Column Generation for Generalized Min-Cost Flows with Losses

Jonas Alker Marc E. Pfetsch

TU Darmstadt
Department of Mathematics
{alker, pfetsch}@mathematik.tu-darmstadt.de

November 03, 2026

Abstract

The generalized flow problem deals with flows through a network with losses or gains along the arcs. Motivated by energy networks, this paper concentrates on the case with losses along cycles. Such networks can become extremely large, mostly because they are considered over large time horizons. We therefore develop a column generation approach for a path-based formulation. The pricing problems amount to finding shortest paths with loss factors, which we show to be solvable in $O(nm)$ by dynamic programming, where n is the number of nodes and m the number of arcs. We then perform a comprehensive computational comparison of the column generation approach on the path-based formulation and the dual simplex algorithm on the arc-based model. Solving the path formulation turns out to be superior to the arc formulation for instances with a large number of sources/demands and large capacities or if memory is critical. Using interior point (barrier) methods is not competitive. Moreover, the implemented stabilization scheme does not have a significant positive effect overall, but a path initialization heuristic does.

1 Introduction

The *generalized min-cost flow problem* (GMCF) is a well-known and well investigated problem, in which the arcs of a directed graph have efficiencies that allow for losses or gains of the flow. As the name suggests, it is a generalization of the classical flow problem in which the efficiencies are all 1. The problem has already been considered in Jewell (1962) and Dantzig (1963). The GMCF has a wide range of applications ranging from exchanges of currencies to modeling energy conversion processes, see Section 2.1. This paper is motivated by energy networks and we therefore restrict attention to losses, i.e., the efficiencies of all cycles are in the interval $(0, 1]$. In order to predict the behavior of future energy systems, often very large networks are considered. Although GMCF can be solved by linear programming (LP) approaches, the problem sizes pose a computational bottleneck. Additionally, if the problems are solved repeatedly, e.g., in a sensitivity analysis context, fast running times might be essential.

In this paper, we develop a column generation (CG) approach for a path-based formulation of GMCF with losses. To the best of our knowledge, this represents the first CG based approach to tackle a generalized flow problem, constituting a primary contribution of this work. Beyond that, we formally demonstrate that the corresponding pricing problem can be modeled as a shortest path problem and develop a specialized dynamic programming algorithm to solve it. To evaluate the efficacy of the proposed methodology, we conduct an extensive computational study on large-scale instances. To this end, we implement the column generation framework in C++ using SCIP (Hojny et al., 2025) and compare its performance against Gurobi's simplex method applied to an arc-based formulation.

We test our approach on generated and realistic energy network instances. The code and generated instances are publicly available.

The computational experiments demonstrate that the best method depends on problem parameters like the number of supplies/demands and the range of the capacities. If both are large or memory consumption is critical, the proposed CG approach achieves significant speedups establishing it as a method of choice, while the dual simplex is faster if these numbers are small. Notably, alternatives such as interior point (barrier) methods turn out to be much slower, even when omitting the crossover phase. Furthermore, we evaluate the integration of a stabilization method for the CG approach; however, our empirical results indicate that stabilization slightly increases running times instead of accelerating convergence. Conversely, initializing the master problem with heuristically generated paths proves to be very effective by dramatically accelerating the solving process. This initialization strategy is particularly potent for instances derived from real-world energy network models. Ultimately, our results establish this specialized exact method as a competitive, high-performance solver for large-scale generalized networks with losses, particularly under certain network parameter configurations.

This paper is structured as follows. In Section 2, the basic model is introduced and the current state of the literature is discussed. The CG approach is developed in Section 3. The computational results are presented in Section 4. We end the paper with conclusions in Section 5. Additional computational results are presented in the appendix.

2 Basic Problem Setting

Let (V, A) be a finite directed *graph* with a set of *nodes* V and a set of (directed) *arcs* A. An arc $a = (u, w) \in A$ is a pair of nodes where u is the *start* and w the *endnode* of a. For $v \in V$ we write $\delta^-(v)$ (resp. $\delta^+(v)$) for the set of incoming (resp. outgoing) arcs of v. Each arc $a \in A$ has a nonnegative *capacity* $u_a \in \mathbb{R}_+ \cup \{+\infty\}$ determining how much flow can be sent via arc a and a *cost* parameter $c_a \in \mathbb{R}_+$ describing the costs per unit. Additionally, for every arc $a \in A$ there is a positive *efficiency* parameter $\mu_a \in \mathbb{R}_{>0}$ indicating what proportion of the flow reaches the endnode of the arc. Moreover, each node v has a *supply* value $b_v \in \mathbb{R}$. The supply can be positive or negative, representing input and output nodes of the network, respectively. For this sake, we define $S := \{v \in V \mid b_v > 0\}$ and $D := \{v \in V \mid b_v < 0\}$. The amount of flow traversing an arc a is denoted by the variable x_a. Then the *generalized min-cost flow problem* is:

$$
\begin{aligned}
\min_{x} \quad & \sum_{a \in A} c_a \, x_a, \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} \mu_a \, x_a = b_v && \forall v \in V \setminus S, \\
& \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} \mu_a \, x_a \leq b_v && \forall v \in S, \\
& 0 \leq x_a \leq u_a && \forall a \in A.
\end{aligned}
\tag{GMCF}
$$

This problem is a strict generalization of the classical min-cost flow problem, in which case $\mu_a = 1$ for $a \in A$ and $\sum_{v \in V} b_v = 0$. Since in classical networks ($\mu_a = 1$) there are no losses or gains of flow, the inequality limiting the amount of input at nodes with positive supply value is always satisfied with equality if the demands are balanced ($\sum_{v \in V} b_v = 0$).

2.1 Literature Overview

We give a short overview over the literature on generalized min-cost flows: Jewell (1962) presented a primal-dual algorithm for the generalized problem with capacities. The formulation he considered fixes the node balance to zero for all nodes except for one source where it is fixed to a nonnegative amount. One year later, Dantzig (1963) presented an extension of the network simplex algorithm to handle

the generalized formulation. Ahuja et al. (1993) discuss possible data structures of the generalized network simplex algorithm and present the basis structure of the capacitated problem with a fixed node balance at every node.

The classical setting ($\mu_a = 1$ for all $a \in A$) is known to be solvable in strongly polynomial time, e.g., by Orlin's algorithm (Orlin, 1993). The algorithm solves the uncapacitated version with fixed node balances at every node. Solving the version with capacity constraints can be solved in strongly polynomial time as well, since the capacitated version can be transformed to an uncapacitated formulation on a bipartite graph, see, e.g., (Korte and Vygen, 2018, Lem.9.3). The question whether the generalization can also be solved in strongly polynomial time has been open until recently. Since the problem is a linear program, it is solvable in (weakly) polynomial time by a polynomial time algorithm like the ellipsoid method (Khachiyan, 1979). Wayne (2002) presented a polynomial combinatorial algorithm for the capacitated generalized circulation problem where all node balances are zero, extending the algorithms presented by Cohen and Megiddo (1994). A similar algorithm was given by Goldfarb and Lin (2002). Their combinatorial interior point method has the same worst case running time as the combinatorial algorithm by Wayne. Végh (2014) introduced a strongly polynomial algorithm for the generalized flow maximization. In this formulation with capacities the inflow at a node t should be maximized, while for every other node

$$\sum_{a \in \delta^-(v)} \mu_a x_a - \sum_{a \in \delta^+(v)} x_a \geq 0 \quad \forall v \in V \setminus \{t\}$$

should hold. For flow maximization, this formulation is equivalent to a formulation with equality in the node constraints (Shigeno, 2004). It is also equivalent to a version without arc capacities but with supplies:

$$\sum_{a \in \delta^-(v)} \mu_a x_a - \sum_{a \in \delta^+(v)} x_a \geq b_v \quad \forall v \in V \setminus \{t\}.$$

Indeed, every feasibility problem for systems of the form $Ax = b$, $0 \leq x \leq u$, where the matrix A has at most two non-zero entries per column, can be formulated as a generalized flow maximization problem (Végh, 2014). The strongly polynomial algorithm for this problem uses a scaling method and contracting arcs. In 2020 an improved version of this algorithm was presented by Olver and Végh (2020). The question of strongly polynomial solvability of the generalized min-cost flow problem was resolved by Dadush et al. (2024), based on an interior point method.

The generalized min-cost flow problem is important for the modeling of energy systems. Conversion processes of different commodities have the natural structure of arcs in the generalized flow setting. There are several modeling tools for energy systems in which generalized min-cost flow problems play a central role such as the well-known PyPSA (Brown et al., 2018), OSeMOSYS (Howells et al., 2011) and TIMES (Loulou et al., 2005) or the more recent models OCGModel (Barbosa et al., 2021) and PERSEUS-gECT (Slednev, 2024). All of these modeling tools rely on general purpose (mixed-integer) linear programming solving techniques to solve the models.

2.2 Alternative Formulations

Any instance of the generalized flow problem can be transformed into an equivalent formulation with a single source s and a single sink t. By introducing additional nodes s and t, arcs (s, v) for all input nodes $v \in S$ with capacity $b_v > 0$, arcs (v, t) for all output nodes $v \in D$ with capacity $-b_v > 0$ and defining $b_s := \sum_{v \in S} b_v > 0$, $b_t := \sum_{v \in D} b_v < 0$, the supply values of all original nodes can be set to zero. Here, all newly introduced arcs have cost 0 and efficiency 1. After applying this transformation on (GMCF), the constraint

$$\sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} \mu_a x_a \leq b_s$$

is redundant due to the capacities on the outgoing arcs of s and we obtain the formulation:

$$\begin{aligned}
\min_x \quad & \sum_{a\in A} c_a\, x_a, \\
& \sum_{a\in\delta^+(v)} x_a - \sum_{a\in\delta^-(v)} \mu_a x_a = 0 \quad && \forall v \in V\setminus\{s,t\}, \\
& \sum_{a\in\delta^+(t)} x_a - \sum_{a\in\delta^-(t)} \mu_a x_a = b_t, \\
& 0 \le x_a \le u_a && \forall a \in A.
\end{aligned} \tag{GMCF-A}$$

In the literature, generalized min-cost flow problems are typically modeled with equality constraints for every node. We can transform any instance of (GMCF) into such a formulation by introducing loops $a_v = (v, v)$ at every input node $v \in S$ with capacity $u_{a_v} = \infty$, costs $c_{a_v} = 0$ and efficiency $\mu_{a_v} = 0.5$ and changing the inequality constraint at v to an equality. In this way, the full supply is forced to be used, but the surplus can be consumed freely within the introduced loops.

