Contents

1 Introduction

1.1 Motivation and Outline

Today more and more real-world problems in the areas of simulation and optimization are solved by mathematical and computational methods. A growing number of these problems can be solved without problems, i.e. even huge instances give an optimal or near optimal solution within seconds. Still, there remain problems that even on modern computers are hard to solve. For these problems it is important to find ways to increase the efficiency of the algorithms.

The topic of this thesis arises from the computation of flow in natural gas networks, which is currently developed in the FORNE Project in a cooperation of OGE with universities and research insitutes including ZIB. The flow of natural gas in a network is described by nonlinear equations and depends on many parameters, which makes the problem hard to solve. If we can find good upper and lower bounds for the flow on an arc during the preprocessing, we can hope to improve the behavior of the nonlinear solver by giving these tigther bounds.

The flow is induced by pressure differences, so in reality there can't be cyclic flow (if we exclude compressor stations). Without the condition of acyclic flow, it is sufficient to run a standard min-cost-flow algorithm where the maximized arc e gets weight $w_e = -1$ and all others are 0. However, the arising bounds are far from optimal. If arc e is contained in any cycle we could decrease the cost by pushing more and more flow around this cycle until the arcs capacity is at its limits.

This master thesis will deal with the problem of finding a network flow with no directed cycles (acyclic flow), which at the same time maximizes the amount of flow on a specified arc e of the network. We will discuss the complexity, an exact algorithm based on a mixed integer program with separation of inequalities that forbid cycles and also a heuristic approach that yields results much faster (but not optimal).

1.2 Basic Notation and Definitions

Since there are many definitions, which may differ slightly, we want to introduce now the basic notation and definitions that we use throughout this thesis. The definitions in this chapter are mainly taken from the textbook about combinatorial optimization from Korte and Vygen [3].

An undirected graph is a triple (V, E, Ψ), where V and E are finite sets and $\Psi : E \to \{X \subseteq V : |X| = 2\}$. A directed graph or digraph is a triple (V, E, Ψ), where V and E are finite sets and $\Psi : E \to \{(v, w) \in V \times V : v \neq w\}$. In this thesis by a graph we mean normally the directed graph. If we talk about undirected graphs it will be stated explicitly. The elements of V are called vertices, the elements of E are the edges. Edges of undirected graphs can also be called arcs to make clear that they are directed.

Two edges e, e' with $\Psi(e) = \Psi(e')$ are called parallel. Graphs without parallel edges are called simple. For simple graphs we usually identify an edge e with its image $\psi(e)$ and write $G = (V(G), E(G))$, where $E(G) \subseteq \{X \subseteq V(G) : |X| = 2\}$ or $E(G) \subseteq V(G) \times V(G)$. We often use this simpler notation even in the presence of parallel edges, then the "set" $E(G)$ may contain several "identical" elements. In this thesis all graphs are considered simple if nothing different is said. $|E(G)|$ denotes the number of edges, and for two edge sets E and F we always have $|E \cup F| = |E| + |F|$ even if parallel edges arise.

We say that an edge $e = \{v, w\}$ or $e = (v, w)$ joins v and w. In this case, v and w are adjacent. v is a neighbour of w (and vice versa). v and w are the endpoints of e. If v is an endpoint of an edge e, we say that v is incident with e. In the directed case we say that (v, w) leaves v and enters w, v is the tail and w is the head of the arc e. Two edges which share at least one endpoint are called adjacent.

