Lecture Notes

# Multicommodity Flows and Column Generation

Marc Pfetsch
Zuse Institute Berlin
pfetsch@zib.de

last change: 12/18/2006

Technische Universität Berlin
Fakultät II, Institut für Mathematik
WS 2006/07

Ganzzahlige Optimierung im Öffentlichen Verkehr

Ralf Borndörfer, Christian Liebchen, Marc Pfetsch

Chapter 3

# Multicommodity Flows and Column Generation

The goal of this chapter is to give a short introduction into multicommodity flows and column generation. Both will be used later on.

## 3.1 Multicommodity Flows

We begin our journey to multicommodity flows with a review of maximum flows, i.e., the case where we have only one commodity. The main goal in this part is to fix notation and remind the reader of the basic results on max flows.

### *3.1.1 Maximum Flows*

Let $D = (V, A)$ be a directed graph (or *digraph*) with nonnegative capacities $u_a \geq 0$ for each arc $a \in A$. The digraph $D$ together with the vector $(u_a)$ is sometimes called a (capacitated) *network*. Let $s$ and $t$ be two distinct nodes of $D$; $s$ is called *source* and $t$ is called *target*; see Figure 3.1 for an example. A (nonnegative) vector $\boldsymbol{x} \in \mathbb{R}_+^A$ is called an $(s-t)$-*flow*, if it fulfills the following constraints:

$$\sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = 0, \quad \text{for all } v \in V \setminus \{s, t\}. \tag{3.1}$$

Here $\delta^+(v) := \{(v, w) \in A\}$ and $\delta^-(v) := \{(u, v) \in A\}$ are the out-arcs and in-arcs of $v$, respectively. These equations are called *flow conservation constraints*, since they guarantee that the in-flow equals the out-flow for all nodes except $s$ and $t$. The *value* of a flow $(x_a)$ is the net out-flow of the

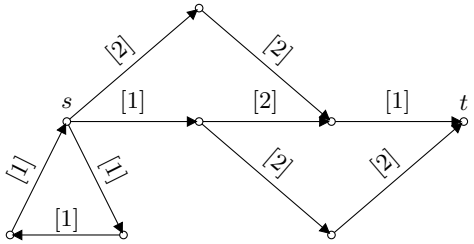


**Figure 3.1:** Example for maximum flows. Given are the capacities of each arc.

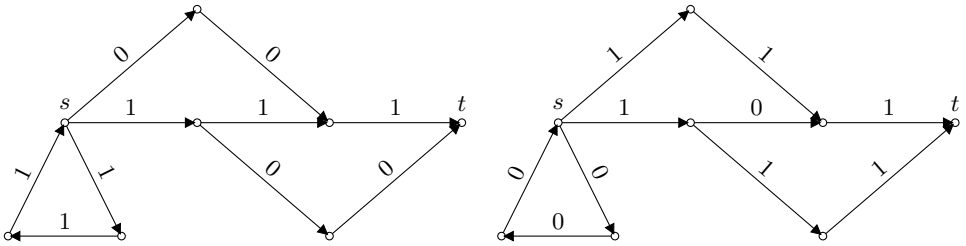


**Figure 3.2:** Two feasible flows for the example: the flow on the left has value 1 and the flow on the right has value 2.

source $s$, i.e., the out-flow minus the in-flow of node $s$:

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

A flow $(x_a)$ is called *feasible* if

$$x_a \leq u_a \qquad \text{for all } a \in A. \tag{3.2}$$

The *maximum $(s-t)$-flow problem* is to find a feasible flow of maximum value. It can be computed by the following linear program:

$$\begin{aligned} \max \sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} x_a & & \\ s.t. \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = 0 & \qquad \text{for all } v \in V \setminus \{s,t\} & \qquad (3.3) \\ 0 \leq x_a \leq u_a & \qquad \text{for all } a \in A. & \end{aligned}$$

Since linear programs can be solved in polynomial time, the maximum $(s-t)$-flow problem can be solved in polynomial time.

An interesting variant is the one in which we require the flow to be integer. In fact, the flow value does not decrease when enforcing this restriction.

**Theorem 3.1** (Ford and Fulkerson, 1968)**.** *If the capacities are integers, i.e., $u_a \in \mathbb{Z}_+$ for all $a \in A$, there exists an* integer *solution $(x_a) \in \mathbb{Z}_+^A$ to the maximum $(s-t)$-flow problem.*

This theorem follows from the so-called *augmenting path* algorithm (of Ford and Fulkerson) that at each stage maintains an integer feasible flow and increases the flow value. A variant of the original algorithm (due to Edmonds and Karp) can be shown to run in time $\mathcal{O}(m^2 n)$ and there exist algorithms that run in time $\mathcal{O}(n^3)$, see, for example, Korte and Vygen [5]. Hence, the *integer* maximum $(s-t)$-flow problem is solvable in *strongly* polynomial time, i.e., in time that only depends on $n$ and $m$ and not on the capacities.