In this paper we consider the special case of the classical formulation with inequalities at the input nodes as in (GMCF). This is motivated by the application to energy models, where demands should be satisfied strictly, but supplies should be determined in a cost-efficient way such that the flow balances out including the efficiencies. This assumption allows us to formulate a purely path-based formulation of (GMCF): While classical flows can be decomposed in paths and cycles, for the generalized version a generalized concept of flow decomposition is needed. We state such a decomposition in Lemma 1. Introducing s and t as described above, the GMCF from the literature (with equality constraints for every node) can be written as

$$\begin{aligned}
\min_x \quad & \sum_{a\in A} c_a\, x_a, \\
& \sum_{a\in\delta^+(v)} x_a - \sum_{a\in\delta^-(v)} \mu_a x_a = b_v \quad && \forall v \in V, \\
& 0 \le x_a \le u_a && \forall a \in A,
\end{aligned} \tag{GMCF-st}$$

where $b_s > 0$, $b_t < 0$ and $b_v = 0$ for every $v \in V \setminus \{s, t\}$. Applying flow decomposition on (GMCF-st) yields a representation that might also depend on flow consuming cycles connected to s, if the graph is not acyclic, to handle the full supply. In the context of energy systems, this corresponds to "burning" the surplus of energy in the cheapest possible way. We therefore use the form (GMCF), since in practice reducing the import seems to be preferable over wasting the excess. This way we can purely rely on s–t path variables, which we will show in the following section.

3 Column Generation for the Path Formulation

Recall from the previous section that any instance of (GMCF) can be transformed in the form (GMCF-A). In the following, we show that the model (GMCF-A) can be equivalently formulated by a purely path-based model if all arc costs are nonnegative and no flow generating cycles exist. A decomposition of generalized flows is given in Goldberg et al. (1991), Theorem 2.6. We present a variation of this decomposition adjusted to problem (GMCF-A). Since in generalized flow problems sending flow along a directed cycle does not necessarily conserve the flow, we follow the literature and define three different types of cycles. For a directed cycle C, define its *efficiency* as $\mu_C := \prod_{a\in C} \mu_a$. If $\mu_C = 1$, the cycle is called *flow conserving*. If $\mu_C > 1$ (resp. $\mu_C < 1$) the cycle is called *flow generating* (resp. *flow consuming*). For a given node $v \in V$ and $x \in \mathbb{R}^A_+$ we call $\sum_{a\in\delta^+(v)} x_a - \sum_{a\in\delta^-(v)} \mu_a x_a$ the *imbalance* of x at v.

Lemma 1 (Generalized Flow Decomposition). *Let x be feasible for (GMCF-A) with $b_t < 0$. Then there exist $x^i \in \mathbb{R}^A_+$ for $i = 1, \dots, k \leq |A|$ with*

$$x_a = \sum_{i=1}^{k} x_a^i$$

such that each x^i is exactly positive on a set of arcs of one of the following six types:
(a) a path from s to t (with positive imbalance at s and negative imbalance at t);
(b) a flow generating cycle and a path connecting the cycle to t (with negative imbalance at t);
(c) a flow generating cycle and a path connecting the cycle to s (with negative imbalance at s);
(d) a flow consuming cycle and a path connecting s to it (with positive imbalance at s);
(e) a flow conserving cycle;
(f) a flow generating cycle and a flow consuming cycle connected by a path.

Proof. Goldberg et al. (1991) provide a flow decomposition based on the sign of the imbalances of all nodes in the graph. For (GMCF-A) the imbalance of t is negative ($b_t < 0$) and for all $v \in V \setminus \{s, t\}$ it is zero. Therefore, for a solution x we consider the sign of the imbalance b_s at node s

$$b_s = \sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} \mu_a x_a.$$

If $b_s > 0$, Goldberg et al. (1991) allow to decompose the flow into types (a), (b), (d), (e) and (f), because s (resp. t) is the only node with a positive (resp. negative) imbalance. For $b_s < 0$, we obtain a decomposition into flows of types (a), (b), (c), (e) and (f), because no node with positive imbalance exists and s and t have a negative imbalance. Finally, for $b_s = 0$ the node t is the only node with a negative imbalance b_t and we get flows of types (a), (b), (e) and (f). Therefore any generalized flow can be decomposed in flows of the six types (a)–(f). □

The flow decomposition allows us to formulate the problem (GMCF-A) with variables corresponding to the six different flow types defined in (a)–(f). Using additional assumptions on the arc costs and efficiencies, we can show that there always exists an optimal solution of (GMCF-A) that can be decomposed into paths from s to t, i.e., applying Lemma 1 yields x^i which are all of type (a). This allows us to state the problem in a purely path-based formulation without variables for flows of types (b)–(f).

Lemma 2. *Let (GMCF-A) be feasible, $\mu_C \leq 1$ for all cycles C and $c_a \geq 0$ for all $a \in A$. Then there exists an optimal solution that can be decomposed into s–t paths.*

Proof. Since the problem is feasible and bounded, (GMCF-A) has an optimal solution x. By Lemma 1, x can be decomposed into types (a)–(f). The assumption $\mu_C \leq 1$ ensures that no flows of type (b), (c) and (f) exist. Flows of type (d) and (e) have no influence on the demand constraint or the flow conservation constraint for all nodes except s, since they do not create any excess at any node except for s. Due to the assumption on the arc costs, they have nonnegative cost. By removing flows of type (d) and (e) the solution stays feasible and the objective value does not increase. □

Let $\mathcal{P}$ be the set of simple directed paths from s to t. A path variable x_P represents the amount of flow that reaches the sink t via path $P \in \mathcal{P}$. In the generalized setting, this is not necessarily equal to the amount of flow sent from the source or the amount of capacity that is consumed by this path variable at arcs within the path. Let $\mu_P = \prod_{a \in P} \mu_a$ be the product of all efficiencies of arcs in a path $P \in \mathcal{P}$. We denote the product of efficiencies for the tail of path P starting at a certain arc $a \in P$ by:

$$\mu_{P,a} \coloneqq \prod_{\substack{e \in P, \\ e \geq a}} \mu_e,$$

where $e \geq a$ for two arcs e, $a \in P$ holds if e does not appear before a. Using $\mu_{P,a}$ we can compute the amount of flow $x_{P,a}$ on a certain arc a for any path variable x_P as follows:

$$x_{P,a} = \mu_{P,a}^{-1} x_P.$$

We can verify that using this notion, flow is indeed conserved along any path P: For two consecutive arcs $a_1 = (u, v)$ and $a_2 = (v, w)$ in an s–t path P with flow value x_P, flow conservation holds at node v considering only path P:

$$\mu_{a_1} x_{P,a_1} = \mu_{a_1} \Big(\prod_{\substack{e \in P, \\ e \geq a_1}} \mu_e \Big)^{-1} x_P = \Big(\prod_{\substack{e \in P, \\ e \geq a_2}} \mu_e \Big)^{-1} x_P = x_{P,a_2}. \tag{1}$$

Let $\mathcal{P}_a$ denote the subset of paths in $\mathcal{P}$ that contain arc $a \in A$. The cost of a solution is given by

$$\sum_{a \in A} c_a \sum_{P \in \mathcal{P}_a} x_{P,a} = \sum_{P \in \mathcal{P}} \sum_{a \in P} c_a \, x_{P,a} = \sum_{P \in \mathcal{P}} \Big(\sum_{a \in P} \mu_{P,a}^{-1} \, c_a \Big) x_P, \tag{2}$$

leading to the path-cost definition:

$$c_P \coloneqq \sum_{a \in P} \mu_{P,a}^{-1} \, c_a.$$

Theorem 3. *Let $\mu_C \leq 1$ for all cycles C and $c_a \geq 0$ for all $a \in A$. Then for every optimal solution of (GMCF-A) there exists an optimal solution of*

$$\begin{aligned} \min_{x} \quad & \sum_{P \in \mathcal{P}} c_P \, x_P, \\ s.t. \quad & \sum_{P \in \mathcal{P}} x_P = -b_t, \\ & \sum_{P \in \mathcal{P}_a} \mu_{P,a}^{-1} \, x_P \leq u_a \quad \forall a \in A, \\ & x_P \geq 0 \qquad\qquad\quad \forall P \in \mathcal{P}, \end{aligned} \tag{GMCF-P}$$

with the same objective value and vice versa.

Proof. Let y be an optimal solution of (GMCF-A). By Lemma 2, we can assume y decomposes into paths. Therefore, there exist paths P_i for $i = 1, \ldots, k \leq |A|$ and variables $z^{P_i} \in \mathbb{R}_+^A$ with $\sum_{i=1}^{k} z_a^{P_i} = y_a$ for all $a \in A$. We define the path variable x_{P_i} as the absolute value of the imbalance at the sink t of z^{P_i}, i.e.,

$$x_{P_i} \coloneqq \Big| \sum_{a \in \delta^+(t)} z_a^{P_i} - \sum_{a \in \delta^-(t)} \mu_a \, z_a^{P_i} \Big| = \mu_{a_i} \, z_{a_i}^{P_i}, \quad \text{for } i = 1, \ldots, k,$$

where $a_i \in \delta^-(t)$ is the last arc in P_i. One can easily show that x is feasible for (GMCF-P) with the same objective value.

