Column Generation for Generalized Min-Cost Flows with Losses

Jonas Alker, Marc E. Pfetsch

November 12, 2025

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 on the basis of the linear programming solvers of CPLEX, Gurobi and SoPlex. The column generation approach turns out to be superior to the dual simplex algorithm for instances with a large number of sources/demands and large capacities. Using interior point (barrier) methods is not competitive. Moreover, stabilization of the column generation 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 have efficiencies that allow for losses or gains of the flow. As the name already suggests, it is a generalization of the classical flow problem in which the efficiencies are all 1. The problem has already been considered by Jewell in 1962 [18] and Dantzig in 1963 [10]. The GMCF has a wide range of applications ranging from exchanges of currencies to modeling energy conversion processes, see Section 3. 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, especially when many variant networks have to be considered.

We therefore develop a column generation (CG) approach for a path-based formulation of GMCF with losses. We show that for this case, the pricing problem is a shortest path problem. We then conduct an extensive computational comparison on large-scale instances. The column generation was implemented in C++ using SCIP [5] and we consider the three LP-solvers CPLEX, Gurobi and SoPlex to solve the underlying master problems. We compare this approach to solving the arc-based formulation using the same LP-solvers. We test our approach on generated and realistic energy network instances. The code and generated instances are publicly available.

It turns out that the best method depends on problem parameters like the number of supplies/demands and the range of the capacities. If both numbers are large, CG is significantly faster, making it a method of choice for such cases. The dual simplex is faster if these numbers are small. Using interior point (barrier) methods turns out to be much slower, even without crossover. We also implemented a stabilization method for CG, but this does not seem to be advantageous. However, an initialization of the master problems with heuristically generated paths is very effective. This applies in particular to instances obtained from energy models. We therefore conclude that CG enlarges the available toolbox for such networks and if memory consumption is critical.

This paper is structured as follows. In Section 2, the basic model is introduced and Section 3 gives an overview on the literature. The CG approach is developed in Section 4. The computational results are presented in Section 5. We end the paper with conclusions in Section 6. 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. 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 positive efficiency parameter $\mu_a \in \mathbb{R}_{>0}$ indicating how much of the flow reaches the endnode of the arc. Additionally, 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. The amount of flow traversing an arc a is denoted by the variable x_a. Then the *generalized min-cost flow problem* is:

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

Note that 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 ($\sum_{v \in V} b_v = 0$).

In the literature, generalized min-cost flow problems are modeled with fixed node balances for every node. For (GMCF), we choose to model the flow conservation constraint of input nodes as an inequality instead of an equality. 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 losses/gains. In Section 4.2, we discuss the implications on the solving process of relaxing our assumptions.

Without loss of generality, we assume to have a single source s with positive supply and a single target node t with negative supply. This assumption is indeed no restriction. By introducing artificial nodes s and t and arcs from s to all input nodes and from all output nodes to t any instance of the generalized flow problem can be transformed accordingly. Note that the restricted supply at s can be controlled by the capacities of the added outgoing arcs. (GMCF) can then be written as:

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

Let $\mathcal{P}$ be the set of simple directed paths from s to t. Let $\mathcal{P}_a$ be the subset of paths in P that contain arc $a \in A$. Let μ_P 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 after a certain arc a by:

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

where $e \ge a$ for two arcs e, $a \in P$ holds if e does not appear before a. A path variable should then represent the amount of flow that reaches the target node t via a certain path. In the generalized setting, this is not necessary 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. Therefore, $\mu_{P,a}$ is necessary for computing the amount of flow on certain arcs of a path. The following identity holds for subpaths and can easily be seen:

Lemma 1. *Let $P \in \mathcal{P}$ be an s–v path ending with an arc $\bar{a} = (u, v)$. Then for* *every arc $a \in P$ with $a \neq \bar{a}$:*

$$\mu_{P,a} = \mu_{\bar{a}}\, \mu_{Q,a},$$

where Q is the s–u path P without the last arc $\bar{a}$.

Then we can calculate the amount of flow $x_{P,a}$ on a certain arc a for any path variable x_P as follows:

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

For two adjacent 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:

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

Therefore the path formulation of (GMCF-A) is given by:

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

where we define

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

such that the following holds for the objective:

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

We can show that under certain assumptions the path formulation (GMCF-P) is equivalent to (GMCF-A). To prove this, we need a decomposition of an arc-based solution into paths and a transformation from paths to flow values on arcs. A decomposition of generalized flows is given for example given by [13, Thm. 2.6]. We present a variation of this decomposition adjusted to our problem formulation with no constraint at the source node. Since in generalized flow problems sending flow along a directed cycle does not necessarily conserve the flow, we define three different types of cycles. For a directed cycle C define the *efficiency* of the cycle $\mu_C \coloneqq \prod_{a \in C} \mu_a$. If $\mu_C = 1$, the cycle is called *flow conserving*. If $\mu_C > 1$ (resp. $\mu_C < 1$) we call the cycle *flow generating* (resp. *flow consuming*).

Lemma 2 (Generalized Flow Decomposition [13, Thm. 2.6])**.** *Let x be a generalized flow feasible for* (GMCF-A) *with $b_t < 0$. Then x can be decomposed into $k \leq |A|$ flows*

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

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

Proof. The proof for this lemma can be found in [13]. In our setting, t is a node with a deficit ($b_t < 0$) and we have no constraint on the source node s. Considering the sign of the imbalance b_s of x at node s

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

the source node is either a node with a deficit, excess or a node where flow conservation holds. If $b_s > 0$, we have a generalized flow with a single excess node and a single deficit node. Then [13, Thm. 2.6] gave a flow decomposition into types (a), (b), (d), (e) and (f). For $b_s < 0$, we obtain a decomposition into flows of types (a), (b), (c), (e) and (f) and for $b_s = 0$ we get flows of types (a), (b), (e) and (f). □

To show that the formulations (GMCF-A) and (GMCF-P) are indeed equivalent, we first observe that under certain conditions the arc formulation has an optimal solution without cycles.

