0$. In the first case, we proceed with another edge in $\delta^-(v_i)$ with positive flow. In the second case, we again consider the maximum index $i < n$ with $\delta_{v_i} > 0$ until no such index exists. Eventually, we get rid of the positive excess at each $v \in V \setminus \{s,t\}$ by considering each edge at most once, which shows the claim. $\square$

7.5.3 Series-Parallel Graphs

We now restrict our considerations to the case of series-parallel graphs. Although Corollary 7.23 already provides an algorithm that solves CGMFP on series-parallel graphs exactly, it is possible to obtain a better running time by exploiting the inherent structure of the underlying graph. As it was shown in (Holzhauser et al., 2015b), the problem becomes solvable in $\mathcal{O}(2.83^m \cdot (m + n^2))$ time when using a more sophisticated approach of creating the basis structures. We present a revised approach that comes with an improved running-time of $\mathcal{O}(2.707^m \cdot (m + n^2))$ time and a simplified proof:

Theorem 7.24:
A maximum convex generalized flow in a series-parallel graph can be computed in $\mathcal{O}(2.707^m \cdot (m + n^2))$ time.

Proof: The idea of the algorithm is similar to the one presented in the proof of Theorem 7.22: For a given basis structure (L, T, U), we try to reconstruct the flow on the edges such that flow conservation is fulfilled at each node $v \in V$ if possible. Although the reconstruction procedure used in this proof has an increased running time of $\mathcal{O}(m + n^2)$ compared to the procedure used in the proof of Theorem 7.22, it yields a better overall running time since it can be interleaved with a traversal of a decomposition tree of the series-parallel graph in order to reduce the number of partitions that need to be considered within the algorithm.

Contraction Procedure:

We start by describing the new procedure for reconstructing the flow from a given partition of E into L, T, and U (or deciding that the partition does not correspond to a feasible flow). As a fourth kind of node, we introduce *unspecified* edges $e \in X$ for which the type will be determined in a later step of the algorithm. For each edge $e \in E$, we store two attributes: the interval in_e of potential inflows and the interval out_e of potential outflows. Obviously, for each $e \in L$, we get $\mathrm{in}_e = \mathrm{out}_e = [0, 0]$ and, for each edge $e \in U$, we get $\mathrm{in}_e = [u_e, u_e]$ and $\mathrm{out}_e = [g_e(u_e), g_e(u_e)]$. Analogously, for $e \in T$ or $e \in X$, we have $\mathrm{in}_e = [0, u_e]$ and $\mathrm{out}_e = [0, g_e(u_e)]$ initially.

The algorithm is based on a fixed decomposition tree of the underlying series-parallel graph G. It repeatedly identifies *series trees* (maximal subtrees in which all inner nodes correspond to series compositions, cf. Figures 7.9a and 7.9c) or *parallel trees* (maximal subtrees in which all inner nodes correspond to parallel compositions, cf. Figure 7.9b) of the decomposition tree and contracts them into single edges. As an invariant that will be established in the creation process of the partitions below, we assume that every edge that corresponds to a leaf in a series tree is contained in X. Similarly, we assume that exactly one edge corresponding to a leaf in a parallel tree is in X while every other edge is contained in $L \cup U$.

Contraction of a series trees: Consider a series tree T with $k_T \geqslant 2$ leaves corresponding to edges $e_1, \dots, e_{k_T}$ in the underlying series-parallel graph. As noted above, we can assume that the type of all of these edges is unspecified, so $\mathrm{in}_{e_i} = [a_i, b_i]$ and $\mathrm{out}_{e_i} = [c_i, d_i]$ for some values $a_i, b_i, c_i, d_i \geqslant 0$. In the original graph, the sequence $(e_1, \dots, e_{k_T})$ corresponds to a path P of length k_T. Note that the flow on P is determined by the flow on one single edge on P. Similar as in the proof of Lemma 7.2, we can find a maximal interval of the form $[a, b]$ such that, for each $y \in [a, b]$, it holds that $x_{e_i} \in [a_i, b_i]$ for each $i \in \{2, \dots, k_T\}$ in a flow x on P with $x_{e_1} = y$. We can, thus, replace the path P by a single unspecified edge $e \in X$ with $\mathrm{in}_e = [a, b]$ and $\mathrm{out}_e = [g_{e_{k_T}}(\dots(g_{e_1}(a))\dots), g_{e_{k_T}}(\dots(g_{e_1}(b))\dots)]$. In the decomposition tree, we similarly replace the series tree T by a new leaf corresponding to the edge e. Note that, if $\mathrm{in}_e = \emptyset$, we can skip the given partition since it does not allow a feasible flow.