For a digraph G we sometimes consider the underlying undirected graph, i.e. the undirected graph G' on the same vertex set which contains an edge $\{v, w\}$ for each edge (v, w) of G. We also say that G is an orientation of G'. A subgraph of a graph $G = (V(G), E(G))$ is a graph $H = (V(H), E(H))$ with $V(H) \subset V(G)$ and $E(H) \subset E(G)$. We also say that G contains H. H is an induced subgraph of G if it is a subgraph of G and $E(H) = \{\{x, y\}$ resp. $(x, y) \in E(G) : x, y \in V(H)\}$. Here H is the subgraph of G induced by $V(H)$. We also write $H = G[V(H)]$. A subgraph H of G is called spanning if $V(H) = V(G)$. If $v \in V(G)$, we write $G - v$ for the subgraph of G induced by $V(G) \setminus v$. If $e \in E(G)$, we define $G - e := (V(G), E(G) \setminus \{e\})$. Furthermore, the addition of a new edge e is abbreviated by $G + e := (V(G), E(G) \cup e)$. If G and H are two graphs, we denote by $G + H$ the graph with $V(G + H) = V(G) \cup V(H)$ and $E(G + H)$ being the disjoint union of $E(G)$ and $E(H)$ (parallel edges may arise). For a graph G and $X, Y \subseteq V(G)$ we define $E(X, Y) := \{\{x, y\} \in E(G) : x \in X \setminus Y, y \in Y \setminus X\}$ resp. $E^+(X, Y) := \{(x, y) \in E(G) : x \in X \setminus Y, y \in Y \setminus X\}$. For undirected graphs G and $X \subseteq V(G)$ we define $\delta(X) := E(X, V(G) \setminus X)$. The set of neighbours of X is defined by $\Gamma(X) := \{v \in V(G) \setminus X : E(X, \{v\}) \neq \emptyset\}$. For digraphs G and $X \subseteq V(G)$ we define $\delta^+(X) := E^+(x, V(G) \setminus X)$, $\delta^-(x) := \delta^+(V(G) \setminus X)$ and $\delta(X) := \delta^+(x) \cup \delta^-(x)$. We use subscripts (e.g. $\delta_G(X)$) to specify the graph G if necessary.

For singletons, i.e. one-element vertex sets $\{v\}(v \in V(G))$ we write $\delta(v) := \delta(\{v\})$, $\Gamma(v) := \Gamma(\{v\}), \delta^+(v) := \delta^+(\{v\})$ and $\delta^-(v) := \delta^-(\{v\})$. The degree of a vertex v is $|\delta(v)|$, the number of edges incident to v. In the directed case, the in-degree is $|\delta^-(v)|$, the out-degree is $|\delta^+(v)|$, and the degree is $|\delta^+(v)| + |\delta^-(v)|$. A vertex v with zero degree is called isolated. A graph where all vertices have degree k is called

k-regular.
An edge progression W in G is a sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$ such that $k \geq 0$, and $e_i = (v_i, v_{i+1}) \in E(G)$ resp. $e_i = \{v_i, v_{i+1}\} \in E(G)$ for $i = 1, \ldots, k$. If in addition $e_i \neq e_j \, \forall \, 1 \leq i < j \leq k$, W is called a walk in G. W is closed if $v_1 = v_{k+1}$. A path is a graph $P = (\{v_1, ..., v_{k+1}\}, \{e_1, ..., e_k\})$ such that $v_i \neq v_j$ for $1 \leq i < j \leq k+1$ and the sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$ is a walk. P is also called a path from v_1 to v_{k+1} or a $v_1 - v_{k+1}$-path. v_1 and v_{k+1} are the endpoints of P. By $P_{[x,y]}$ with $x, y \in V(P)$ we mean the (unique) subgraph of P which is an $x - y$-path. Evidently, there is an edge progression from a vertex v to another vertex w if and only if there is a $v - w$-path.
A cycle is a graph $(\{v_1, \ldots, v_k\}, \{e_1, \ldots, e_k\})$ such that the sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_1$ is a (closed) walk and $v_i \neq v_j$ for $1 \leq i < j \leq k$. An easy induction argument shows that the edge set of a closed walk can be partitioned into edge sets of cycles.
The length of a path or cycle is the number of its edges. If it is a subgraph of G, we speak of a path or cycle in G. A spanning path in G is called a Hamiltonian path while a spanning cycle in G is called a Hamiltonian cycle or a tour. A graph containing a Hamiltonian cycle is a Hamiltonian graph. For two vertices v and w we write $dist(v, w)$ or $dist_G(v, w)$ for the length of a shortest $v - w$-path (the distance from v to w) in G. If there is no $v - w$-path at all, i.e. w is not reachable from v, we set $dist(v, w) := \inf$. In the undirected case, $dist(v, w) = dist(w, v)$ for all $v, w \in V(G)$.
We shall often have a cost function $c : E(G) \to \mathbb{R}$. Then for $F \subseteq E(G)$ we write $c(F) := \sum_{e \in F} c(e)$ (and $c(\emptyset) = 0$). This extends c to a modular function $c : 2^{E(G)} \to \mathbb{R}$. Moreover, $dist_{(G,c)}(v, w)$ denotes the minimum $c(E(P))$ over all $v - w$-paths P in G.