Lemma 3. *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 contains no cycles.*

Proof. Since the problem is feasible and bounded, (GMCF-A) has an optimal solution x. By Proposition 2, x can be decomposed into types (a)–(f). The assumption $\mu_C \leq 1$ assures that no elementary flows of (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) from the flow x, it stays feasible and objective value does not increase. □

Theorem 4. *Let $\mu_C = \prod_{a \in C} \mu_a \leq 1$ for all cycles C and $c_a \geq 0$ for all $a \in A$. Then the problems* (GMCF-A) *and* (GMCF-P) *are equivalent.*

Proof. By Lemma 3, if (GMCF-A) is feasible, there exists a cycle free solution that can be decomposed into paths. It can easily be shown that those paths form a solution for (GMCF-P) with the same objective value.

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

$$x_a = \sum_{P \in \mathcal{P}_a} \mu_{P,a}^{-1} y_P \quad \forall a \in A,$$

where P_a is the set of paths including a. It can easily be shown that x is feasible for (GMCF-A) and has the same objective as y. □

3 Literature Review

The generalized min-cost flow problem has a long history and has been tackled in various formulations. We give an overview in the following.

In 1962, Jewell presented a primal-dual algorithm for the generalized problem with capacities [18]. 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 presented an extension of the network simplex algorithm to handle the generalized formulation [10]. Ahuja et al. [2] discuss the efficient implementation 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 [25]. The algorithm solves the uncapacitated version with fixed node balances at every node. Solving the version with a capacity constraints can be solved in strong polynomial time as well, since the capacitated version can be transformed to an uncapacitated formulation on a bipartite graph [21, 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 [19]. Wayne [30] presented a strongly polynomial combinatorial algorithm for the capacitated generalized circulation problem where all node balances are zero, extending the algorithms presented by Cohen and Megiddo in [7]. A similar, but slightly different algorithm was given by Goldfarb and Lin [14]. Their presented combinatorial interior point method has the same worst case running time like the combinatorial algorithm by Wayne. In 2013 Végh [29] 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 [27]. 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 [29]. The strongly polynomial algorithm for this problem uses a scaling method and contracting arcs. In 2020 an improved version of this algorithm in was presented in [24]. The question of strongly polynomial solvability of the generalized min-cost flow problem was resolved by Dadush et al [9], based 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 plays a central role such as the well-known PyPSA [6], OSeMOSYS [17] and TIMES [22] or the more recent models OCGModel [3] and PERSEUS-gECT [28]. All of these modeling tools rely on general purpose (mixed-integer) linear programming solving techniques to solve the models.

4 Column Generation for the Path Formulation

The formulation (GMCF-P) has $|P|$ 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, we refer [11, 23]. The basic idea is to solve the *(restricted) master problem*, a restricted version of (GMCF-P) using a subset of variables (paths). Using the so-called *pricing problem*, one can compute the reduced costs of variables not yet present. If the reduced costs are negative, the corresponding variables are added to the master problem. Otherwise, the problem is solved optimally. In the following, we show how 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 are met.

The dual problem of (GMCF-P) reads:

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

Let $x^\star$ be a feasible solution for the restricted master problem and $(\lambda^\star, \pi^\star)$ be the corresponding dual solution. Then $x^\star$ is optimal for (GMCF-P) if $(\lambda^\star, \pi^\star)$

is feasible for (D-GMCF-P). 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 master problem.

Let ν(GMCF-P) be the optimal value of (GMCF-P). We can obtain a lower bound on ν(GMCF-P) based on the solution value of the pricing problem. Therefore, define $\nu(\text{PP}) \coloneqq \min_{P \in \mathcal{P}} \sum_{a \in P} \mu_{P,a}^{-1} \, (\pi_a^\star + c_a) - \lambda^\star$ be the value of the pricing problem and $x^\star$ the current solution of the restricted problem. If $\nu(\text{PP}) < 0$, there exists a path with negative reduced cost, so we cannot terminate yet. But we can conclude a lower bound on the optimal value of (GMCF-P), by showing that $(\lambda^\star + \nu(\text{PP}), \pi^\star)$ is a feasible dual solution:

$$\begin{aligned}
&\lambda^\star - \sum_{a \in P} \mu_{P,a}^{-1} \, \pi_a^\star + \nu(\text{PP}) \\
&= \lambda^\star - \sum_{a \in P} \mu_{P,a}^{-1} \, \pi_a^\star + \min_{Q \in \mathcal{P}} \sum_{a \in Q} \mu_{Q,a}^{-1} \, (\pi_a^\star + c_a) - \lambda^\star \\
&\leq - \sum_{a \in P} \mu_{P,a}^{-1} \, \pi_a^\star + \sum_{a \in P} \mu_{P,a}^{-1} \, (\pi_a^\star + c_a) = c_P
\end{aligned}$$

for all $P \in \mathcal{P}$. The optimal value of (GMCF-P) can be bounded by:

$$\nu(\text{GMCF-P}) \geq -b_t \, (\lambda^\star + \nu(\text{PP})) - \sum_{a \in A} \pi_a^\star \, u_a = c^\top x^\star - b_t \, \nu(\text{PP}).$$

4.1 Generalized Shortest Paths

To solve the pricing problem (PP), we develop a dynamic programming algorithm [4] for finding a generalized shortest path. 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 is

$$\tilde{c}(W) \coloneqq \sum_{a \in W} \mu_{W,a}^{-1} \, \alpha_a. \tag{2}$$

When solving the pricing problem (PP) we define $\alpha_a \coloneqq \pi_a + c_a$. Then, the assumption $c_a \geq 0$ and $\pi_a \geq 0$ imply $\alpha_a \geq 0$ for all a. It is then sufficient to check whether $\tilde{c}(P) < \lambda$ holds for the shortest path w.r.t. $\tilde{c}(P)$ to solve the pricing problem. Note that the costs $\tilde{c}(P)$ depend on the order of the path and thus classical shortest path algorithms do not work directly.