Contraction of parallel trees: Now consider a parallel tree T with $k_T \geqslant 2$ leaves corresponding to edges $e_1, \dots, e_{k_T}$ in the underlying series-parallel graph. According to the above invariant, we can assume that there is exactly one index $j \in \{1, \dots, k_T\}$ such that $e_j \in X$ and that there are two disjoint index sets I_L and I_U denoting empty and full edges, respectively, such that $I_L \cup I_U = \{1, \dots, k_T\} \setminus \{j\}$. Note that the intervals in_{e_j} and out_{e_j} are of the form $\mathrm{in}_{e_j} = [a_j, b_j]$ and $\mathrm{out}_{e_j} = [c_j, d_j]$ while the intervals of every other edge e_i are of the form $\mathrm{in}_{e_i} = [a_i, a_i]$ and $\mathrm{out}_{e_i} = [c_i, c_i]$. Hence, the flow in the series-parallel subgraph G' that corresponds to the parallel tree T only depends on the flow on e_j, so we can replace T by a single leaf corresponding to an edge e with the intervals

$$\mathrm{in}_e := \left[a_j + \sum_{i \in I_L \cup I_U} a_i, b_j + \sum_{i \in I_L \cup U_L} a_i \right]$$

and

$$\mathrm{out}_e := \left[c_j + \sum_{i \in I_L \cup I_U} c_i, d_j + \sum_{i \in I_L \cup U_L} c_i \right].$$

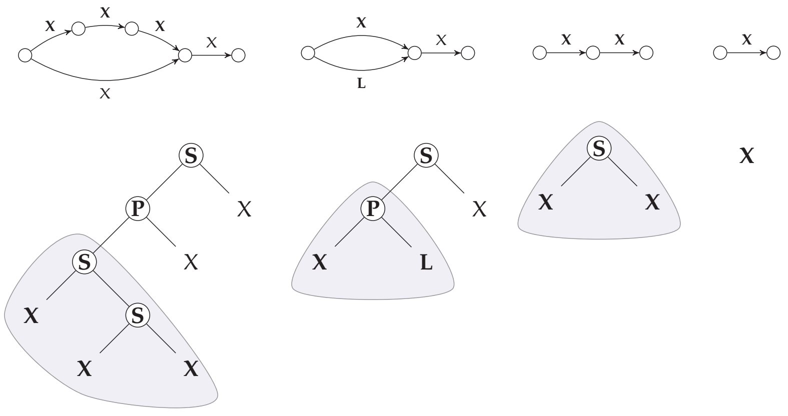
Virtually, we replace the parallel edges $e_1, \dots, e_{k_T}$ in G by the single edge e.

Before we derive the running time of the above contraction procedure, first note that each parallel tree and series tree can be determined by a single traversal of the decomposition tree in $\mathcal{O}(m)$ time in a preprocessing step. For each of the $\mathcal{O}(n)$ series trees, we need to compute the new intervals in_e and out_e, which results in an evaluation of $\mathcal{O}(k_T)$ outflow functions. Since each of these functions may result from the decomposition of other outflow functions from prior contraction steps, we obtain an overhead of $\mathcal{O}(n)$ per series tree. Similarly, each parallel tree T with k_T leaves causes an overhead of $\mathcal{O}(k_T)$. Since there are at most $\mathcal{O}(n)$ parallel trees (since the root of each parallel tree is either the root of the decomposition tree or a child of a series composition) and since each leaf in a parallel tree either corresponds to an edge in the original graph G or a series tree, the contraction steps of all parallel trees cause a total overhead of $\mathcal{O}(m + n)$. Hence, we obtain a total running-time of $\mathcal{O}(m + n^2)$ time for the reconstruction of the flow in each considered partition.