1.3 The Acyclic Flowbound Problem

We already described the gas flow problem, where the motivation for this thesis came from. Here we want to define the problem as a general combinatorial flow problem with specific contraints. We will also give the formulation as a Mixed Integer Program (MIP) and as well some natural relaxations, which might be easier to solve.
We will represent our originally undirected graph by a directed graph where flow is allowed to go over edges backward and forward as well. This allows us to specify directions forward and backward on every edge in a consistant way.

Definition 1.1. Let $G = (V, A)$ be a directed Graph and $e \in A$ a specific arc of G. For every vertex $v \in V$ let there be a prescribed amount of flow $b(v) \in \mathbb{R}$ entering or leaving the network, where $\sum_{v \in V} b(v) = 0$.
Let there be capacities $c_l(a) \leq 0 \leq c_u(a) \, \forall a \in A$, and a flow $f : A \to \mathbb{R}$ with $c_l(a) \leq f(a) \leq c_u(a) \, \forall a \in A$ and $\sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a) + b(v) = 0 \, \forall v \in V$. We call this flow a *feasible network flow on G*.

Definition 1.2. A *cyclic flow* within such a feasible network flow f is a flow $f' : C \to \mathbb{R}$, where $C \subseteq A$ is a cycle, $\sum_{a \in \delta^+(v) \cap C} f(a) - \sum_{a \in \delta^-(v) \cap C} f(a) = 0 \, \forall v \in C$, for all arcs the flow directions in f' and f are the same, i.e. $f(a) \geq 0 \Rightarrow f'(a) \geq 0$, $f(a) \leq 0 \Rightarrow f'(a) \leq 0$ and there is really non-zero flow, i.e. $f'(a) \neq 0 \, \forall a \in C$.
If there is any cyclic flow f' in a network flow f we say that f contains a flow cycle. If there is no cyclic flow, we say f is an *acyclic flow* on G.

The conditions imply, that a cyclic flow is one that has no sources or sinks and the same value of flow on each arc.
The *size of a cyclic flow* is the absolute amount of of flow contributing to the cycle, that means it is the maximum value $|f'(a)|$ a cyclic flow f' can reach.

Definition 1.3. Given a graph $G = (V, A)$ like above, with $e \in A$ a specific arc of G. We call the problem of finding an acyclic flow $f : A \to \mathbb{R}$ with $f(e) \geq f'(e) \, \forall f' : A \to \mathbb{R}$ s.t.f' is an acyclic flow on G the *Edge-Maximizing Acyclic Flow Problem*.

Now that we defined our problem, we can investigate it deeper. It is obviously closely related to other flow problems, though the special constraints we built make it more complicated. One way to deal with this is to look at related problems, which could give bounds on our problem or even the same solution in special cases.