Lemma 5. *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. The Lemma is trivial if either W_1 or W_2 is empty. Assume both walks are not empty. Let $W := W_1 + W_2$. We can write W as a sequence of arcs $(a_1, \ldots, a_k)$ where the first $r < k$ arcs belong to the walk W_1 and the remaining $k - r$ arcs belong to W_2. We can extract the cost of the last arc a_k from $\tilde{c}(W)$:

$$\begin{aligned}\tilde{c}(W) &= \sum_{a \in W} \mu_{W,a}^{-1}\,\alpha_a = \sum_{i=1}^{k}\prod_{j=i}^{k}\mu_{a_j}^{-1}\,\alpha_{a_i} = \mu_{a_k}^{-1}\left(\sum_{i=1}^{k}\prod_{j=i}^{k-1}\mu_{a_j}^{-1}\,\alpha_{a_i}\right)\\ &= \mu_{a_k}^{-1}\,\alpha_{a_k} + \mu_{a_k}^{-1}\left(\sum_{i=1}^{k-1}\prod_{j=i}^{k-1}\mu_{a_j}^{-1}\,\alpha_{a_i}\right).\end{aligned}$$

Repeating the above procedure $k - r$ times yields:

$$\begin{aligned}\tilde{c}(W) &= \sum_{i=r+1}^{k}\prod_{j=i}^{k}\mu_{a_j}^{-1}\,\alpha_{a_i} + \prod_{i=r+1}^{k}\mu_{a_i}\left(\sum_{i=1}^{r}\prod_{j=i}^{r}\mu_{a_j}^{-1}\,\alpha_{a_i}\right)\\ &= \tilde{c}(W_2) + \mu_{W_2}^{-1}\,\tilde{c}(W_1).\end{aligned}$$

□

Lemma 5 allows us to show that the assumptions on the input data implies that the cost function $\tilde{c}(W)$ is *conservative*, i.e., extending a walk W by a cycle C does not decrease the overall cost.

Lemma 6. *Let $\mu_C \leq 1$ for all cycles C and $\alpha_a \geq 0$ for all $a \in A$. Then the cost function $\tilde{c}(p)$ 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 5 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 5 and Lemma 6 give us the ability to formulate a result concerning optimality of subpaths.

Lemma 7. *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 $Q := P \setminus \{\bar{a}\}$ is a shortest s–u path with at most $k - 1$ arcs.*

Proof. Assume there exists an s–u path Q' having at most $k - 1$ arcs with $\tilde{c}(Q') < \tilde{c}(Q)$. Let P' be the s–u path Q' followed by the arc $\bar{a}$. By Lemma 5:

$$\tilde{c}(P') = \mu_{\bar{a}}^{-1}\,\alpha_{\bar{a}} + \mu_{\bar{a}}^{-1}\,\tilde{c}(Q') < \mu_{\bar{a}}^{-1}\,\alpha_{\bar{a}} + \mu_{\bar{a}}^{-1}\,\tilde{c}(Q) = \tilde{c}(P). \tag{3}$$

If Q' does does not visit v, P' is a shorter s–v path than P.

Otherwise, let $Q'_{[s,v]}$ and $Q'_{[v,u]}$ be the subpaths of Q' split at v such that $Q' = Q'_{[s,v]} + Q'_{[v,u]}$. Note that $C \coloneqq Q'_{[v,u]} + \bar{a}$ gives a cycle at node v. We now apply Lemma 5 twice in a similar way:

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

Using $\tilde{c}(C) = \alpha_{\bar{a}} + \mu_{\bar{a}}^{-1}\, \tilde{c}(Q'_{[v,u]})$ and the conservativeness of the cost function, see Lemma 6, we obtain:

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

Either way, we found an s–v path P' with k arcs of smaller cost than P, which contradicts the optimality of P. □