Partitioning procedure:

We now show how we can interleave the generation of the partitions into the contraction procedure described above while maintaining the claimed invariants. In particular, we are able to save some of the 3^m possible combinations by generating the partitions "just in time", i.e., prior to each contraction step.

Initially, we assume each edge to be unspecified. Suppose that the algorithm starts with the contraction of a series tree T with k_T leaves corresponding to the edges $e_1, \dots, e_{k_T}$. As described above, we do not need to consider any partition of these edges but can replace them by a new unspecified edge. At some point in time (if the graph does not consist of a single path), we come across a parallel tree T with k_T leaves corresponding to the edges $e_1, \dots, e_{k_T}$. At that time, all of these edges are unspecified. In order to fulfill the invariant required above, we need to consider different partitions of these edges into L, T, and U before the subsequent contraction step. According to Proposition 7.21, we can assume that *at most* one of these edges is of type T in a basis structure while the remaining edges are either empty or full. Equivalently, we can assess that *exactly* one edge remains *unspecified* while the remaining edges must be of type L or U. In total, we need to consider each of the k_T possible positions of the unspecified edge and, for every such position, each of the 2^{k_T-1} possible assignments of the remaining edges into L and U. We then contract the parallel tree into a single leaf, which is again unspecified as described above, and continue the procedure. An exemplary course of the overall procedure is depicted in Figure 7.9.



(a) Series tree (b) Parallel tree (c) Series tree (d) Result

Figure 7.9: Iterative contraction of series and parallel trees in the algorithm. For each iteration, the upper figure shows the current graph and the lower one the current decomposition tree. The series tree in (a) can be contracted to a single edge without considering any further partitions, which yields the graph shown in (b). Afterwards, the two leaves of the parallel tree need to be assigned to L, U, or X such that the tree can be contracted. This yields the tree shown in (c), which can immediately be contracted into the single leaf shown in (d).

Complexity of the overall algorithm:

The bound on the time needed for the contraction steps has already been derived above. It remains to prove that the number of partitions that need to be considered can be bounded by 2.707^m. To this end, note that the number of partitions only increases when contracting a parallel tree.

Now consider a series tree T with $k := k_T$ leaves $e_1, \ldots, e_k$ prior to its contraction. Several contraction steps before, each e_i either corresponded to a parallel tree T_i with k_i leaves or to a tree T_i with only $k_i = 1$ leaf e_i that corresponds to an edge in the original graph (cf. Figure 7.10).

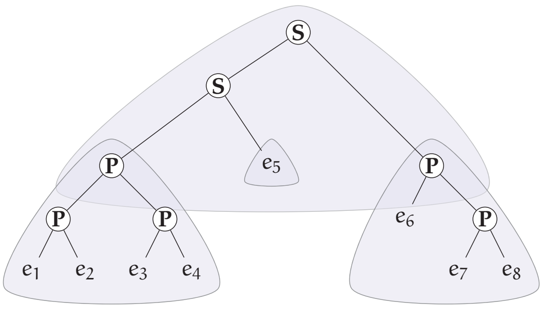


Figure 7.10: A series tree with $k = 3$ leaves and two parallel trees with $k_1 = 4$ and $k_3 = 3$ leaves. Before the algorithm contracts the series tree, it considers partitions for the two parallel trees and contracts these trees into single edges. It then contracts the series tree into a single edge.