Conversely, an optimal solution x for the path formulation can be easily transformed into a solution y for (GMCF-A) as follows:

$$y_a \coloneqq \sum_{P \in \mathcal{P}_a} x_{P,a} = \sum_{P \in \mathcal{P}_a} \mu_{P,a}^{-1} \, x_P \quad \forall a \in A.$$

Using (1) one can directly see that y satisfies the flow conservation constraint of (GMCF-A) for all $v \in V \setminus \{s, t\}$. Additionally, $\sum_{P \in \mathcal{P}} x_P = -b_t$ implies the satisfaction of the demand constraint of (GMCF-A). Therefore y is feasible and (2) shows that it has the same objective value as x. □

Formulation (GMCF-P) has $|\mathcal{P}|$ variables. Since for general graphs, the number of paths can be exponential in the number of nodes and arcs, we apply column generation to solve (GMCF-P). For an overview on column generation, see, e.g., Desrosiers and Lübbecke (2005); Lübbecke (2011). In an iterative process one repeatedly solves the *restricted master problem* (RMP), a restricted version of (GMCF-P) that considers only a subset of variables (paths). Given an optimal solution of the RMP, the so-called *pricing problem* is used to determine whether there exists a variable with negative reduced cost that is currently not in the RMP. If such a variable exists, it is added to the RMP. Otherwise, the solution of the RMP is optimal for the full formulation (GMCF-P). In the following, we show that the pricing problem for the generalized min-cost flow problem can be solved in strongly polynomial time with a variation of the Bellman-Ford shortest path algorithm if certain assumptions hold.

Let λ be the dual variable corresponding to the demand constraint and π_a for the capacity constraint of arc $a \in A$. Then, the dual problem of (GMCF-P) can be stated as:

$$\begin{aligned} \max_{\lambda,\pi} \quad & -b_t\,\lambda - \sum_{a\in A} \pi_a\,u_a \\ \text{s.t.} \quad & \lambda - \sum_{a\in P} \mu_{P,a}^{-1}\,\pi_a \le c_P \quad \forall P \in \mathcal{P}, \\ & \pi_a \ge 0 \qquad\qquad\qquad\quad \forall a \in A. \end{aligned} \tag{D-GMCF-P}$$

Let $x^\star$ and $(\lambda^\star, \pi^\star)$ be an optimal primal-dual solution pair for the RMP. Then $x^\star$ is optimal for (GMCF-P) if $(\lambda^\star, \pi^\star)$ is feasible for (D-GMCF-P). Recalling the definition $c_P = \sum_{a\in P} \mu_{P,a}^{-1}\,c_a$, this leads to the following pricing problem:

$$\text{Find } P \in \mathcal{P} \text{ with } \sum_{a\in P} \mu_{P,a}^{-1}\,(\pi_a^\star + c_a) < \lambda^\star \tag{PP}$$

or decide that $(\lambda^\star, \pi^\star)$ is feasible for (D-GMCF-P). In the former case, x_P is added to the RMP.

If the RMP is infeasible (e.g. when starting with an empty set of paths), one may introduce a slack variable with very large objective coefficient to ensure a bounded dual problem and obtain a finite optimal dual solution. Such a slack variable would correspond to an artificial arc from s to t with very high cost. Since choosing a suitable objective coefficient for the artificial variable may be difficult without creating numerical problems, we instead apply the approach of *Farkas pricing* (Achterberg, 2007) by setting $c_a = 0$ and using the dual ray that proves primal infeasibility in the pricing problem.

3.1 Generalized Shortest Paths

To solve the pricing problem (PP), we develop a dynamic programming algorithm (Bellman, 1957) for finding a generalized shortest path. As far as we know, this is the first time a shortest path problem of the form (PP) is studied. To this end, let W be a *walk*, i.e., any sequence of arcs such that consecutive arcs are connected via nodes. For two walks W_1 and W_2, where W_2 starts at the endnode of W_1, we write $W_1 + W_2$ for the concatenation of W_1 and W_2 in this order. If W_2 (resp. W_1) is of length 1, i.e., consists of only one arc a, we also write $W_1 + a$ (resp. $a + W_2$). For every arc a we have a positive efficiency parameter $\mu_a > 0$ and a cost parameter α_a. The length of a walk W is

$$\tilde{c}(W) \coloneqq \sum_{a\in W} \mu_{W,a}^{-1}\,\alpha_a.$$

When solving the pricing problem (PP) we define $\alpha_a \coloneqq \pi_a^\star + c_a$. Then, the assumption $c_a \ge 0$ and $\pi_a^\star \ge 0$ imply $\alpha_a \ge 0$ for all a and (PP) can be solved by checking whether $\min_{P\in\mathcal{P}} \tilde{c}(P) < \lambda^\star$. Note that the costs $\tilde{c}(P)$ for $P \in \mathcal{P}$ depend on the order of the arcs within the path and thus classical shortest path algorithms do not work directly.

Lemma 4. *Let W_1 and W_2 be two walks, such that W_2 starts at the endnode of W_1. Then*

$$\tilde{c}(W_1 + W_2) = \mu_{W_2}^{-1}\,\tilde{c}(W_1) + \tilde{c}(W_2).$$

Proof. Let $W \coloneqq W_1 + W_2$. For any arc $a \in W_1$ we have:

$$\mu_{W,a} = \prod_{\substack{e \in W, \\ e \geq a}} \mu_e = \mu_{W_2} \prod_{\substack{e \in W_1, \\ e \geq a}} \mu_e = \mu_{W_2}\, \mu_{W_1,a}.$$

Additionally, for an arc $a \in W_2$, we obtain:

$$\mu_{W,a} = \prod_{\substack{e \in W, \\ e \geq a}} \mu_e = \prod_{\substack{e \in W_2, \\ e \geq a}} \mu_e = \mu_{W_2,a}.$$

Using these two identities, we can directly show the statement of the lemma:

$$\begin{aligned}\tilde{c}(W) = \sum_{a \in W} \mu_{W,a}^{-1}\, \alpha_a &= \sum_{a \in W_1} \mu_{W,a}^{-1}\, \alpha_a + \sum_{a \in W_2} \mu_{W,a}^{-1}\, \alpha_a \\ &= \mu_{W_2}^{-1} \sum_{a \in W_1} \mu_{W_1,a}^{-1}\, \alpha_a + \sum_{a \in W_2} \mu_{W_2,a}^{-1}\, \alpha_a = \mu_{W_2}^{-1}\, \tilde{c}(W_1) + \tilde{c}(W_2).\end{aligned}$$ □

We call the cost function $\tilde{c}$ *conservative*, if extending any walk W by a cycle C (such that $W + C$ is a walk) does not decrease the overall cost, i.e., if $\tilde{c}(W + C) \geq \tilde{c}(W)$. Note that for linear cost functions this definition is equivalent to the classical definition in Korte and Vygen (2018). Lemma 4 allows us to show that the assumptions on the input data imply conservativeness of the cost function $\tilde{c}(W)$.

Lemma 5. *Let $\mu_C \leq 1$ for all cycles C and $\alpha_a \geq 0$ for all $a \in A$. Then the cost function $\tilde{c}$ is conservative.*

Proof. Since $\alpha_a \geq 0$ and $\mu_a > 0$ for every arc, we know that $\tilde{c}(W)$ is nonnegative for every walk W. Lemma 4 then directly implies for all walks W:

$$\tilde{c}(W + C) = \mu_C^{-1}\, \tilde{c}(W) + \tilde{c}(C) \geq \tilde{c}(W).$$ □

The conservativeness of the cost function implies that among all shortest s–v walks there exists a shortest s–v path. Lemma 4 and Lemma 5 give us the ability to formulate a result concerning optimality of subpaths.

Lemma 6. *Let $\mu_C \leq 1$ for all cycles C and $\alpha_a \geq 0$ for all $a \in A$. Let P be a shortest s–v path among all s–v paths with at most k arcs. Let arc $\bar{a} = (u, v)$ be the last arc of P. Then the path $Q \coloneqq P \setminus \{\bar{a}\}$ is a shortest s–u path with at most $k - 1$ arcs.*

Proof. Suppose there exists an s–u path Q' having at most $k-1$ arcs with $\tilde{c}(Q') < \tilde{c}(Q)$. Let $P' = Q' + \bar{a}$ be the s–u path Q' followed by the arc $\bar{a}$ and note that $P = Q + \bar{a}$. By Lemma 4:

$$\tilde{c}(P') = \mu_{\bar{a}}^{-1}\, \tilde{c}(Q') + \mu_{\bar{a}}^{-1}\, \alpha_{\bar{a}} < \mu_{\bar{a}}^{-1}\, \tilde{c}(Q) + \mu_{\bar{a}}^{-1}\, \alpha_{\bar{a}} = \tilde{c}(P).$$

If Q' does not visit v, P' is an s–v path with at most k arcs and $\tilde{c}(P') < \tilde{c}(P)$, contradicting the optimality of P.