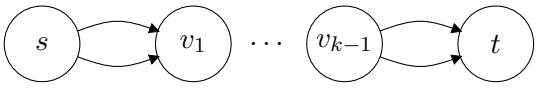


**Figure 3.3:** Example network with exponentially many $(s-t)$-paths. There are $k+1$ nodes in a sequence $s = v_0, v_1, \ldots, v_k = t$. Between each node pair there are two parallel arcs. In total there are $2^k$ paths between $s$ and $t$.

There exists an alternative proof of the above theorem, which shows that the constraint matrix of the above LP is *totally unimodular*. This implies that there exists an integer optimal solution to every linear program with this constraint matrix, as long as the right hand side is integer.

### *3.1.2 Path Flows*

The feasible flows used in formulation (3.3) are also called *edge flows*. There exists an alternative formulation that uses a nonnegative variable $f_p$ for each $(s-t)$-path $p$. An $(s-t)$-path is a sequence of arcs $a_1, \ldots, a_k$ that starts at $s$ and ends at $t$, such that no node is visited twice by the path (this sometimes is called a *simple* path). Let $\mathcal{P}_{st}$ be the set of all $(s-t)$-paths and let $\mathcal{P}_a$ be the paths that use arc $a$, i.e., $\mathcal{P}_a := \{p \in \mathcal{P}_{st} : a \in p\}$. A *feasible path flow* is a nonnegative vector $(f_p)_{p\in\mathcal{P}_{st}}$ such that:

$$\sum_{p\in\mathcal{P}_a} f_p \leq u_a \qquad \text{for all } a \in A.$$

The *path formulation* of the maximum $(s-t)$-flow problem is the following:

$$\begin{aligned} \max \sum_{p\in\mathcal{P}_{st}} f_p & \\ s.t. \sum_{p\in\mathcal{P}_a} f_p \leq u_a & \qquad \text{for all } a \in A \\ f_p \geq 0 & \qquad \text{for all } p \in \mathcal{P}_{st}. \end{aligned} \tag{3.4}$$

Flow conservation constraints like (3.1) are not necessary in this formulation, since the flow is automatically conserved along each path. This simpler structure of (3.4) comes at the cost of exponentially many variables, see Figure 3.3. We will discuss a method that allows to efficiently solve the path formulation in a bit more general context in the next section. We call the sum in the objective function the *value* of the path flow $\boldsymbol{f} = (f_p)_{p\in\mathcal{P}}$.

For the meantime assume that we have a path flow $(f_p)$ available. Then we can easily compute an edge flow $x_a$ as follows:

$$x_a = \sum_{p\in\mathcal{P}_a} f_p.$$

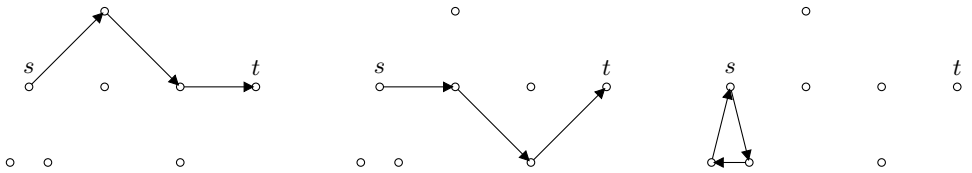


**Figure 3.4:** Path decomposition of the flow on the right hand side of Figure 3.2. The values $f_p$ are equal to 1 in both cases. Left over are only cycles, see the right most picture.

This shows that each path flow uniquely determines an edge flow. Conversely, we can also compute a path flow with the same value from an edge flow by a method that is called *path decomposition*. Let $D(\boldsymbol{x})$ be the directed graph consisting of those arcs $a$ of $D$ for which $x_a > 0$. Let $p \in \mathcal{P}_{st}$ be an $(s-t)$-path in $D(\boldsymbol{x})$. We let

$$f_p := \min\{x_a \,:\, a \in p\}$$

and subtract $f_p$ from $x_a$ for all arcs $a \in p$. Then we repeat the process with the new digraph $D(\boldsymbol{x})$. The process ends when there is no $(s-t)$-path left. We then remove the remaining flow – this flow can only appear along cycles that do not contribute to the flow from $s$ to $t$. See Figure 3.4 for an example.

Note that the path flow is not uniquely determined by the edge flow, but this will not play a role in the following.