Let $M(T)$ denote the number of partitions that need to be considered in order to process all of these trees T_i and the series tree T. As shown above, we get that

$$M(T) = \prod_{i=1}^{k} k_i \cdot 2^{k_i - 1}. \tag{7.2}$$

Since each of the considered trees is binary, it holds that the number n_i of nodes in each tree T_i is given by $2k_i - 1$ and that the number n_S of nodes in the series tree together with the nodes in the trees T_i is given by $n_S = \left(\sum_{i=1}^{k} n_i\right) + k - 1$. Note that, after the contraction steps of the trees T_i and the series tree T, the number of nodes in the decomposition tree is reduced by an absolute amount of $n_S - 1$. By substituting $k_i = \frac{n_i+1}{2}$ in (7.2), we can bound the number $M(T)$ of partitions as follows:

$$M(T) = \prod_{i=1}^{k} \frac{n_i + 1}{2} \cdot 2^{\frac{n_i - 1}{2}} = \prod_{i=1}^{k} \frac{n_i + 1}{2\sqrt{2}} \cdot 2^{\frac{n_i}{2}} = 2^{\sum_{i=1}^{k} \frac{n_i}{2}} \cdot \prod_{i=1}^{k} \frac{n_i + 1}{2\sqrt{2}}.$$

By using the inequality of arithmetic and geometric means (cf. Cauchy (1821)), we get that

$$\begin{aligned}
M(T) &\leqslant \left(\sqrt{2}\right)^{\sum_{i=1}^{k} n_i} \cdot \left(\frac{\sum_{i=1}^{k}(n_i+1)}{2\sqrt{2}\cdot k}\right)^k \\
&= \left(\sqrt{2}\right)^{\sum_{i=1}^{k} n_i} \cdot \left(\frac{1}{2\sqrt{2}} \cdot \left(\frac{\sum_{i=1}^{k} n_i}{k} + 1\right)\right)^k .
\end{aligned}$$

For $z := \frac{\sum_{i=1}^{k} n_i}{k} \geqslant 1$, we further obtain that

$$\begin{aligned}
M(T) &= \left(\sqrt{2}\right)^{\sum_{i=1}^{k} n_i} \cdot \left(\frac{1}{2\sqrt{2}} \cdot (z+1)\right)^{\frac{\sum_{i=1}^{k} n_i}{z}} \\
&= \left(\sqrt{2} \cdot \left(\frac{1}{2\sqrt{2}} \cdot (z+1)\right)^{\frac{1}{z}}\right)^{\sum_{i=1}^{k} n_i} .
\end{aligned}$$

It can be seen that the term $\sqrt{2} \cdot \left(\frac{1}{2\sqrt{2}} \cdot (z+1)\right)^{\frac{1}{z}}$ has a maximum value of

$$-4eW\left(-\left(2\sqrt{2}e\right)^{-1}\right) \approx 1.64524$$

for $z \geqslant 1$, where e is Euler's number and W denotes the Lambert W function. Hence, since $k \geqslant 2$, we get that

$$M(T) \leqslant 1.64524^{\sum_{i=1}^{k} n_i} = 1.64524^{n_S - k + 1} \leqslant 1.64524^{n_S - 1}.$$

Thus, in total, we only need to evaluate 1.64524^{n_S-1} partitions in order to remove $n_S - 1$ nodes from the decomposition tree. After contracting the series tree, we can repeat this procedure until the decomposition tree only consists of a single edge or of a single parallel tree. In any case, since we are interested in a maximum convex generalized flow, we do not need to consider any further partitions since it is clearly optimal to assign the remaining edges to U. Hence, since there are $2m-1$ nodes in the decomposition tree, we get the following bound on the total number M of partitions that need to be considered:

$$\begin{aligned}
M &\leqslant \prod_{\text{series tree}} 1.64524^{n_S-1} = 1.64524^{\sum_{\text{series tree}}(n_S-1)} \leqslant 1.64524^{2m-1} \\
&\leqslant 1.64524^{2m} \leqslant 2.707^{m},
\end{aligned}$$

which shows the claim. □

7.5.4 Extension-Parallel Graphs

As it was shown in Theorem 7.24, we can reduce the number of partitions that need to be considered from 3^m to 2.707^m by using a more sophisticated generation procedure for the problem of series-parallel graphs. When applying this algorithm to the convex generalized maximum flow problem on extension-parallel graphs, the number of partitions that need to be considered can be further reduced to 2.404^m as it is shown in the following corollary:

Corollary 7.25:
A maximum convex generalized flow can be computed in $\mathcal{O}(2.404^m \cdot m)$ time on extension-parallel graphs.