Otherwise, let $Q'_{[s,v]}$ and $Q'_{[v,u]}$ be the subpaths of Q' split at v such that $Q' = Q'_{[s,v]} + Q'_{[v,u]}$. In this case, we show that $Q'_{[s,v]}$ is shorter than P. Note that $Q'_{[s,v]}$ has less than k arcs and $C \coloneqq Q'_{[v,u]} + \bar{a}$ gives a cycle at node v. We apply Lemma 4 twice in a similar way to obtain the following identities:

$$\tilde{c}(Q'_{[s,v]}) = \mu_{Q'_{[v,u]}} \big(\tilde{c}(Q') - \tilde{c}(Q'_{[v,u]})\big), \quad \tilde{c}(Q) = \mu_{\bar{a}}\, \tilde{c}(P) - \alpha_{\bar{a}}.$$

Together with $\tilde{c}(C) = \mu_{\bar{a}}^{-1}\,\alpha_{\bar{a}} + \mu_{\bar{a}}^{-1}\,\tilde{c}(Q'_{[v,u]})$ and the conservativeness of the cost function, see Lemma 5, we obtain:

$$\begin{aligned}
\tilde{c}(Q'_{[s,v]}) &= \mu_{Q'_{[v,u]}}\big(\tilde{c}(Q') - \tilde{c}(Q'_{[v,u]})\big) \\
&< \mu_{Q'_{[v,u]}}\big(\tilde{c}(Q) - \tilde{c}(Q'_{[v,u]})\big) \\
&= \mu_{Q'_{[v,u]}}\big(\mu_{\bar{a}}\,\tilde{c}(P) - \alpha_{\bar{a}} - \tilde{c}(Q'_{[v,u]})\big) \\
&= \mu_C\big(\tilde{c}(P) - \mu_{\bar{a}}^{-1}\,\alpha_{\bar{a}} - \mu_{\bar{a}}^{-1}\,\tilde{c}(Q'_{[v,u]})\big) \\
&= \mu_C\big(\tilde{c}(P) - \tilde{c}(C)\big) \\
&\leq \tilde{c}(P),
\end{aligned}$$

contradicting the optimality of P. □

Motivated by Lemma 6, we compute a shortest path with respect to $\tilde{c}(P)$ with a dynamic programming scheme. Let C_v^k be the costs of a shortest path from source s to node v with at most k arcs. We initialize these values as ∞ for every node, except s and $C_s^k = 0$. Let $n := |V|$. Then we check $n-1$ times whether we can extend the previously found paths by another arc. The procedure is summarized in Algorithm 1.

Algorithm 1: Dynamic Programming for Pricing Problem

Input: Graph (V, A), μ_a, α_a for all $a \in A$, start and end nodes s and t.
Output: The minimum cost of an s–t path w.r.t. $\tilde{c}(P) = \sum_{a\in P} \mu_{P,a}^{-1}\,\alpha_a$ and $\mathrm{pr}(v)$ for all $v \in V$ representing the predecessor of v in an optimal s–t path.

```
C_v^k ← ∞ for all v ≠ s and all k = 0, ..., n − 1;
C_s^k ← 0 for all k = 0, ..., n − 1;
for k = 1, ..., n − 1 do
    C_v^k ← C_v^{k−1} for all v ∈ V;
    for a = (u, v) ∈ A do
        if C_v^k > μ_a^{−1}(α_a + C_u^{k−1}) then
            C_v^k ← μ_a^{−1}(α_a + C_u^{k−1});
            pr(v) ← u;
return C_t^{n−1}, pr(v) for all v ∈ V;
```

Theorem 7. *Algorithm 1 correctly determines a shortest s–t path if $\alpha_a \geq 0$, $\mu_a > 0$ for all $a \in A$ and $\mu_C \leq 1$ for all cycles C. The running time is $O(nm)$, where $n = |V|$ and $m = |A|$.*

The proof works exactly as proving correctness of the standard Bellman-Ford algorithm with conservative costs, see, for example, Korte and Vygen (2018), Theorem 7.5. The formula for calculating the path-cost has been established in Lemma 4 and conservative costs are given due to Lemma 5. The running time is obvious.

Remark 1. The computed predecessors $\mathrm{pr}(v)$ of every $v \in V$ allow us to generate a path from s to t by backtracking the predecessors from t until we reach s. Indeed, Algorithm 1 computes costs of a shortest path C_v^{n-1} from s to every node $v \in V$. Therefore, we can find and create up to $|D|$ many paths per pricing round (recall that D is the set of original demand nodes) by checking the reduced costs $C_v^{n-1} + \alpha_a - \lambda^\star$ of an s–t path ending with arc $a = (v, t)$ for any $a \in \delta^-(t)$.

4 Implementation and Numerical Results

We implemented the above described dynamic programming algorithm in C++ as a pricing method within the SCIP Framework (Hojny et al., 2025), available at `scipopt.org`. For all numerical experiments, we use a prerelease version of SCIP 10.0.3 with Gurobi 13.0.1 as LP solver. SCIP allows

Table 1: Generated instances with different numbers of supply and demand nodes [nsupp] and different capacity ranges [maxcap].

Class	#Inst.	$\|V\|$	$\|A\|$	$\|S\|$	$\|D\|$	Capacity
nsupp25	10	20 000	1 000 000	25	25	[5, 15]
nsupp50	10	20 000	1 000 000	50	50	[5, 15]
nsupp75	10	20 000	1 000 000	75	75	[5, 15]
nsupp100	10	20 000	1 000 000	100	100	[5, 15]
nsupp125	10	20 000	1 000 000	125	125	[5, 15]
nsupp150	10	20 000	1 000 000	150	150	[5, 15]
maxcap5	10	20 000	1 000 000	25	25	[5, 5]
maxcap15	10	20 000	1 000 000	25	25	[5, 15]
maxcap25	10	20 000	1 000 000	25	25	[5, 25]
maxcap35	10	20 000	1 000 000	25	25	[5, 35]
maxcap45	10	20 000	1 000 000	25	25	[5, 45]
maxcap55	10	20 000	1 000 000	25	25	[5, 55]

the implementation of a custom pricer which is then included in the column generation loop. Note that we only solve LPs, no branching is performed. We will compare the performance of the column generation approach for (GMCF-P) with solving the arc formulation (GMCF-A) directly, using Gurobi 13.0.1 as LP solver. All experiments were run on a Linux cluster with 3.5 GHz Intel Xeon E5-1620 Quad-Core CPUs, having 32 GB main memory and 10 MB cache each. All computations were run single-threaded. Our implementation can be found at `https://github.com/dopt-TUDa/gmcf`.

4.1 Test Instances

We test our implementation on various random instances generated by the GNETGEN (Glover, 1992) graph generator, a modification of NETGEN (Klingman et al., 1974). We use this generator with only slight modifications (increasing the maximal instance sizes and precision on efficiency parameter). We generate various instance classes with different properties by varying the number of demand/supply nodes and the maximal capacity. The input data and the graph files can be found in `https://github.com/dopt-TUDa/GenFlowGraphs`.

An instance class contains multiple graphs with the same structural properties, generated with different random seeds. An overview over our test classes can be found in Table 1. For every instance class (Class), we show the number of instances in the class (#Inst.), the number of nodes ($|V|$) and arcs ($|A|$) of a graph in this class, the number of supply nodes ($|S|$) and demand nodes ($|D|$) and the capacity range for all arcs (Capacity). As mentioned in the introduction, we can always transform instances with multiple supply or demand nodes into one with a single source and a single target by introducing artificial nodes and arcs. For all instances we generate random efficiencies with $\mu_a \in [0.9, 1.0]$ and costs with $c_a \in [1.0, 100.0]$ for all $a \in A$. The total supply is 100 000 and the total demand is 10 000 randomly distributed over all supply/demand nodes.

4.2 Initial Heuristic

Since we have no guarantee that, during Farkas pricing, the generated paths are helpful for finding an optimal solution, we additionally try to generate high quality paths heuristically. To this end, once at the beginning of the solving process, we iteratively compute a cheapest s–t path with respect to the original cost (ignoring the dual variables) with Algorithm 1 and send as much flow along it as allowed by the current capacities. We then reduce the capacities on the path by the sent flow and iterate until we reach primal feasibility or no more paths can be generated, since t is no longer reachable from s. If the heuristic finds a feasible solution, the initial RMP can be warm-started. In this case it always

provides a feasible basis, since flow values are maximized on generated paths. If the heuristic does not find a feasible solution, we still obtain a set of initial paths, which hopefully helps reducing the number of Farkas pricing iterations.

4.3 Separation of Capacity Constraints

Another challenge is that the path formulation contains a linear constraint for every arc in the network. The arc formulation (GMCF-A) has the advantage that the capacity constraints on the arcs are given as variable bounds which can be handled more efficiently than other linear constraints, since the bounds do not increase the size of a basis. Since we expect that the majority of the capacity constraints are not active in an optimal solution, we solve the master problem in a price-and-cut loop. The first RMP uses the heuristically generated paths and the capacity constraints of the arcs that are active for these path variables. After the relaxed RMP is solved to optimality, using the column generation approach, it is checked whether the obtained solution satisfies all capacity constraints and possibly violated constraints are added to the relaxed RMP. This process is repeated until the solution for the relaxed RMP is feasible for the master problem (GMCF-P). Then the solution is optimal for the master problem. This technique leads to small intermediate linear programs in comparison to the network size, if we only need a small number of paths to find the optimal solution. For the experiments we activate the initial paths heuristic and the separation of capacity constraints. Recall that (GMCF-A) has $|A| = 1\,000\,000$ columns and $|V| - 1 = 19\,999$ rows. The linear programs solved by the path formulation are very small in comparison; see #cols and #rows in Table 3 showing the sizes of the last RMP solved during the price-and-cut loop. Especially if the available memory is an issue, the price-and-cut method may be preferred due to the vastly reduced sizes of the linear programs.

The downside of the method is the necessity of solving not only a single linear program but many of them and the additional execution of the pricing algorithm. Note that the process of separating the capacity constraints is automatically performed by SCIP in our implementation. In SCIP this can be achieved by creating the linear constraint, but not adding it to the initial LP.