Motivated by Lemma 7, we compute a shortest path with respect to $\tilde{c}(P) = \sum_{a \in P} \mu_{P,a}^{-1}\, \alpha_a$ 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 $n \coloneqq |V|$. 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 $(V, A)$, $\mu_a$, $\alpha_a$ for all $a \in A$, start and end nodes $s$ and $t$.
   Output: The minimum cost of an $s$–$t$ path according to the pricing
           costs $\tilde{c}(p) = \sum_{a \in p} \mu_{p,a}^{-1}\, \alpha_a$ and $\mathrm{pr}(v)$ for all $v \in V$ representing
           the predecessor of $v$ in an optimal $s$–$t$ path.
1  $C_v^k \leftarrow \infty$ for all $v \neq s$ and all $k = 0, \dots, n-1$;
2  $C_s^k \leftarrow 0$ for all $k = 0, \dots, n-1$;
3  for $k = 1, \dots, n-1$ do
4      $C_v^k \leftarrow C_v^{k-1}$ for all $v \in V$;
5      for $a = (u, v) \in A$ do
6          if $C_v^k > \mu_a^{-1}\big(\alpha_a + C_u^{k-1}\big)$ then
7              $C_v^k \leftarrow \mu_a^{-1}\big(\alpha_a + C_u^{k-1}\big)$;
8              $\mathrm{pr}(v) \leftarrow u$;
9  return $C_t^{n-1}$, $\mathrm{pr}(v)$ for all $v \in V$;
```

Theorem 8. *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 $m = |A|$.*

Proof. The proof works exactly as proving correctness of the standard Bellman-Ford algorithm with conservative costs, see for example [21]. The formula for calculating the path-cost has been established in Lemma 5 and conservative costs are given due to Lemma 6. The running time is obvious. □

4.2 Alternative Formulations

In this section we discuss the influence of our assumptions on the pricing problem. Motivated by the application of energy systems we assume to have no flow generating cycles, no arcs with negative cost and no constraint at the source node s. An equivalent formulation would be given by the following: Assume the graph contains no flow consuming cycles, i.e., $\mu_C \geq 1$ for all cycles C, the arc costs are nonnegative, the node balance at the source node s is fixed to $b_s > 0$ and no constraint at the target node t is given. By defining path variables by the amount of flow leaving s, we obtain the path formulation:

$$\begin{aligned} \min_x \quad & \sum_{P \in \mathcal{P}} c_P\, x_P, \\ \text{s.t.} \quad & \sum_{P \in \mathcal{P}} x_P = b_s, \\ & \sum_{P \in \mathcal{P}_a} \mu_{P,a}\, x_P \leq u_a \qquad \forall a \in A, \\ & x_P \geq 0 \qquad\qquad\quad\;\; \forall P \in \mathcal{P}, \end{aligned} \tag{GMCF-P-s}$$

with $c_p := \sum_{a \in P} \mu_{P,a}\, c_a$ and $\mu_{P,a} := \prod\{\mu_e : e \in P, e < a\}$. The pricing problem for a dual solution $(\lambda^\star, \pi^\star)$ then reads:

$$\text{Find } P \in \mathcal{P} \text{ with } \tilde{c}(P) := \sum_{a \in P} \mu_{P,a}\, (\pi^\star_a + c_a) < \lambda^\star. \tag{PP-s}$$

Since $\mu_C \geq 1$ for all cycles C and $c_a \geq 0$ for $a \in A$ the cost function $\tilde{c}(P)$ is conservative and we can apply a similar dynamic programming scheme (starting at t and extending paths towards s).

Assume we add a constraint to the path formulation fixing the node balance at s to $b_s > 0$ additionally to the demand constraint at t, i.e., we add to (GMCF-P) the constraint:

$$\sum_{P \in \mathcal{P}} \mu_P^{-1}\, x_P = b_s. \tag{4}$$

Let η be the corresponding dual variable and $(\lambda^\star, \eta^\star, \pi^\star)$ a dual solution. The pricing problem changes to:

$$\text{Find } P \in \mathcal{P} \text{ with } \tilde{c}(P) := \sum_{a \in P} \mu_{P,a}^{-1}\, (\pi^\star_a + c_a) - \mu_P^{-1}\, \eta^\star < \lambda^\star. \tag{PP-st}$$

This is already significantly harder to solve than (PP), since the cost function $\tilde{c}(P)$ is no longer conservative. Additionally, (GMCF-P) together with (4) is not equivalent to the corresponding arc formulation, where the constraint

$$\sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} \mu_a x_a = b_s \tag{5}$$

for $b_s > 0$ is added to (GMCF-A), which is equivalent to formulating the inequalities at the input in (GMCF) as equality constraints. By the flow decomposition of Lemma 2, the path formulation needs variables for s–t paths and flows on combinations of flow consuming cycles and connecting paths from s. Therefore, it is not sufficient to only generate path variables in the pricing problem, but also flow consuming cycles connected to s, if the graph is not acyclic. In the context of energy systems, this corresponds to "burning" the surplus of energy in the energy system the cheapest possible way. Thus, we use the model without constraint on the source node, since in practice reducing the import seems to be preferable over wasting the excess.

We could also relax the assumptions on efficiencies and costs even further, i.e., only ask for $\mu_a > 0$. The formulation

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

with $b_s > 0$, $b_t < 0$ and $b_v = 0$ for all $v \in V \setminus \{s, t\}$ can also be reformulated in a path-like formulation. Here variables for all types of flows described in Lemma 2 (a)–(f) are needed. The pricing problem is then to find such a flow with negative reduced costs.

5 Implementation and Numerical Results

We implemented the above described dynamic programming algorithm in C++ as a pricing method within the SCIP Framework [5], available at `scipopt.org`. For all numerical experiments we use a prerelease version of SCIP 9.2.4. SCIP allows the implementation of a custom pricer which is then included in the branch-and-price loop. Note that we only solve LPs, no branching is performed. We will investigate the performance of solving the restricted master problems of our column generation approach for (GMCF-P) using different LP solvers, including SoPlex 7.1.3 [31, 5], available at `soplex.zib.de`, CPLEX 12.10.0.0 [8] and Gurobi 11.0.2 [15]. Of course, we can also solve the arc formulation (GMCF-A) with any of these LP-solvers. 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`.

Table 1: Generated instances with different numbers of supply and demand nodes.