1.3.1 Relation To Other Flow Problems

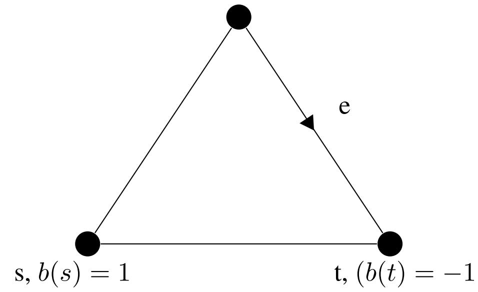
If we look for related problems, it is natural to just drop the constraints of our problem. In this case we could for example easily forget the contraint that flows have to be acyclic. If we do this, our problem becomes a special instance of a Min-Cost-Flow where we allow edge weights to be negative:

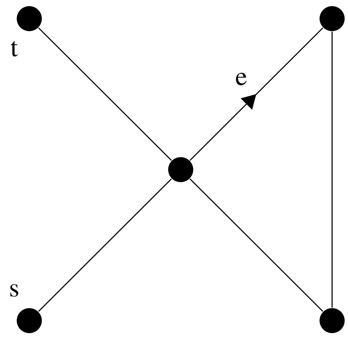
Definition 1.4. The *Minimum Cost Flow Problem* is the problem to determine a feasible network flow with the least possible cost c_f. The cost function $c : A \to \mathbb{R}$ (or often $c : A \to \mathbb{R}_{\geq 0}$) is defined on every arc, the cost of the flow is $c_f = \sum_{a \in A} |f(a)| \cdot c(a)$. So for a given network and cost function we look for a flow f s.t. $c_f \leq c_{f'} \forall f' : A \to \mathbb{R}$.

This problem is normally defined with nonnegative cost functions, like in [1] where Jack Edmonds and Richard Karp present their well known $O(V \cdot E^2)$ flow algorithm. At least cycles of negative weight should be excluded, since they lead to problems in the algorithm's subroutines like shortest path. Nevertheless there are algorithms that can handle negative cycles without a problem - but they will of course give a solution where as much flow as possible is just flowing around these negative cycles.
In our application we want to find upper and lower bounds on the possible flow over each arc. For this we set the weight on the arc e where we want to maximize flow to -1 and compute a Minimum Cost Flow in the graph. If e is contained in a cycle of G whose arcs all have a very high upper capacity bound, e will get a very high bound as well. The reason is that any cyclic flow over e can reduce the cost of the flow. Hence as much cyclic flow as possible will be used to obtain a Minimum Cost Flow.
So in most cases the bound computed with a Min-Cost Flow algorithm is not sharp. Still it gives an upper bound for the possible flow. Hence one could ask how good this bound will be, and whether we can use it as an approximation. It turns out that the gap between the amount of flow on an arc e in an acyclic flow and in a possibly cyclic flow can be arbitrarily large:

Proposition 1.5
The gap between the maximum flow values $f(a)$ on an arc $a \in A$ in a normal and in an acyclic flow problem is unbounded.

Proof. As a proof look at the two examples. In this first picture we see that the flow on the maximized arc e in the acyclic case has to be 1. In the normal case it could be





any number we choose as capacities for all arcs in the network. Hence the relative gap $\frac{f_{cyc}(e)}{f_{acyclic}(e)}$ is $\frac{\text{min capacity on the cycle}}{1}$ which is unbounded.
The second picture shows a graph where the flow on e in the acyclic case is 0. If we do not forbid cyclic flows, the flow on e will always be $\min_{a \in \text{ cycle } C} c_u(a)$. So the gap in relative numbers $\frac{f_{cyc}(e)}{f_{acyclic}(e)}$ is not even defined! □

As a different relaxation we could drop the capacity constraints, but still insist on an acyclic flow that is maximum on an arc e. In the normal Max-Flow Problem this reduces the problem to finding a shortest path between the source and the sink.
In our case , if there is only one source and one sink, we have to decide whether there is a simple path from the source to the sink that contains the maximized arc e. If there are more sources and sinks, we already have the problem of finding a set of simple paths. Is this problem easy to solve, and how to solve it? TODO

1.3.2 Computational Complexity

In order to talk about algorithms it is always good to kno the complexity of the underlying problem. This chapter will deal with this issue. We show, that Cycle-free Min-Cost-Flow with negative weights is NP-complete, and that our problem is just a special case.