Remark 2. To illustrate the memory advantage of the CG method, we have created a very large instance, using the GNETGEN generator with 20 000 000 arcs. We were not able to solve the instance in the arc formulation (GMCF-A) despite using another machine with 64 GB main memory. Switching to CG on the path formulation, however, allows us to solve it within 300 s. We highlight that for instances of this size, the dynamic programming method is the main bottleneck for CG, since the RMP sizes do not change drastically. The instance can be found in the same github repository as the other instances.

4.4 Stabilization

We have also investigated the effect of stabilization on the column generation process by applying the in-out stabilization technique introduced by Pessoa et al. (2013). The idea is to stabilize the dual solution π^k at iteration k with another feasible dual solution $\bar{\pi}$, the *stabilization center*. In the pricing algorithm, the dual variables are replaced by a convex combination $\alpha\pi^k + (1-\alpha)\bar{\pi}$ for some $\alpha \in [0, 1]$. We observe that 0 is a feasible point for the dual problem (D-GMCF-P), which allows to use this point as the stabilization center ($\bar{\pi} = 0$). Alternatives are taking the dual solution of the previous iteration ($\bar{\pi} = \pi^{k-1}$) or the currently best known dual solution ($\bar{\pi} = \tilde{\pi}$), i.e., the dual solution that induced the current dual bound, to stabilize the dual variables. Using modified dual variables may lead to *mispricing* (Pessoa et al., 2013), i.e., the pricing algorithm fails to find a negative reduced cost column/path although there exists one. Therefore, whenever we fail to find such a path, we rerun the pricing again with the correct dual variables. After a mispricing round, we increase α as described by Pessoa et al. (2013) for the next pricing round and turn off stabilization completely, once α is sufficiently close to 1.0. By using the stabilization technique we were able to slightly reduce the number of generated paths at the cost of higher solving times and a higher percentage of running time spent on the pricing problem. We provide numerical evidence for this in Appendix A.

Table 2: Solving times in seconds, using Gurobi as LP solver.

	(GMCF-A)			(GMCF-P)			
Class	average	min	max	average	min	max	wins
nsupp25	36.17	34.61	37.60	54.31	47.83	70.69	0
nsupp50	32.91	30.70	37.47	33.19	26.25	48.45	5
nsupp75	30.77	28.59	32.75	20.23	16.13	24.61	10
nsupp100	29.55	27.58	31.80	18.30	15.23	24.25	10
nsupp125	28.41	27.63	29.63	16.22	14.18	20.99	10
nsupp150	28.00	27.34	30.57	11.91	10.56	13.65	10
maxcap5	39.92	38.88	41.53	143.69	106.74	181.75	0
maxcap15	36.81	34.72	39.80	64.80	42.57	101.14	0
maxcap25	35.40	34.29	38.21	40.95	30.34	58.86	3
maxcap35	35.63	33.74	40.32	32.34	23.05	50.05	8
maxcap45	34.84	32.57	39.45	26.04	18.44	39.61	9
maxcap55	35.29	33.61	38.99	20.64	15.87	28.21	10

4.5 Numerical Results

We compare the CPU times of solving the arc formulation (GMCF-A) with our column generation approach on the path formulation (GMCF-P) on the instances explained in Section 4.1. When solving the path-formulation we use the initial heuristic (Section 4.2) and the separation of capacity constraints (Section 4.3) for all test runs. We can observe that both methods significantly improve the running time for the path formulation. We provide empirical evidence for this in Appendix B. We evaluate the solving times only on very large instances ($|A| = 1\,000\,000$). Note that we have already observed that the number of arcs cannot be increased arbitrarily without running into memory limitations on the arc formulation. We solve every instance to optimality. For the arc formulation we always use the dual simplex method. Alternatively one could apply a barrier or a crossover method to solve linear programs, but we can observe that using the dual simplex method is superior on the arc formulation for our instances. The running times of the different LP solving methods (simplex, barrier, crossover), using CPLEX or Gurobi can be found in Appendix C.

We compare the solving times of the arc and path formulation, using Gurobi as LP solver. In Appendix D we provide additional computations using the LP solvers SoPlex 8.0.2 (Wunderling, 1996; Hojny et al., 2025), available at `soplex.zib.de` and CPLEX 22.1.2.0. Table 2 shows the average (average), minimal (min) and maximal (max) solving times in seconds for instances of a certain class, using Gurobi for the arc and path formulations. Additionally the column 'wins' shows how often (GMCF-P) was solved at least as fast as (GMCF-A) (with a tolerance of 10^{-6}). Recall that the arc formulation is solved with a simplex algorithm and the path formulation by our price-and-cut method, using the dynamic programming scheme to solve the pricing problem. Let us first focus on the results solving the arc formulation. In general, increasing the number of supply/demand nodes seems to make the problems easier (nsupp25: 36.17 s, nsupp150: 28.00 s) and the same holds for increasing the maximal capacity (maxcap5: 39.92 s, maxcap55: 35.29 s).

Looking at the path formulation, this observation can be confirmed. The solving times drastically decrease when increasing the number of supplies/demands (nsupp25: 54.31 s, nsupp150: 11.91 s) or the maximal capacity (maxcap5: 143.69 s, maxcap55: 20.64 s). A more detailed analysis of the solving behavior for the price-and-cut method can be seen in Table 3. Here, the number of paths generated (#paths), the number of used paths in the computed optimal solution (solsize), the size of the last RMP (#cols, #rows) during the price-and-cut loop, the overall number of iterations the LP solver takes (LPiter) and the percentage of the running time spent on the initial heuristic (heur%), the pricing problem (PP%) and the RMP (RMP%) is provided. The remainder of the running time mainly goes

Table 3: Analysis of price-and-cut, using Gurobi as LP solver.

Class	#paths	solsize	#cols	#rows	LPiter	Heur%	PP%	RMP%	h-gap%
nsupp25	7190.4	2754.6	6370.1	3612.9	109573.4	27.11	58.52	11.24	8.01
nsupp50	6809.7	2767.8	6201.1	3574.0	68260.1	26.64	59.03	9.99	8.11
nsupp75	6150.4	2732.4	5551.8	3481.0	39069.4	27.70	58.65	7.58	7.41
nsupp100	5921.9	2726.4	5304.6	3448.9	31349.2	27.12	59.82	6.48	7.10
nsupp125	5595.0	2709.7	5082.4	3399.0	24236.8	27.37	60.05	5.39	6.63
nsupp150	5295.9	2653.8	4832.6	3335.7	18981.3	27.67	58.20	5.03	6.33
maxcap5	14181.9	4730.4	8751.2	5237.0	407647.1	20.89	47.59	28.43	8.80
maxcap15	7306.0	2758.8	6532.4	3618.2	115685.9	28.28	58.34	10.48	8.08
maxcap25	4983.9	2012.9	4633.0	2704.0	56014.0	28.88	61.54	6.22	8.02
maxcap35	3798.5	1604.4	3609.4	2173.6	35150.7	29.96	62.09	4.14	7.69
maxcap45	3070.4	1357.4	2906.9	1839.5	23814.9	30.71	61.78	3.16	7.24
maxcap55	2575.0	1174.3	2423.7	1585.3	17698.2	31.24	60.89	2.64	7.27

into checking and separating the linear capacity constraints and is not listed in the tables. Again, all values are averaged over all instances in a certain instance class. Additionally, the last column (h-gap%) presents the average gap of the initial heuristic for instances where it found a solution in percent. The gap is $(h - \text{opt})/\text{opt}$, where h is the value of the heuristic and opt the optimal value.

We can observe that over all instances, around 60 % of the running time goes into solving the pricing problem. The only exception here is the maxcap5 class (47.59 %). This can be explained by the necessity of more paths in a solution (solsize: 4730.4) due to the tighter capacities leading to a significant increase in the RMP sizes (maxcap5 #cols: 8751.2, maxcap15 #cols: 6532.4). Considering the quality of the heuristically found solution, we observe that the quality of the heuristic solution is overall very good, having a gap between 6 % and 9 % to the optimal value. With increasing supply/demand nodes and with increasing maximal capacity, the quality of the heuristic solution improves. For the nsupp instances, this can be explained by the fact that having more sources and sinks but a comparable number of paths within optimal solutions, the paths of an optimal solution may be distributed more evenly across the graph. This way, there may be fewer "bottleneck" arcs leading to suboptimality of the heuristic solution. For the maxcap instances, we can observe that with increasing maximal capacity fewer paths are needed (see solsize) giving the cheapest paths a higher weight within optimal solutions.

Comparing the two approaches, we can observe that the performances depend on the class characteristics. Let us first focus on the nsupp instance classes. Having a very small number of sources and sinks, the simplex method on the arc formulation seems to be superior (nsupp25: 36.17 s vs. 54.31 s) with 0 wins for the price-and-cut method. In Table 3 we can see that more than half of the running time (58.52 %) is used for solving the pricing problem. Even though only 2754.6 paths (on average) are needed to find an optimal solution, the pricer generates 7190.4 paths. The two approaches seem to be comparable when increasing the number of supply nodes to 50 (nsupp50: 32.91 s vs. 33.19 s) with 5 wins for the price-and-cut method. While the average time solving (GMCF-P) is a bit higher, the fastest solving time is lower (nsupp50-min: 30.70 s vs. 26.25 s) and the longest solving time higher (nsupp50-max: 37.47 s vs. 48.45 s). On instances with a higher number of supply/demand nodes, the price-and-cut method is superior (nsupp150: 28.00 s vs. 11.91 s) with 10 wins. Still, the pricing problem takes more than half of the running time (58.20 %), but this time we generate far less unnecessary paths (#paths: 5295.9, solsize: 2653.8). In our computations, the price-and-cut method beats the simplex method on all instances with at least 75 supply and demand nodes, i.e., already for a relatively low number compared to the network size of 20 000 nodes. In fact, considering all instances from the classes nsupp75–150, the worst performance of price-and-cut on (GMCF-P) (24.61 s) is faster than the best performance of Gurobi's dual simplex on (GMCF-A) (27.34 s).