Proof: Assume that we apply the algorithm that was described in the proof of Theorem 7.24 to an instance of CGMFP on an extension-parallel graph G. Again, in order to bound the number of partitions that need to be evaluated, consider a series tree T with k leaves corresponding to the edges $e_1, \dots, e_k$. According to the structure of extension-parallel graphs, it holds that *at most one* edge e_j among these edges results from a prior contraction of a parallel tree T_j with k_j leaves into a single edge. Again, since all of the considered trees are binary, it holds that the number n_j of nodes in T_j is given by $n_j = 2k_j - 1$ and that the number of nodes n_S in the series tree T together with the nodes in T_j is given by $n_S = 2k - 1 + (n_j - 1)$. Since $k \geqslant 2$, we get that $n_j = n_S + 2 - 2k \leqslant n_S - 2$. The number $M(T)$ of partitions that need to be considered in order to process the parallel tree (together with the series tree) is then given by

$$M(T) \leqslant k_j \cdot 2^{k_j - 1} = \frac{n_j + 1}{2} \cdot 2^{\frac{n_j - 1}{2}} \leqslant \frac{n_S - 1}{2} \cdot 2^{\frac{n_S - 3}{2}} = \frac{n_S - 1}{4} \cdot \left(\sqrt{2}\right)^{n_S - 1}.$$

For $z := n_S - 1$, we then get that

$$M(T) \leqslant \frac{z}{4} \cdot \left(\sqrt{2}\right)^{n_S - 1} = \left(\left(\frac{z}{4}\right)^{\frac{1}{z}}\right)^{n_S - 1} \cdot \left(\sqrt{2}\right)^{n_S - 1} = \left(\sqrt{2} \cdot \left(\frac{z}{4}\right)^{\frac{1}{z}}\right)^{n_S - 1}.$$

The maximum of $\sqrt{2} \cdot \left(\frac{z}{4}\right)^{\frac{1}{z}}$ is given by $\sqrt{2} e^{\frac{1}{4e}} \approx 1.55045$. As in the proof of Theorem 7.24, we finally get that the number M of partitions that need to be considered is bounded by

$$\begin{aligned} M &\leqslant \prod_{\text{series tree}} 1.55045^{n_S - 1} = 1.55045^{\sum_{\text{series tree}} (n_S - 1)} \leqslant 1.55045^{2m - 1} \\ &\leqslant 1.55045^{2m} \leqslant 2.404^m, \end{aligned}$$

which shows the claim. □

7.5.5 Restricted Extension-Parallel Graphs

We close the study of graph classes with a special case of extension-parallel graphs that is solvable in polynomial time. As it was shown in Theorem 7.18, the problem CGMFP is $\mathcal{NP}$-hard to solve even if the underlying graph is restricted to be extension-parallel, i.e., if it is series-parallel but series compositions are only allowed in case that one of the two graphs consists of a single edge. We now show that CGMFP can be solved in linear time if we require that the *right hand side graph* (i.e., the graph whose source is identified with the sink of the other graph) in every series composition consists of a single edge. In the following, we refer to extension-parallel graphs with this additional restriction as *restricted* extension-parallel graphs.

Theorem 7.26:
A maximum convex generalized flow in a restricted extension-parallel graph can be computed in $\mathcal{O}(m)$ time.

Proof: Let $G = (V, E)$ be a restricted extension-parallel graph. The idea of the algorithm is to "pump" as much flow as possible into the graph in order to obtain a maximum preflow and to subsequently turn this preflow into a flow. For any series-parallel subgraph G' of G that corresponds to a node in the decomposition tree of G, we let $F(G')$ denote the maximum value of a convex generalized flow in G'. Starting from the leaves of the decomposition tree of G, these values $F(G')$ can be computed recursively as follows:

- If G' is a leaf of the decomposition tree corresponding to a single edge $e \in E$, we set $F(G') := g_e(u_e)$.
- If G' is the parallel composition of G_1 and G_2, we set $F(G') := F(G_1) + F(G_2)$.
- If G' is the series composition of G_1 and G_2, the right hand side graph G_2 must be a single edge e and we set $F(G') := g_e(\min\{F(G_1), u_e\})$.