2 Models and Solving Approaches

2.1 MIP Formulations

For many combinatorial Problems it is the best practical solution to formulate them as a Mixed Integer Problem and just solve this problem with modern MIP Solvers such as CPlex or Gurobi. Often there are different possible formulations as MIP, which might yield very different running times due to numerical or algorithmical reasons. In our problem, it is easy to model the flow conservation and the ingoing and outgoing flow on vertices. Like described before, we can assign a negative weight to an arc variable and this way maximize flow over this arc by minimizing the overall cost. This would be the typical MIP formulation of a min cost flow:

$$\begin{aligned} &\min \sum_{a \in A} w_a \cdot x_a && \\ s.t. &\sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = b_v && \forall v \in V \\ &c_l(a) \leq x_a \leq c_u(a) && \forall a \in A \end{aligned}$$

This model still allows cyclic flow, so we have to find constraints to avoid cyclicity.

2.1.1 Model 1: Node Potentials

Another idea (that is unfortunately not a linear formulation anymore) would be to set potentials on he nodes and allow only flow from higher to lower potential. This is quite close to reality, where gas will only flow from places wih high pressure to places with lower pressure in the network. So each vertex v would become a variable π_v, each arc a as before a variable x_a for the amount of flow. We have to ensure that the values of the potentials are all different, or that there is no flow between nodes of the same potential. Otherwise any solution where all potentials are set to the same value would fulfill this constraint, regardless if it is acyclic or not. So in pracicewe could set a very small constant $\varepsilon > 0$ to describe the minimum distance between the potentials. A feasible solution now has to fulfill

$$x_a \cdot (\pi_v - \pi_w) \geq 0 \, \forall a = (v, w) \in A$$

We get the Mixed Integer Nonlinear Program

$$
\begin{aligned}
\min \sum_{a\in A} & w_a \cdot x_a \\
s.t. \sum_{a\in\delta^+(v)} x_a - \sum_{a\in\delta^-(v)} x_a &= b_v && \forall v \in V \\
x_a &\le c_u(a) && \forall a \in A \\
-x_a &\le c_l(a) && \forall a \in A \\
-x_a \cdot (\pi_v x - \pi_w) &\le 0 && \forall a = (v,w) \in A \\
(\pi_v - \pi_w)^2 &\ge \varepsilon && \forall v, w \in V \\
x_a &\in \mathbb{R} && \forall a \in A \\
\pi_v &\in \mathbb{R} && \forall v \in V \\
d_a &\in \{0,1\} && \forall a \in A
\end{aligned}
$$

We show that this flow indeed is an acyclic one:

Proposition 2.1
A flow which fulfills the constraints of the above nonlinear program is always acyclic.

Proof. Assume a flow fulfilling the above constraints would have cyclic flow on a cycle C. Without loss of generality we number the vertices on the cycle from 1 to n and assume all arcs are directed forward on this cycle. This means there are arcs $a_1 = (v_1, v_2), a_2 = (v_2, v_3), \ldots a_n = (v_n, v_1) \in C$ such that $x_{a_i} > 0\, \forall i = 1, \ldots, n$. Also we know from the constraints $x_a \cdot (\pi_v - \pi_w) \ge 0$ and $\pi_v - \pi_w \ne 0$ (Hence $\Rightarrow \pi_v \ne \pi_w) \forall v, w \in V$. So we conclude an ordering of the vertices potentials: $x_{a_1} \cdot (\pi_{v_1} - \pi_{v_2}) \ge 0 \Rightarrow \pi_{v_1} > \pi_{v_2}$ and so on. This yields a sequence $\pi_{v_1} > \pi_{v_2} > \cdots > \pi_{v_n} > \pi_{v_1}$, which is a contradiction. ` □