Class	#Inst.	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert S\rvert$	$\lvert D\rvert$	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]

Table 2: Generated instances with different capacity ranges.

Class	#Inst.	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert S\rvert$	$\lvert D\rvert$	Capacity
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]

5.1 Test Instances

We test our implementation on various random instances generated by the GNETGEN [12] graph generator by Glover, a modification of NETGEN [20] by Klingman, Napier, and Stutz. We use this generator with only slight modifications (increasing 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 Tables 1 and 2. For every instance class (Class), we show the number of instances in the class (#Inst.), the number of nodes ($|N|$) 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). For this sake, we define $S \coloneqq \{v \in V \mid b_v > 0\}$ and $D \coloneqq \{v \in V \mid b_v < 0\}$. As mentioned in the introduction, we can always transform instances with multiple supply or demand nodes into 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 $a \in A$.

5.2 Primal Feasibility and Price-and-Cut

In the description of the column generation method above, we assumed that the current set of paths are feasible. However, especially when starting the solving process with an empty set of paths, the restricted master problem may be infeasible. One well-known possibility of achieving feasibility is to 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 or may lead to numerical problems, we apply the approach of *Farkas pricing* [1].

The Farkas pricing problem is as the standard pricing problem, but the variable costs are set to 0, i.e., we use $c_a = 0$ and thus $\alpha_a = \pi_a$, to generate new paths. Setting the cost to 0 corresponds to using a dual ray, which proves primal infeasibility, as the dual variables in the pricing problem. A new path cuts the dual ray off, i.e., destroys the primal infeasibility proof.

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) 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. Even if this heuristic does not find a feasible solution, we obtain a set of initial paths, which hopefully helps reducing the number of Farkas pricing iterations.

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 LP in a price-and-cut loop. The first LP of (GMCF-P) uses the heuristically generated paths and the capacity constraints of the arcs that are active for these path variables. After the relaxed LP 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 LP. This process is repeated until the solution for the relaxed problem is feasible for the master problem (GMCF-P) is optimal. 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. The average sizes over all instances of the last LP during the price-and-cut loop is displayed in Tables 3 and 4. For the experiments we activate the initial paths heuristic and the separation of capacity constraints. Recall that the LP of (GMCF-A) has $|A| = 1\,000\,000$ columns and $|N| - 1 = 19\,999$ rows. The linear programs solved by the path formulation are in comparison very small. If the available memory is tight the

Table 3: Average size of last LP for nsupp instances.

	SoPlex		CPLEX		Gurobi	
Class	#cols	#rows	#cols	#rows	#cols	#rows
nsupp25	6366.8	3613.1	6369.1	3613.1	6376.1	3613.1
nsupp50	6174.8	3573.6	6197.9	3574.8	6211.2	3573.7
nsupp75	5522.4	3480.7	5551.5	3479.9	5552.5	3479.9
nsupp100	5343.0	3448.9	5308.4	3448.7	5313.5	3448.8
nsupp125	5057.4	3398.9	5073.0	3399.0	5074.8	3399.1
nsupp150	4820.0	3335.9	4829.3	3335.9	4832.3	3335.9

Table 4: Average size of last LP for maxcap instances.

	SoPlex		CPLEX		Gurobi	
Class	#cols	#rows	#cols	#rows	#cols	#rows
maxcap5	7368.2	5202.6	6824.0	5202.6	8532.0	5219.9
maxcap15	6352.3	3583.1	6140.7	3566.0	6392.9	3614.3
maxcap25	4640.9	2704.6	4641.5	2704.4	4536.7	2704.5
maxcap35	3610.1	2173.6	3608.9	2173.6	3609.5	2173.6
maxcap45	2907.2	1839.5	2905.6	1839.5	2907.2	1839.5
maxcap55	2422.3	1585.3	2421.6	1585.3	2422.3	1585.3

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 a 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.

We use both additions (initial heuristic and separation of capacity constraints) for all test runs when using the path formulation. We can observe that both methods significantly improve the running time for the path formulation. We provide empirical evidence for this in the appendix, see Section A.1.

5.3 Numerical Results

We compare the CPU times of solving the arc formulation (GMCF-A) with using our column generation approach on the path formulation (GMCF-P). We solve every instance to optimality. For the arc formulation we always use the dual simplex method. CPLEX and Gurobi are capable of applying 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) can be found in the appendix, see Section A.2.

Tables 5 and 6 show the average solving times for instances of a certain class using the three different LP solvers SoPlex, CPLEX and Gurobi and for

Table 5: Average solving time for nsupp instances.

	(GMCF-A)			(GMCF-P)		
Class	SoPlex	CPLEX	Gurobi	SoPlex	CPLEX	Gurobi
nsupp25	909.16	36.52	36.69	63.36	70.44	56.15
nsupp50	792.00	31.60	32.51	39.10	42.59	34.14
nsupp75	659.12	28.21	31.16	23.33	24.85	20.90
nsupp100	586.27	26.88	29.73	20.92	21.72	18.93
nsupp125	528.71	24.92	28.75	18.05	18.70	16.74
nsupp150	481.99	24.17	27.96	13.27	13.80	12.35

Table 6: Average solving time for maxcap instances.

	(GMCF-A)			(GMCF-P)		
Class	SoPlex	CPLEX	Gurobi	SoPlex	CPLEX	Gurobi
maxcap5	770.69	45.96	40.39	185.77	257.34	147.66
maxcap15	895.62	37.06	36.58	74.18	84.37	67.19
maxcap25	1061.98	34.09	35.60	45.20	47.59	42.38
maxcap35	960.55	32.61	35.80	34.94	35.69	33.41
maxcap45	921.24	31.59	36.24	27.77	28.02	26.84
maxcap55	887.30	31.86	36.33	21.79	21.86	21.19

the arc and path formulations. 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. 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. In general, increasing the number of supply/demand nodes seems to make the problems easier and the same holds for increasing the maximal capacity.