Since each of the above steps requires only constant time $\mathcal{O}(1)$ and the decomposition tree contains $\mathcal{O}(m)$ nodes, this shows that we can compute the flow value $F(G)$ of a maximum convex generalized flow in G in $\mathcal{O}(m)$ time.

In order to compute the flow x_e on the edges of G in a maximum convex generalized flow x, we let $\text{out}_x(G')$ denote the outflow of each graph G' in the decomposition of G under x. Starting from the root of the decomposition tree, where we set $\text{out}_x(G) := F(G)$, each of these values $\text{out}_x(G')$ can be computed recursively as follows:

- If G' is the parallel composition of G_1 and G_2, we split the value $\text{out}_x(G')$ arbitrarily such that $\text{out}_x(G') = \text{out}_x(G_1) + \text{out}_x(G_2)$ and $\text{out}_x(G_1) \leqslant F(G_1)$, $\text{out}_x(G_2) \leqslant F(G_2)$.

- If G' is the series composition of G_1 and G_2, the right hand side graph G_2 must be a single edge e and we set $\text{out}_x(G_2) := \text{out}_x(G')$ and $\text{out}_x(G_1) := g_e^{-1}(\text{out}_x(G'))$.

Note that we always have that $\text{out}_x(G') \leqslant F(G')$ during the above procedure, so the splitting of $\text{out}_x(G')$ in case of a parallel composition is always possible and the procedure computes all values $\text{out}_x(G')$ in $\mathcal{O}(m)$ time. Afterwards, the flow x_e on each edge e can be computed from the value $\text{out}_x(e)$ obtained for the corresponding leaf of the decomposition tree as $x_e := g_e^{-1}(\text{out}_x(e))$. □

7.6 Integral Flows

We finally consider *integral flows* (i.e., feasible flows with integral in- and outflows for all edges) and assume that the outflow functions map integers to integers. Note that the $\mathcal{NP}$-completeness results from Theorem 7.17 and Theorem 7.18 remain valid for the case of integral flows. However, we are now able to derive a pseudo-polynomial-time algorithm for the problem on series-parallel graphs. In the following, let $U := \max_{e \in E} u_e$ and $\overline{U} := \max_{e \in E} g_e(u_e)$ denote the maximum possible inflow and outflow of an edge, respectively, which can be assumed to be integral as well without loss of generality.

Theorem 7.27:
A maximum integral convex generalized flow in a series-parallel graph can be computed in $\mathcal{O}(m^5 \cdot U^2 \cdot \overline{U}^2)$ time.

Proof: Consider a decomposition tree of G. For each component G' of this decomposition tree and for each value $x \in \{0, \ldots, m \cdot U\}$ and $y \in \{0, \ldots, m \cdot \overline{U}\}$, we compute the boolean function $A_{G'}(x, y)$, which is true if and only if an inflow of value x can produce an outflow of value y in G'.

Consider a leaf G' of the decomposition tree that corresponds to some edge e of the original graph G. For each $x \in \{0, \ldots, m \cdot U\}$ and $y \in \{0, \ldots, m \cdot \overline{U}\}$, we set $A_{G'}(x, y) :=$ TRUE if and only if $x \leqslant u_e$ and $g_e(x) = y$. If G' is the series composition of the two series-parallel graphs G_1 and G_2, we are able to achieve an outflow y with an inflow of x in G' if and only if there is some value $x' \in \{0, \ldots, m \cdot U\}$ that is both an outflow of G_1 and an inflow of G_2, i.e.,

$$A_{G'}(x, y) = \bigvee_{x' \in \{0, \ldots, m \cdot U\}} A_{G_1}(x, x') \wedge A_{G_2}(x', y).$$