2.1.2 Model 2: Acyclicity Constraints On All Cycles

One idea (which was implemented in Lamatto by Robert Schwarz) is the following: Every arc has a direction and the sign of the flow on this arc tells us in which direction flow is send over this arc. Hence we can introduce variables $d_a \in \{0, 1\}$ for each arc $a \in A$ that indicate the direction of flow and are coupled with the flow variables:

$$
\begin{aligned}
d_a = 1 &\Rightarrow x_a \ge 0 \\
d_a = 0 &\Rightarrow x_a \le 0
\end{aligned}
$$

These indicator constraints can be handled the way they are by standard MIP solvers, but to make them real MIP constraints we have to formulate them as follows (with M a constant bigger than any possible x_a , e.g. $M = \sum_{v\in V} |b_v|$) :

$$
\begin{aligned}
x_a + M \cdot (1 - d_a) &\ge 0 \\
x_a - M \cdot d_a &\le 0
\end{aligned}
$$

Now we have decision variables for the flow direction, we add constraints to avoid cycles. There should be at least one arc in each direction, so we get

$$1 \le \sum_{a\in C \text{ forward}} d_a + \sum_{a\in C \text{ backward}} 1 - d_a \le n - 1$$

Let us formulate this in a slightly different way to get only one sum: For each cycle of size $C_n = C_l + C_m$ let C_m be the number of arcs directed forward and C_l the number of backward directed arcs. We define that always $C_l \leq C_m$, so forward is defined as the direction where more arcs are pointing towards (left or right would only make sense in a planar embedding). Then we get the constraint

$$1 - l \leq \sum_{a \in C} d_a \leq n - (l + 1)$$

that forbids any cyclic flow on C. We will show later under which conditions such constraints also forbid cyclic flow on other cycles.
So our MIP formulation of the model is finally

$$\begin{aligned}
& \min \sum_{a \in A} w_a \cdot x_a & \\
s.t. & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = b_v & \forall v \in V \\
& c_l(a) \leq x_a \leq c_u(a) & \forall a \in A \\
& x_a + M \cdot (1 - d_a) \geq 0 & \forall a \in A \\
& x_a - M \cdot d_a \leq 0 & \forall a \in A \\
& 1 - l \leq \sum_{a \in C} d_a \leq n - (l+1) & \forall \text{ cycle } C \in G \\
& x_a \in \mathbb{R} & \forall a \in A \\
& d_a \in \{0, 1\} & \forall a \in A
\end{aligned}$$

or with only $\leq$ inequalities:

$$\begin{aligned}
& \min \sum_{a \in A} w_a \cdot x_a & & & (1) \\
s.t. & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a & =& b_v & \forall v \in V & \quad (2) \\
& x_a & \leq & c_u(a) & \forall a \in A & \quad (3) \\
& - x_a & \leq & c_l(a) & \forall a \in A & \quad (4) \\
& - x_a - M \cdot (1 - d_a) & \leq & 0 & \forall a \in A & \quad (5) \\
& x_a - M \cdot d_a & \leq & 0 & \forall a \in A & \quad (6) \\
& 1 - l - \sum_{a \in C} d_a & \leq & 0 & \forall \text{ cycle } C \in G & \quad (7) \\
& \sum_{a \in C} d_a + (l+1) - n & \leq & 0 & \forall \text{ cycle } C \in G & \quad (8) \\
& x_a \in \mathbb{R} & & & \forall a \in A & \quad (9) \\
& d_a \in \{0, 1\} & & & \forall a \in A & \quad (10)
\end{aligned}$$

2.2 The number of cycles we have to forbid

2.3 A Path-Based Heuristic Approach

3 Computational Results

4 Summary

References

[1] Jack Edmonds and Richard M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, J. ACM **19** (1972), no. 2, 248–264.

[2] Martin Grötschel, Michael Jünger, and Gerhard Reinelt, *On the acyclic subgraph polytope*, Mathematical Programming **33** (1985), no. 1, 28–42 (English).

[3] Bernhard Korte and Jens Vygen, *Combinatorial optimization: Theory and algorithms*, 4th ed., Springer Publishing Company, Incorporated, 2007.