Table 4: Avg. solving times in seconds on instances with $c_a = 10$ for all $a \in A$, using Gurobi as LP solver..

	(GMCF-A)			(GMCF-P)			
Class	average	min	max	average	min	max	wins
nsupp25	31.29	30.09	33.57	22.34	17.27	29.29	10
nsupp50	25.58	24.66	26.64	9.37	7.42	11.92	10
nsupp75	24.56	23.99	25.40	4.83	3.75	6.05	10
nsupp100	24.63	23.60	25.05	4.11	3.22	4.88	10
nsupp125	25.51	24.49	26.53	3.49	2.97	4.17	10
nsupp150	27.85	26.41	30.71	3.17	2.91	3.47	10
maxcap5	33.86	30.74	38.23	66.41	43.33	90.57	0
maxcap15	31.77	29.15	34.39	28.29	19.40	42.56	8
maxcap25	30.98	28.11	34.57	12.48	8.82	18.34	10
maxcap35	30.14	28.70	31.75	9.00	7.62	12.88	10
maxcap45	30.87	27.59	34.82	6.36	4.66	7.49	10
maxcap55	29.37	27.66	32.45	5.03	4.00	7.46	10

We consider the maxcap instances next. Here, we use 25 supply and demand nodes for all instances. With a capacity range of [5, 15] we have already seen that solving the arc formulation is superior (note that the maxcap15 instances have the same network properties as the nsupp25 instances). We investigate whether increasing the maximal capacity leads to the path formulation being superior on graphs with 25 supply and demand nodes. We can expect that the number of paths needed for optimal solutions decreases with increasing maximal capacity and therefore the path formulation should clearly benefit from increasing the maximal capacity. This can be confirmed by looking at the columns 'solsize' and '#paths' in Table 3 and the solving times in Table 2. With higher maximal capacity, less paths are needed for optimal solutions (maxcap5: 4730.4; maxcap55: 1174.3) leading to less generated paths, e.g., 14 181.9 for maxcap5 and 2575.0 for maxcap55. This results in a shorter running time (maxcap5: 143.69 s; maxcap55: 20.64 s). With tighter capacities, the price-and-cut method generates a lot of paths that are not part of the optimal solution and is not competitive with the arc formulation solved by Gurobi (maxcap5: 39.92 s vs. 143.69 s). Increasing the maximal capacity shifts this behavior such that at a capacity range of [5, 35] (maxcap35), the running times are comparable (maxcap35: 35.63 s vs. 32.34 s) with 8 wins for the price-and-cut method. Increasing the maximal capacity even further, we can observe that the path formulation is now clearly superior (maxcap55: 35.29 s vs. 20.64 s) with 10 wins. For these instances, especially the relative time for solving the linear programs decreases: 28.43 % for maxcap5, 2.64 % for maxcap55. This can be explained by the decreased RMP sizes due to larger capacities.

4.6 Investigating Different Cost Structures

In this section we evaluate the influence of the arc costs and efficiencies on the solution times. Recall that previously the arc costs (resp. efficiencies) were chosen randomly from $[1, 100]$ (resp. $[0.9, 1.0]$). Here, we fix c_a and μ_a for all arcs $a \in A$ to a specific value and compare the solving times of the dual simplex on (GMCF-A) and CG on (GMCF-P) again. Table 4 shows the solving times with fixed arc costs $c_a = 10$, Table 5 with fixed arc efficiency $\mu_a = 0.95$ and Table 6 applying both modifications.

We can observe from the numerical results that especially fixing the arc costs to a specific value results in easier instances and therefore reduced solving times. While this is also true for the arc formulation, for the CG method the decrease in solving times is even more drastic. Fixing the arc costs to 10, allowed us to solve all instance from nsupp100–150 within 5 s, while Gurobi takes around 25 s for (GMCF-A). Indeed, CG is now superior on all instances, except for the maxcap5–15 instances

Table 5: Avg. solving times in seconds on instances with $\mu_a = 0.95$ for all $a \in A$, using Gurobi as LP solver.

	(GMCF-A)			(GMCF-P)			
Class	average	min	max	average	min	max	wins
nsupp25	31.16	2.98	38.61	48.80	21.86	62.59	0
nsupp50	30.49	29.16	32.49	32.20	25.16	47.36	4
nsupp75	29.03	27.56	30.35	20.46	15.96	24.23	10
nsupp100	28.29	26.45	31.15	18.09	14.94	23.78	10
nsupp125	27.16	26.11	28.41	16.35	13.53	21.53	10
nsupp150	26.93	25.27	29.53	12.33	11.41	14.21	10
maxcap5	36.90	36.11	38.31	133.54	97.60	168.68	0
maxcap15	34.42	31.54	40.87	65.96	42.64	106.99	0
maxcap25	33.30	31.71	34.27	40.30	31.07	57.80	3
maxcap35	32.67	30.78	35.74	32.63	23.32	51.40	6
maxcap45	33.16	32.30	35.42	25.81	18.30	38.90	9
maxcap55	34.51	32.66	39.71	20.34	13.97	28.21	10

with the tightest capacities. The dual simplex is faster on maxcap5 (33.86 s vs. 66.41 s, 0 wins for CG), while on the maxcap15 instances the two approaches seem to be comparable (31.77 s vs. 28.29 s, 8 wins), see Table 4. When fixing the arc costs and efficiencies (Table 6), the solution times decreased even more. Here, for all nsupp50–150 and maxcap25–55 instances the longest solving time of the CG approach is still faster than the fastest solving time of the dual simplex. This advantage of the price-and-cut method over the arc formulation can be explained mainly by the influence on our initial primal heuristic. If the cost of a path variable only depends on the length of the path (which is the case when arc costs and efficiencies are constant), the heuristic seems to produce higher quality solutions leading to fewer pricing rounds. We provide empirical evidence for this by considering the optimality gap of the heuristic solution. While this gap for heuristic was between 6 % and 9 % for the original instance, this gap decreases for the modified instances. With constant costs and random efficiencies, the optimality gap for the heuristic solutions are between 0.24 % for maxcap55 and 3.89 % for maxcap5. For the nsupp instances we have 0.84 % for nsupp150 and 2.51 % for nsupp25. Fixing costs *and* efficiencies gives even smaller gaps. The optimality gaps of the heuristic are between 0.16 % for maxcap55 and 0.53 % for maxcap5. For nsupp instances the results are similar. Fixing only the efficiency to 0.95 did not change the results drastically, see Table 5. The same applies for the heuristic solution quality, which is (similar to the original instances) between 6.41 % for nsupp150 and 10.50 % for maxcap5. We have also tested the influence of the stabilization scheme of Pessoa et al. (2013) on the instances with constant costs and efficiencies leading to the same conclusions as for the original instances.

4.7 Application on Energy Systems Models

We test our column generation method on two energy models and compare the running time with solving the arc formulation directly. It should be mentioned that both models arise as network design problems, where the decision is which conversion processes and storages should be built and how much capacity has to be installed to be able to satisfy the upcoming energy demands while minimizing expenses. Since we do not control the capacities with the generalized min-cost flow model, we use fixed capacities and only solve the remaining flow problem. Again, we compare the running time of using the dual simplex algorithm on the arc formulation (GMCF-A) and the CG method on (GMCF-P).

The first problem was provided by Siemens in the context of a joint research project and is a basic Power-to-Heat (P2H) model with two main conversion processes generating heat energy. The

Table 6: Avgerage solving times in seconds on instances with $c_a = 10$ and $\mu_a = 0.95$ for all $a \in A$, using Gurobi as LP solver.

	(GMCF-A)			(GMCF-P)			
Class	average	min	max	average	min	max	wins
nsupp25	21.80	2.98	31.96	18.81	10.88	32.40	7
nsupp50	25.78	23.39	27.05	7.35	5.56	9.23	10
nsupp75	24.42	23.11	26.02	4.21	3.53	4.71	10
nsupp100	23.40	22.57	24.27	4.05	3.47	5.02	10
nsupp125	22.86	20.22	26.43	3.74	3.41	4.33	10
nsupp150	22.27	18.73	26.15	3.30	3.14	3.54	10
maxcap5	33.62	25.47	39.24	50.29	32.05	79.16	1
maxcap15	23.77	22.10	31.28	27.22	15.30	36.97	4
maxcap25	20.83	19.95	21.59	9.64	7.89	11.56	10
maxcap35	20.81	19.60	22.44	7.65	6.66	10.85	10
maxcap45	20.76	19.52	22.86	5.37	2.56	6.59	10
maxcap55	20.08	19.19	21.22	3.52	2.16	6.46	10

Table 7: Solving times for energy models, using Gurobi as LP solver.

Model	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert S\rvert$	$\lvert D\rvert$	(GMCF-A)	(GMCF-P)
P2H	122 640	148 920	21 899	34 910	2.46	0.54
CESM	365 000	1 058 450	18 250	109 500	24.56	2.60

objective is to satisfy the demanded heat over a planning horizon of one year. The heat can either be generated by a heat pump consuming electricity or alternatively by a gas boiler. The demands are given by the heat and electricity consumption of private households and industry, while the supply of electricity and gas is limited by the capacity of the power grid, the availability of solar energy and gas pipelines at different time steps within the year. Imports and conversions of different commodities can be modeled via arcs within a network. Storages for electricity and hot water connect the networks of consecutive time steps leading to an increased model size when increasing temporal resolution, i.e., the number of time steps within the planning horizon. A demand series, i.e., costs for imports and varying availabilities of solar energy lead to different network parameters at different time steps.