Looking at the path formulation, this observation can be confirmed. The solving times drastically decrease for all LP solvers when increasing the number of supplies/demands or the maximal capacity. Here, SoPlex is competitive with the commercial solvers, i.e., SoPlex performs slightly better than CPLEX and slightly worse than Gurobi. A more detailed analysis of the solving behavior for the price-and-cut method using Gurobi as an LP solver can be seen in Tables 7 and 8. They provide the number of paths generated (#paths), the number of used paths in the computed optimal solution (solsize), the overall number of iterations the LP solver takes (LPiter), the solving time in seconds (time) and the percentage of the running time spent on the initial heuristic (Heur%), the pricing problem (PP%) and the LP (LP%). Again, all values are averaged over all instances in a certain instance class. The remainder of the running time

Table 7: Analysis of price-and-cut using Gurobi on nsupp instances.

Class	#paths	solsize	LPiter	time	heur%	PP%	LP%
nsupp25	7189.5	2754.6	112884.6	56.15	27.32	57.94	11.23
nsupp50	6808.7	2767.8	71201.8	34.14	27.00	58.35	10.22
nsupp75	6149.2	2732.4	41869.4	20.90	28.10	57.98	8.04
nsupp100	5922.9	2726.4	34272.2	18.93	27.52	58.86	7.20
nsupp125	5594.8	2709.7	26651.4	16.74	27.87	59.36	5.94
nsupp150	5294.6	2653.8	21113.2	12.35	28.14	57.47	5.84

Table 8: Analysis of price-and-cut using Gurobi on maxcap instances.

Class	#paths	solsize	LPiter	time	heur%	PP%	LP%
maxcap5	14195.0	4730.4	411660.6	147.66	21.13	46.88	27.81
maxcap15	7304.7	2758.8	119560.2	67.19	28.55	57.60	10.55
maxcap25	4983.9	2012.9	59222.1	42.38	29.26	60.75	6.38
maxcap35	3798.5	1604.4	36216.5	33.41	30.42	61.52	4.21
maxcap45	3070.4	1357.4	24929.7	26.84	31.17	61.25	3.25
maxcap55	2575.0	1174.3	18589.4	21.19	31.85	60.57	2.71

mainly goes into checking and separating the linear capacity constraints.

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 two formulations and solving approaches are more comparable. Let us first focus on the nsupp instance classes (Table 5). Having a very small number of sources and sinks (e.g., nsupp25) the simplex method on the arc formulation seems to be superior (CPLEX: 36.52 s vs. 70.44 s; Gurobi: 36.69 s vs. 56.15 s). In Table 7 we can see that more than half of the running time (57.97 %) 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 7189.5 paths. On instances with a higher number of supply/demand nodes, e.g., nsupp150, the price-and-cut method is far more competitive (CPLEX: 24.17 s vs. 13.80 s; Gurobi: 27.96 s vs. 12.35 s). Still, the pricing problem takes more than half of the running time (57.47 %), but this time we generate far less unnecessary paths (#paths: 5294.6, 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.

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. 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',

'#paths' and 'time' in Table 8. With higher maximal capacity, less paths are needed for optimal solutions (maxcap5: 4730.4; maxcap55: 1174.3) leading to less generated paths, e.g., 14 195.0 for maxcap5 and 2575.0 for maxcap55. This results in a shorter running time (maxcap5: 147.66 s; maxcap55: 21.19 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 CPLEX and Gurobi (CPLEX: 45.96 s vs. 257.34 s; Gurobi: 40.39 s vs. 147.66 s). Increasing the maximal capacity shifts this behavior such that at a capacity range of [5, 35] (maxcap35), the running times are comparable (CPLEX: 32.61 s vs. 35.69 s; Gurobi: 35.80 s vs. 33.41 s). Increasing the maximal capacity even further, we can observe that the path formulation is now superior (CPLEX: 31.86 s vs. 21.86 s; Gurobi: 36.33 s vs. 21.19 s). For these instances, especially the relative time for solving the linear programs decreases: 27.81 % for maxcap5, 2.71 % for maxcap55.

5.4 Application on Energy Systems Model

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 column generation 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 demanded 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. The models size grows when increasing 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. Consecutive time steps are coupled via storages for electricity and hot water.

The second problem is based on the CESM tool [16], which provides a modeling tool of the German energy supply and demand and is an extension to the OCGModel modeling tool [3]. 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 different at time steps. The problem size depends on the number of time steps within the time horizon.

The results for the two models are presented in Table 9. For the arc formulation, we use CPLEX for solving the LPs and for the column generation we

Table 9: Solving times for generalized min-cost flow problem of energy models.

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	29.64	0.15
CESM	365 000	1 058 450	18 250	109 500	11.58	1.02

use Gurobi, since these choices proved to be superior previously. 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`.

The arc formulation cannot compete with the path formulation for these instances. We can observe that the initial heuristic already provides the optimal solution for both instances. The heuristically computed solution is then proven to be optimal with a single run of the pricing algorithm. This is not necessarily an indicator for the path formulation to be superior to the arc formulation for large scale energy models. However, for the given practical examples, this shows the importance of primal heuristics. Our path based heuristic has the advantage that it not only provides already a primal bound, but also a feasible basis, since we always maximize flow values on generated paths. This can be used to warm start the solution of the inital LP.

5.5 Stabilization