Similarly, if G' is the parallel composition of the two series-parallel graphs G_1 and G_2, an outflow of y can be achieved with an inflow of x if and only if some amount y_1 of the outflow can be created with an inflow of x_1 in G_1 and the remaining outflow $y - y_1$ can be created with the remaining inflow $x - x_1$ in G_2. Hence, we get

$$A_{G'}(x, y) = \bigvee_{x_1 \in \{0,\dots,x\}} \bigvee_{y_1 \in \{0,\dots,y\}} A_{G_1}(x_1, y_1) \wedge A_{G_2}(x - x_1, y - y_1).$$

Note that there are $\mathcal{O}(m)$ nodes in the decomposition tree of G and we need to evaluate $\mathcal{O}((m \cdot U) \cdot (m \cdot \overline{U}))$ entries for each node. Clearly, for a single edge, each entry can be computed in constant time $\mathcal{O}(1)$. For the case of a series composition, we need to iterate over all possible values of x_1, which yields a complexity of $\mathcal{O}(m \cdot U)$ per entry. Finally, the evaluation of an entry for a node G' of the decomposition tree that corresponds to a parallel composition takes $\mathcal{O}((m \cdot U) \cdot (m \cdot \overline{U}))$ time. This yields the claimed running time of $\mathcal{O}\left(m \cdot (m \cdot U)^2 \cdot (m \cdot \overline{U})^2\right) = \mathcal{O}\left(m^5 \cdot U^2 \cdot \overline{U}^2\right)$. $\square$

7.7 Conclusion

We studied an extension of the generalized maximum flow problem in which the outflow of an edge is a strictly increasing convex function of its inflow. It turned out that the problem of computing a maximum convex generalized flow is strongly $\mathcal{NP}$-hard to solve even on bipartite acyclic graphs and weakly $\mathcal{NP}$-hard on extension-parallel graphs. For both cases and the case of preflows on general graphs, we presented exponential-time exact algorithms. Moreover, we showed that a flow decomposition similar to the case of traditional generalized flows is still possible and showed that the problem can be solved in pseudo-polynomial-time on series-parallel graphs for the case of integral flows. An overview of the results of this chapter is given in Table 7.1.

The model introduced in this chapter raises several interesting questions for future research. Since CGMFP was only shown to be weakly $\mathcal{NP}$-hard to solve on series-parallel graphs, it remains an open question whether a pseudo-polynomial-time algorithm exists also for the general case in which the flow is not restricted to be integral or whether the problem is actually strongly $\mathcal{NP}$-hard in this case. Furthermore, although the problem was shown to be $\mathcal{NP}$-hard to approximate, it remains open if and how approximate oracles could be used in order to obtain "almost feasible" solutions. Finally, although the running-time of the presented algorithms could be improved for the case of more simple graph classes, it may be possible to obtain faster algorithms by making even more use of the structure of such graph classes.

General Graphs	Acyclic Graphs	SP Graphs	EP Graphs
Theorem 7.13: Decomposable into m elementary subtractions	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$
$\longleftarrow$	**Theorem 7.17**: strongly NP-complete to solve	$\longleftarrow$	**Theorem 7.18**: weakly NP-complete to solve
$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	**Theorem 7.19**: NP-hard to approximate
Theorem 7.22: maximum preflow in $\mathcal{O}(3^m \cdot m)$ time	**Corollary 7.23**: maximum flow in $\mathcal{O}(3^m \cdot m)$ time	**Theorem 7.24**: maximum flow in $\mathcal{O}(2.707^m \cdot (m + n^2))$ time	**Corollary 7.25**: maximum flow in $\mathcal{O}(2.404^m \cdot (m + n^2))$ time
			Theorem 7.26: maximum flow in $\mathcal{O}(m)$ time on restricted extension-parallel graphs
		Theorem 7.27: maximum integral flow in $\mathcal{O}(m^5 \cdot U^2 \cdot \overline{U}^2)$ time	$\longrightarrow$

Table 7.1: The summarized results for the convex generalized maximum flow problem in Chapter 7. Implied results are denoted with gray arrows.