The second problem is based on the CESM tool (Hajikazemi and Barbosa, 2024), which provides a modeling tool of the German energy supply and demand and is an extension to the OCGModel modeling tool (Barbosa et al., 2021). In the model, energy can be obtained from several sources like gas, electricity, nuclear power or waste heat. Moreover, there are different energy conversion processes. The model also contains an electricity storage connecting the networks of consecutive time steps. Therefore, the problem size highly depends on the temporal resolution, i.e., the number of time steps within the time horizon.

The results for the two models are presented in Table 7. For the P2H network, we choose the maximum available number of time steps, i.e., one for each hour within a year (8760). For the CESM network we choose a time horizon of 50 years with 365 time steps per year, one for each day. Both networks can be found in `https://github.com/dopt-TUDa/GenFlowGraphs`. We can observe that the P2H network can be solved very fast by both solving methods ((GMCF-A): 2.46 s, (GMCF-P): 0.54 s). Our initial greedy heuristic found the optimal solution for the P2H instance. Therefore a single pricing round and a single feasibility check for every capacity constraint was sufficient to solve the instance with CG. Note that the instance is far smaller in comparison to the instances of the previous section ($|A| = 148\,920$ instead of $1\,000\,000$) which leads to the faster solving time of the simplex algorithm. However, the CESM instance is comparable in size with the previously

investigated instances ($|A| = 1\,058\,450$) while the density is far lower with 365 000 instead of 20 000 nodes. Additionally, the number of supply (18 250) and demand nodes (109 500) is much higher than in the previous section. Recall that the relative performance of the CG method is getting better with increasing the number of supply and demand nodes compared to the simplex method on the arc formulation. The results on the CESM instance confirm this observation. While the arc formulation is solved in 24.56 s, CG solves the path formulation within 2.60 s. Again, the initial greedy heuristic proved its value by finding an optimal solution.

Note that the presented results are not necessarily an indicator for the path formulation to be superior in comparison to the arc formulation for large-scale energy models. However, for the given practical examples, we could observe the advantage of using the initial heuristic.

5 Conclusion

In this article we have shown that the generalized min-cost flow problem can efficiently be solved by column generation if the following assumptions are met: There are no flow generating cycles, we have nonnegative costs and no flow constraint for the source node. These assumptions are motivated by energy networks. We have seen that the pricing problem of the path-formulation turns out to be a generalized shortest path problem, which can be solved in $O(nm)$ time by dynamic programming. The column generation method has been extensively tested on randomly generated networks with different properties and benchmarked against the dual simplex algorithm. Using interior point methods could not compete with the dual simplex method. On instances with many sources/demands and large arc capacities, column generation turned out to be superior. We observed that initializing column generation with heuristically generated paths and separating the capacity constraints dynamically instead of adding them immediately to the LPs greatly improves the solving time. For the realistic energy models, the heuristic even manages to find optimal solutions directly, allowing us to prove optimality with a single pricing round. This motivates the usage of primal heuristics for large-scale linear programs. We could observe that stabilizing the dual variables with another dual solution indeed reduced the number of generated path variables slightly. However, mispricing rounds lead to an overall increase in the running time using the tested stabilization technique.

Open questions include: How does column generation behave if costs are no longer conservative? To this end, our approach would need to be extended by pricing variables for various types of cycles and bicycles. How do the CG-based approaches compare to the generalized network simplex algorithm (for which no computational results have been published, as far as we know) or other algorithms like the one of Dadush et al. (2024). Can column generation successfully be extended to the network design problem, especially in the context of energy models? Introducing integer (or binary) design variables for choosing capacities leads to a mixed-integer linear problem which could be solved via branch-and-price. Does such a price-and-cut method perform well on interesting networks?

Acknowledgement

This article was supported by the German BMFTR in the context of the RODES project (grant number 05M22RDA). We gratefully acknowledge the contribution of Paul Stursberg from Siemens AG, who provided access to the Power-to-Heat model used in our computational experiments. Further, we thank the anonymous reviewers and the associate editor for their remarks that helped to significantly improve the paper.

References

T. Achterberg. *Constraint Integer Programming*. Dissertation, TU Berlin, 2007.

R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows*. Prentice-Hall, Inc., 1993.

J. Barbosa, C. Ripp, and F. Steinke. Accessible modeling of the german energy transition: An open, compact, and validated model. *Energies*, 14(23), 2021. doi:10.3390/en14238084.

R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

T. Brown, J. Hörsch, and D. Schlachtberger. PyPSA: Python for Power System Analysis. *Journal of Open Research Software*, 6(1), 2018. doi:10.5334/jors.188.

E. Cohen and N. Megiddo. New algorithms for generalized network flows. *Mathematical Programming*, 64:325–336, 1994. doi:10.1007/bf01582579.

D. Dadush, Z. K. Koh, B. Natura, N. Olver, and L. A. Végh. A strongly polynomial algorithm for linear programs with at most two nonzero entries per row or column. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 1561–1572. Association for Computing Machinery, 2024. doi:10.1145/3618260.3649764.

G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

J. Desrosiers and M. E. Lübbecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 1–32. Springer US, Boston, MA, 2005. doi:10.1007/0-387-25486-2_1.

F. Glover. GNETGEN. Technical report, University of Colorado, 1992. URL https://www.netlib.org/lp/generators/gnetgen.

A. V. Goldberg, S. A. Plotkin, and E. Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16(2):351–381, 1991. doi:10.1287/moor.16.2.351.

D. Goldfarb and Y. Lin. Combinatorial interior point methods for generalized network flow problems. *Mathematical Programming*, 93:227–246, 2002. doi:10.1007/s10107-002-0333-y.

S. Hajikazemi and J. Barbosa. Compact energy system modeling tool (cesm), 2024.

C. Hojny, M. Besançon, K. Bestuzheva, S. Borst, A. Chmiela, J. Dionísio, L. Eifler, M. Ghannam, A. Gleixner, A. Göß, A. Hoen, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, S. J. Maher, G. Mexi, E. Mühmer, M. E. Pfetsch, S. Pokutta, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, M. Walter, D. Weninger, and L. Xu. The SCIP Optimization Suite 10.0. Technical report, Optimization Online, November 2025. URL https://optimization-online.org/?p=32699.

M. Howells, H. Rogner, N. Strachan, C. Heaps, H. Huntington, S. Kypreos, A. Hughes, S. Silveira, J. DeCarolis, M. Bazillian, and A. Roehrl. OSeMOSYS: The open source energy modeling system: An introduction to its ethos, structure and development. *Energy Policy*, 39(10):5850–5870, 2011. doi:10.1016/j.enpol.2011.06.033.

W. S. Jewell. New methods in mathematical programming—optimal flow through networks with gains. *Operations Research*, 10(4):476–499, 1962. doi:10.1287/opre.10.4.476.

L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics - Doklady*, 20: 191–194, 1979.

D. Klingman, A. Napier, and J. Stutz. NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20 (5):814–821, 1974. doi:10.1287/mnsc.20.5.814.

B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer, Berlin, Heidelberg, 6th edition, 2018. doi:10.1007/978-3-662-56039-6.

R. Loulou, U. Remne, A. Kanudia, A. Lettila, and G. Goldstein. Documentation for the TIMES model part I, 2005. URL https://iea-etsap.org/index.php/documentation.

M. E. Lübbecke. Column generation. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Ltd, 2011. doi:10.1002/9780470400531.

N. Olver and L. A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. *J. ACM*, 67(2):Article 10, 2020. doi:10.1145/3383454.

J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2): 338–350, 1993. doi:10.1287/opre.41.2.338.

A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. In-out separation and column generation stabilization by dual price smoothing. In *Experimental Algorithms*, pages 354–365. Springer, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-38527-8_31.

M. Shigeno. A survey of combinatorial maximum flow algorithms on a network with gains (network design, control and optimization). *Journal of the Operations Research Society of Japan*, 47:244–264, 01 2004. doi:10.15807/jorsj.47.244.

V. Slednev. *Development of a techno-economic energy system model considering the highly resolved conversion and multimodal transmission of energy carriers on a global scale*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2024.

L. A. Végh. A strongly polynomial algorithm for generalized flow maximization. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 644–653. Association for Computing Machinery, 2014. doi:10.1145/2591796.2591806.

K. D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. *Mathematics of Operations Research*, 27(3):445–459, 2002. doi:10.1287/moor.27.3.445.313.

R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. Dissertation, TU Berlin, 1996. URL `https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/538`.

A Stabilization

We analyze the behavior of two stabilization methods Stab–0 and Stab–π and compare the solving process with the standard price-and-cut method (None). Here, Stab–0 denotes the usage of 0 as the stabilization center and Stab–π the version where the duals of the previous iteration are used. Numerical results are illustrated in Tables 8 and 9. Overall, we can make two main observations using the stabilization scheme. First, we indeed slightly reduce the number of generated paths on all test classes, e.g., nsupp25: None: 7190.4; Stab–0: 7007.4; Stab–π: 7084.8. Unfortunately, this reduced number of generated paths does not lead to faster running times. To the contrary, not only the overall running time increases, e.g., nsupp25: None: 54.31 s; Stab–0: 73.64 s; Stab–π: 70.50 s, but also the percentage of running time spent on the pricing problem, e.g, nsupp25: None: 58.52 %; Stab–0: 69.96 %; Stab–π: 66.75 %. This increase can be explained by the mispricing rounds, which cause a complete run of the dynamic programming algorithm without producing a single useful path variable.