We also investigate the effect of stabilization on the column generation process and consider the in-out separation stabilization technique introduced by Pessoa et al. [26]. 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 currently dual bound, to stabilize the dual variables. As described in [26], using modified dual variables may lead to *mispricing*, 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 in [26] for the next pricing round. When α is sufficiently close to 1.0, we turn off stabilization completely, since we expect to be sufficiently close to finding an optimal solution or proving that the current solution is already optimal. The hope is to reduce the number of overall generated paths by stabilizing the dual variables.

We analyze the behavior of two stabilization methods Stab–0 and Stab–

Table 10: Stabilization methods for (GMCF-P) using Gurobi on nsupp instances.

Class	Stab	#paths	solsize	LPiter	time	PP%	LP%
	None	7189.5	2754.6	112884.6	56.15	57.94	11.23
nsupp25	Stab–0	7006.7	2754.6	110158.8	76.68	69.35	8.08
	Stab–π	7084.7	2754.6	130515.1	72.78	66.29	10.02
	None	6808.7	2767.8	71201.8	34.14	58.35	10.22
nsupp50	Stab–0	6602.4	2767.8	69522.7	47.09	70.03	7.16
	Stab–π	6671.4	2767.8	85246.4	44.38	66.28	9.39
	None	6149.2	2732.4	41869.4	20.90	57.98	8.04
nsupp75	Stab–0	5962.8	2732.4	41284.5	28.98	69.47	5.73
	Stab–π	6025.8	2732.4	48409.5	26.95	66.36	7.29
	None	5922.9	2726.4	34272.2	18.93	58.86	7.20
nsupp100	Stab–0	5751.3	2726.4	33707.2	26.81	70.93	4.98
	Stab–π	5792.7	2726.4	38066.7	24.59	67.66	6.12
	None	5594.8	2709.7	26651.4	16.74	59.36	5.94
nsupp125	Stab–0	5444.3	2709.7	25995.8	23.53	71.12	4.11
	Stab–π	5473.0	2709.7	28621.6	21.46	67.79	4.97
	None	5294.6	2653.8	21113.2	12.35	57.47	5.84
nsupp150	Stab–0	5145.7	2653.8	20903.0	17.62	70.22	4.05
	Stab–π	5176.1	2653.8	22142.6	16.09	67.14	4.68

π and compare the solving process with the standard price-and-cut method (None). Here Stab–0 denotes the 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 10 and 11. 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: 7189.5; Stab–0: 7006.7; Stab–π: 7084.7. 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: 56.15 s; Stab–0: 76.68 s; Stab–π: 72.78 s, but also the percentage of running time spent on the pricing problem, e.g, nsupp25: None: 57.94 %; Stab–0: 69.35 %; Stab–π: 66.29 %. 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.

6 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:

Table 11: Stabilization methods for (GMCF-P) using Gurobi on maxcap instances.

Class	Stab	#paths	solsize	LPiter	time	PP%	LP%
	None	14195.0	4730.4	411660.6	147.66	46.88	27.81
maxcap5	Stab–0	13707.1	4730.4	402369.7	188.15	58.92	21.26
	Stab–π	13933.1	4730.4	503292.6	195.77	55.28	25.51
	None	7304.7	2758.8	119560.2	67.19	57.60	10.55
maxcap15	Stab–0	7107.4	2758.8	116523.3	92.22	69.28	7.49
	Stab–π	7135.9	2758.8	140512.0	87.32	66.02	9.51
	None	4983.9	2012.9	59222.1	42.38	60.75	6.38
maxcap25	Stab–0	4880.5	2012.9	58524.9	59.20	72.01	4.47
	Stab–π	4925.2	2012.9	69811.8	54.01	68.40	5.79
	None	3798.5	1604.4	36216.5	33.41	61.52	4.21
maxcap35	Stab–0	3703.2	1604.4	35671.5	46.62	72.53	2.97
	Stab–π	3742.1	1604.4	41628.4	42.03	69.13	3.77
	None	3070.4	1357.4	24929.7	26.84	61.25	3.25
maxcap45	Stab–0	3016.5	1357.4	24405.9	37.69	72.34	2.24
	Stab–π	3039.0	1357.4	28273.7	33.92	69.16	2.83
	None	2575.0	1174.3	18589.4	21.19	60.57	2.71
maxcap55	Stab–0	2529.1	1174.3	18663.0	29.82	71.81	1.94
	Stab–π	2544.4	1174.3	20547.3	26.60	68.29	2.36

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 $O(nm)$ 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 of different LP solvers. 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. The computational results from of paper motivate the further investigation of column generation based approaches for generalized flow related problems.

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. 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 problems 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.

References

[1] T. Achterberg. "Constraint Integer Programming." Dissertation. TU Berlin, 2007. DOI: 10.14279/depositonce-1634.

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

[3] J. Barbosa, C. Ripp, and F. Steinke. "Accessible Modeling of the German Energy Transition: An Open, Compact, and Validated Model." In: *Energies* 14.23 (2021). DOI: 10.3390/en14238084.

[4] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[5] S. Bolusani, M. Besançon, K. Bestuzheva, A. Chmiela, J. Dionísio, T. Donkiewicz, J. van Doornmalen, L. Eifler, M. Ghannam, A. Gleixner, C. Graczyk, K. Halbig, I. Hedtke, A. Hoen, C. Hojny, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, J. Manns, G. Mexi, E. Mühmer, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, D. Weninger, and L. Xu. *The SCIP Optimization Suite 9.0*. Technical Report. Optimization Online, 2024. URL: https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

[6] T. Brown, J. Hörsch, and D. Schlachtberger. "PyPSA: Python for Power System Analysis." In: *Journal of Open Research Software* 6.1 (2018). DOI: 10.5334/jors.188.

[7] E. Cohen and N. Megiddo. "New algorithms for generalized network flows." In: *Mathematical Programming* 64 (1994), pp. 325–336. DOI: 10.1007/bf01582579.

[8] CPLEX, IBM ILOG. *V12. 1: User's Manual for CPLEX*. 2009.

[9] 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. Association for Computing Machinery, 2024, pp. 1561–1572. DOI: 10.1145/3618260.3649764.

[10] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963. Chap. The Weighted Distribution Problem, pp. 413–432.

[11] J. Desrosiers and M. E. Lübbecke. "A Primer in Column Generation." In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. Boston, MA: Springer US, 2005, pp. 1–32. DOI: 10.1007/0-387-25486-2_1.

[12] F. Glover. *GNETGEN*. Tech. rep. University of Colorado, 1992. URL: https://www.netlib.org/lp/generators/gnetgen.

[13] A. Goldberg, S. Plotkin, and É. Tardos. "Combinatorial algorithms for the generalized circulation problem." In: *Math. Oper. Res.* 16 (1988), pp. 432–443. DOI: 10.1109/SFCS.1988.21959.

[14] D. Goldfarb and Y. Lin. "Combinatorial interior point methods for generalized network flow problems." In: *Mathematical Programming* 93 (2002), pp. 227–246. DOI: 10.1007/s10107-002-0333-y.

[15] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2024. URL: https://www.gurobi.com.

[16] S. Hajikazemi and J. Barbosa. *Compact Energy System Modeling Tool (CESM)*. Version v0.0.9. 2024. DOI: 10.5281/zenodo.13902515.

[17] 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." In: *Energy Policy* 39.10 (2011), pp. 5850–5870. DOI: https://doi.org/10.1016/j.enpol.2011.06.033.

[18] W. S. Jewell. "New Methods in Mathematical Programming—Optimal Flow Through Networks with Gains." In: *Operations Research* 10.4 (1962), pp. 476–499. DOI: 10.1287/opre.10.4.476.

[19] L. G. Khachiyan. "A Polynomial Algorithm in Linear Programming." In: *Soviet Mathematics - Doklady* 20 (1979), pp. 191–194.

[20] D. Klingman, A. Napier, and J. Stutz. "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems." In: *Management Science* 20.5 (1974), pp. 814–821.

[21] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer, 2006. DOI: 10.1007/978-3-662-56039-6.

[22] 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.

[23] M. E. Lübbecke. "Column Generation." In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Ltd, 2011. DOI: 10.1002/9780470400531.

[24] N. Olver and L. A. Végh. "A Simpler and Faster Strongly Polynomial Algorithm for Generalized Flow Maximization." In: *J. ACM* 67.2 (2020), Article 10. DOI: 10.1145/3383454.

[25] J. Orlin. "A Faster Strongly Polynomial Minimum Cost Flow Algorithm." In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Association for Computing Machinery, 1988, pp. 377–387. DOI: 10.1145/62212.62249.

[26] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. "In-Out Separation and Column Generation Stabilization by Dual Price Smoothing." In: *Experimental Algorithms*. Springer, Berlin, Heidelberg, 2013, pp. 354–365. DOI: 10.1007/978-3-642-38527-8_31.

[27] M. Shigeno. "A Survey of Combinatorial Maximum Flow Algorithms on a Network with Gains (Network Design, Control and Optimization)." In: *Journal of the Operations Research Society of Japan* 47 (Jan. 2004), pp. 244–264. DOI: 10.15807/jorsj.47.244.

[28] 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. Karlsruher Institut für Technologie (KIT), 2024. DOI: 10.5445/IR/1000170863.

[29] L. A. Végh. *Strongly polynomial algorithm for generalized flows.* arXiv abs/1307.6809. 2013. URL: http://arxiv.org/abs/1307.6809.

[30] K. D. Wayne. "A Polynomial Combinatorial Algorithm for Generalized Minimum Cost Flow." In: *Mathematics of Operations Research* 27.3 (2002), pp. 445–459. DOI: 10.1287/moor.27.3.445.313.

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

A Appendix

A.1 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. Tables 12–14 show the results for SoPlex, CPLEX and Gurobi on the nsupp instances while Tables 15–17 show the results for the maxcap instances.

Table 12: Initial heuristic and separation of capacity constraints using SoPlex on nsupp instances.

Class	Default	H	C	H+C
nsupp25	839.42	297.49	306.42	63.36
nsupp50	508.32	246.19	189.30	39.10
nsupp75	286.99	226.20	101.26	23.33
nsupp100	221.31	221.04	83.23	20.92
nsupp125	179.56	216.99	63.23	18.05
nsupp150	127.70	215.22	40.76	13.27

Table 13: Initial heuristic and separation of capacity constraints using CPLEX on nsupp instances.

Class	Default	H	C	H+C
nsupp25	1438.57	271.01	414.49	70.44
nsupp50	969.31	231.19	241.71	42.59
nsupp75	321.31	225.79	133.34	24.85
nsupp100	278.59	225.39	111.41	21.72
nsupp125	221.55	223.92	87.87	18.70
nsupp150	161.97	222.10	57.96	13.80

Table 14: Initial heuristic and separation of capacity constraints using Gurobi on nsupp instances.

Class	Default	H	C	H+C
nsupp25	1286.25	233.68	390.20	56.15
nsupp50	680.84	206.43	220.66	34.14
nsupp75	385.11	205.78	118.74	20.90
nsupp100	310.99	209.91	100.20	18.93
nsupp125	242.43	210.86	79.54	16.74
nsupp150	178.90	210.60	50.20	12.35

Table 15: Initial heuristic and separation of capacity constraints using SoPlex on maxcap instances.

Class	Default	H	C	H+C
maxcap5	3600.46	1083.17	1103.01	185.77
maxcap15	950.71	307.75	343.84	74.18
maxcap25	466.60	198.68	189.47	45.20
maxcap35	335.30	164.87	131.62	34.94
maxcap45	247.52	141.01	97.88	27.77
maxcap55	183.94	118.72	73.06	21.79

Table 16: Initial heuristic and separation of capacity constraints using CPLEX on maxcap instances.

Class	Default	H	C	H+C
maxcap5	2971.92	788.16	1601.43	257.34
maxcap15	1266.09	276.14	462.63	84.37
maxcap25	1439.65	189.54	256.01	47.59
maxcap35	1075.45	160.89	180.08	35.69
maxcap45	514.53	140.42	133.24	28.02
maxcap55	351.36	118.00	98.63	21.86

Table 17: Initial heuristic and separation of capacity constraints using Gurobi on maxcap instances.

Class	Default	H	C	H+C
maxcap5	3600.02	500.91	1331.38	147.66
maxcap15	1508.64	238.12	443.33	67.19
maxcap25	782.17	176.25	248.82	42.38
maxcap35	582.60	153.01	177.13	33.41
maxcap45	439.24	136.69	132.50	26.84
maxcap55	326.28	115.74	98.63	21.19

A.2 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 superior compared to the interior points methods. Table 18 shows the results for the nsupp instances and Table 19 for the maxcap instances.

Table 18: Running times of simplex and interior point methods solving the arc formulation on nsupp instances.

	Simplex		Barrier		Crossover	
Class	CPLEX	Gurobi	CPLEX	Gurobi	CPLEX	Gurobi
nsupp25	36.52	36.69	454.96	959.22	438.02	932.76
nsupp50	31.60	32.51	320.06	764.72	336.15	787.17
nsupp75	28.21	31.16	351.72	665.37	313.43	667.40
nsupp100	26.88	29.73	307.39	651.42	321.63	652.60
nsupp125	24.92	28.75	259.47	530.10	273.38	533.13
nsupp150	24.17	27.96	281.04	560.63	271.05	556.41

Table 19: Running times of simplex and interior point methods solving the arc formulation on maxcap instances.

	Simplex		Barrier		Crossover	
Class	CPLEX	Gurobi	CPLEX	Gurobi	CPLEX	Gurobi
maxcap5	45.96	40.39	469.04	1096.42	564.53	1137.40
maxcap15	37.06	36.58	711.73	904.77	379.94	906.89
maxcap25	34.09	35.60	391.88	892.01	407.22	892.87
maxcap35	32.61	35.80	408.27	904.39	422.34	904.25
maxcap45	31.59	36.24	369.08	836.91	381.62	863.50
maxcap55	31.86	36.33	398.74	817.83	412.59	818.84