B Advantage of Heuristic and Price-and-Cut

The following tables show the advantage of the initial heuristic and the price-and-cut method on the path formulation (GMCF-P) in contrast to only using column generation. We present average running times using SoPlex, CPLEX and Gurobi as LP solvers for all instances in the testset. We compare the use of only column generation (Default) with additionally using the initial heuristic (H) and the price-and-cut method for the capacity constraints (C). Additionally we activate the heuristic and the separation of capacity constraints together (H+C) showing the best results for all test instances and solvers. Table 10 shows the results for SoPlex, Table 11 for CPLEX and Table 12 shows the results for Gurobi.

C Solving the Arc Formulation

We compare the running times of the dual simplex method, the barrier algorithm and crossover method from Gurobi and CPLEX on all of our test instances. For both solvers, the simplex method is far

Table 8: Stabilization methods for (GMCF-P) on nsupp instances, using Gurobi as LP solver.

Class	Stab	#paths	solsize	LPiter	time	PP%	RMP%
	(None)	7190.4	2754.6	109573.4	54.31	58.52	11.24
nsupp25	(Stab–0)	7007.4	2754.6	106435.8	73.64	69.96	8.11
	(Stab–π)	7084.8	2754.6	127492.8	70.50	66.75	10.15
	(None)	6809.7	2767.8	68260.1	33.19	59.03	9.99
nsupp50	(Stab–0)	6601.4	2767.8	66325.1	45.17	70.51	7.09
	(Stab–π)	6669.1	2767.8	82186.6	42.96	66.86	9.36
	(None)	6150.4	2732.4	39069.4	20.23	58.65	7.58
nsupp75	(Stab–0)	5962.2	2732.4	38466.4	27.81	70.17	5.46
	(Stab–π)	6025.9	2732.4	45532.7	26.05	67.04	6.90
	(None)	5921.9	2726.4	31349.2	18.30	59.82	6.48
nsupp100	(Stab–0)	5751.3	2726.4	30827.3	25.75	71.55	4.58
	(Stab–π)	5793.3	2726.4	35479.3	23.66	68.32	5.77
	(None)	5595.0	2709.7	24236.8	16.22	60.05	5.39
nsupp125	(Stab–0)	5444.6	2709.7	23532.0	22.64	71.65	3.74
	(Stab–π)	5474.5	2709.7	26109.7	20.64	68.44	4.54
	(None)	5295.9	2653.8	18981.3	11.91	58.20	5.03
nsupp150	(Stab–0)	5145.9	2653.8	18584.0	17.00	70.77	3.46
	(Stab–π)	5176.1	2653.8	19849.3	15.50	67.93	4.05

Table 9: Stabilization methods for (GMCF-P) on maxcap instances, using Gurobi as LP solver.

Class	Stab	#paths	solsize	LPiter	time	PP%	RMP%
	(None)	14181.9	4730.4	407647.1	143.69	47.59	28.43
maxcap5	(Stab–0)	13708.4	4730.4	396401.2	181.16	59.53	21.79
	(Stab–π)	13926.6	4730.4	500419.2	189.22	55.66	26.25
	(None)	7306.0	2758.8	115685.9	64.80	58.34	10.48
maxcap15	(Stab–0)	7107.9	2758.8	112899.9	88.50	69.85	7.50
	(Stab–π)	7137.2	2758.8	137708.2	84.39	66.54	9.60
	(None)	4983.9	2012.9	56014.0	40.95	61.54	6.22
maxcap25	(Stab–0)	4880.6	2012.9	55317.5	57.03	72.57	4.36
	(Stab–π)	4925.0	2012.9	67015.2	52.25	68.97	5.75
	(None)	3798.5	1604.4	35150.7	32.34	62.09	4.14
maxcap35	(Stab–0)	3703.2	1604.4	34746.9	44.86	72.90	3.00
	(Stab–π)	3742.1	1604.4	40671.8	40.82	69.48	3.78
	(None)	3070.4	1357.4	23814.9	26.04	61.78	3.16
maxcap45	(Stab–0)	3016.5	1357.4	23422.9	36.25	72.82	2.22
	(Stab–π)	3039.0	1357.4	27319.5	32.93	69.49	2.83
	(None)	2575.0	1174.3	17698.2	20.64	60.89	2.64
maxcap55	(Stab–0)	2529.3	1174.3	17697.8	28.78	72.22	1.90
	(Stab–π)	2544.4	1174.3	19690.3	25.79	68.54	2.33

Table 10: Initial heuristic and separation of capacity constraints, using SoPlex as LP solver.

Class	Default	H	C	H+C
nsupp25	816.49	316.84	298.25	62.10
nsupp50	501.20	194.11	242.97	38.48
nsupp75	276.56	101.24	223.56	23.30
nsupp100	230.11	82.52	221.02	20.71
nsupp125	175.82	61.87	218.15	17.89
nsupp150	128.42	40.01	216.58	13.16
maxcap5	3584.07	1207.86	1038.71	181.28
maxcap15	951.17	344.34	303.50	72.86
maxcap25	462.39	185.70	196.47	44.31
maxcap35	326.56	130.03	164.94	34.11
maxcap45	245.47	94.92	138.40	27.13
maxcap55	177.40	70.78	118.25	21.28

Table 11: Initial heuristic and separation of capacity constraints, using CPLEX as LP solver.

Class	Default	H	C	H+C
nsupp25	1389.24	389.69	271.47	69.14
nsupp50	764.54	230.42	230.55	41.72
nsupp75	311.10	127.96	225.38	24.48
nsupp100	265.37	104.70	224.49	21.23
nsupp125	216.22	81.59	224.04	18.45
nsupp150	159.78	54.17	221.65	13.50
maxcap5	2953.21	1525.99	793.10	265.67
maxcap15	1185.66	447.36	277.51	82.07
maxcap25	1435.58	239.11	188.33	46.30
maxcap35	1066.23	172.53	159.21	34.69
maxcap45	506.93	128.23	140.41	27.31
maxcap55	343.23	94.39	118.02	21.25

Table 12: Initial heuristic and separation of capacity constraints, using Gurobi as LP solver.

Class	Default	H	C	H+C
nsupp25	1141.73	384.42	230.94	54.31
nsupp50	619.10	215.20	204.89	33.19
nsupp75	354.00	115.33	204.36	20.23
nsupp100	281.40	94.24	208.54	18.30
nsupp125	225.87	76.53	209.10	16.22
nsupp150	163.27	47.75	209.57	11.91
maxcap5	3576.42	1309.28	486.38	143.69
maxcap15	1361.72	441.07	235.31	64.80
maxcap25	685.77	242.34	174.42	40.95
maxcap35	537.34	177.13	151.77	32.34
maxcap45	376.93	130.48	135.82	26.04
maxcap55	279.79	95.50	115.61	20.64

Table 13: Running times of simplex and interior point methods solving the arc formulation.

	Simplex		Barrier		Crossover	
Class	CPLEX	Gurobi	CPLEX	Gurobi	CPLEX	Gurobi
nsupp25	38.47	36.17	448.05	825.85	440.97	826.88
nsupp50	33.53	32.91	321.85	648.65	392.16	648.44
nsupp75	30.07	30.77	353.64	319.97	315.85	308.95
nsupp100	28.00	29.55	308.78	242.64	322.54	245.49
nsupp125	26.77	28.41	260.22	218.64	274.33	219.67
nsupp150	25.74	28.00	285.33	220.58	273.93	222.16
maxcap5	47.87	39.92	549.04	1004.83	482.04	1004.56
maxcap15	38.92	36.81	426.54	915.95	381.30	917.87
maxcap25	36.02	35.40	490.99	892.34	503.47	894.53
maxcap35	34.93	35.63	409.09	806.86	423.23	806.70
maxcap45	33.09	34.84	369.46	743.96	383.92	759.41
maxcap55	34.17	35.29	399.33	672.15	412.99	659.19

superior compared to the interior points methods as shown in Table 13.

D Other LP Solvers

We provide average solving times using the three LP solvers SoPlex, CPLEX and Gurobi for the arc and path formulations in Table 14. Like in Section 4, the arc formulation is solved with a simplex algorithm and the path formulation by our price-and-cut method using the dynamic programming scheme to solve the pricing problem. Considering the results for the arc formulation, we can observe that SoPlex is not competitive with the commercial solvers CPLEX and Gurobi when it comes to solving large-scale problems of the form (GMCF-A). The performance of CPLEX and Gurobi seems to be comparable, but CPLEX slightly wins on most instances. Looking at the path formulation, SoPlex is competitive with the commercial solvers, i.e., SoPlex performs slightly better than CPLEX and slightly worse than Gurobi.

Comparing the arc with the path formulation, we can observe that using SoPlex, the price-and-cut method on the path formulation is always superior. For CPLEX and Gurobi, the performance for the two formulations and solving approaches depend on the class characteristics as described in the Section 4.

Table 14: Average solving times in seconds, using different LP solvers.

	(GMCF-A)			(GMCF-P)		
Class	SoPlex	CPLEX	Gurobi	SoPlex	CPLEX	Gurobi
nsupp25	939.34	38.47	36.17	62.10	69.14	54.31
nsupp50	785.68	33.53	32.91	38.48	41.72	33.19
nsupp75	663.92	30.07	30.77	23.30	24.48	20.23
nsupp100	623.52	28.00	29.55	20.71	21.23	18.30
nsupp125	551.11	26.77	28.41	17.89	18.45	16.22
nsupp150	498.07	25.74	28.00	13.16	13.50	11.91
maxcap5	841.11	47.87	39.92	181.28	265.67	143.69
maxcap15	936.26	38.92	36.81	72.86	82.07	64.80
maxcap25	1066.02	36.02	35.40	44.31	46.30	40.95
maxcap35	1016.29	34.93	35.63	34.11	34.69	32.34
maxcap45	916.87	33.09	34.84	27.13	27.31	26.04
maxcap55	935.95	34.17	35.29	21.28	21.25	20.64