### *3.1.3 The Multicommodity Flow Problem*

A generalization of the maximum $(s-t)$-flow problem is the multicommodity flow (MCF) problem. In this setting, we are given $k$ *commodities*, i.e., node pairs $(s_1, t_1)$, ..., $(s_k, t_k)$; we sometimes also call the index $i \in [k]$ a commodity. For each $i = 1, \ldots, k$, we want to find an $(s_i - t_i)$-flow, such that the sum of these flows on each arc does not exceed a given capacity. We call a vector $(x_a^i)$, with $a \in A$, $i \in [k]$, a *feasible multicommodity flow* if it satisfies the following constraints:

$$\sum_{a \in \delta^+(v)} x_a^i - \sum_{a \in \delta^-(v)} x_a^i = 0 \quad \text{for all } v \in V \setminus \{s_i, t_i\},\ i \in [k],$$

$$\sum_{i=1}^{k} x_a^i \le u_a \quad \text{for all } a \in A.$$

The first type of constraints are separate flow conservation constraints for each commodity. The second type implements the *capacity constraints* on each arc. The *multicommodity flow problem* is to maximize the sum of the

flow values of the individual commodities:

$$\begin{aligned}
\max \sum_{i=1}^{k} \Big( \sum_{a \in \delta^+(s_i)} x_a^i - \sum_{a \in \delta^-(s_i)} x_a^i \Big) & \\
s.t. \sum_{a \in \delta^+(v)} x_a^i - \sum_{a \in \delta^-(v)} x_a^i = 0 & \quad \text{for all } v \in V \setminus \{s_i, t_i\},\ i \in [k] \quad (3.5) \\
\sum_{i=1}^{k} x_a^i \leq u_a & \quad \text{for all } a \in A \\
x_a^i \geq 0 & \quad \text{for all } a \in A,\ i \in [k].
\end{aligned}$$

Since this is a linear program, the multicommodity flow problem can be solved in polynomial time. In fact, Tardos showed that it can be solved in strongly polynomial time, i.e., there exists an algorithm whose running time does not depend on the capacities $u_a$ (see Grötschel, Lovász, and Schrijver [4]). Note, however, that the multicommodity flow problem turns $\mathcal{NP}$-hard, if we require the flows to be integers, see Garey and Johnson [3]. This is in contrast to the maximum $(s-t)$-flow problem, see the discussion after Theorem 3.1. Furthermore, it is not true that exists an *integer* multicommodity flow whenever there exists a feasible multicommodity flow.

As for the maximum $(s-t)$-flow problem there exists a path formulation for the multicommodity flow problem. As above, $\mathcal{P}_{st}$ is the set of all (simple) $(s-t)$-paths and we define $\mathcal{P} = \mathcal{P}_{s_1,t_1} \cup \cdots \cup \mathcal{P}_{s_k,t_k}$. Again $\mathcal{P}_a$ is the set of paths that use arc $a$, i.e., $\mathcal{P}_a := \{p \in \mathcal{P} : a \in p\}$. We have a variable $f_p$ for each $p \in \mathcal{P}_{s_i,t_i}$ and $i \in [k]$. This yields the following program:

$$\begin{aligned}
\max \sum_{p \in \mathcal{P}} f_p & \\
s.t. \sum_{p \in \mathcal{P}_a} f_p \leq u_a & \quad \text{for all } a \in A \quad (3.6) \\
0 \leq f_p & \quad \text{for all } p \in \mathcal{P}.
\end{aligned}$$

A variant is the following. We are given arc costs $c_a$ for every $a \in A$ and demand values $d_{s_i,t_i} \geq 0$ for $i = 1, \ldots, k$. The goal is to find a minimum cost multicommodity flow that fulfills all demands, i.e.,

$$\begin{aligned}
\min \sum_{p \in \mathcal{P}} \big( \sum_{a \in p} c_a \big) f_p & \\
s.t. \sum_{p \in \mathcal{P}_{s_i,t_i}} f_p = d_{s_i,t_i} & \quad \text{for all } i = 1, \ldots, k \\
\sum_{p \in \mathcal{P}_a} f_p \leq u_a & \quad \text{for all } a \in A \\
0 \leq f_p & \quad \text{for all } p \in \mathcal{P}.
\end{aligned}$$

We will use this variant in some line planning models, but will use the first model in this chapter because it is a bit easier to derive the column generation algorithm for it.

Although the edge formulation (3.5) shows that the MCF problem can be solved in polynomial time, in practice, it is often a good idea to use the path formulation and column generation instead (see the next section). The reason is that the storage needed for the edge formulation is often very large: assume that the graph has 1000 nodes and 10000 edges, a size that still seems reasonable. Furthermore, assume that you want to send flow from every node to every other, i.e., the number of commodities is $1000 \cdot 1000 = 1,000,000$. The number of variables in (3.5) is then: $1,000,000 \cdot 10000 = 10,000,000,000$. If you take 8 Bytes for each variable, you already need a storage of at least 80 Giga-Byte. In practice, often column generation can solve the path formulation (3.6) with much less storage (and time).

## 3.2 Column Generation

Column generation is a method to efficiently solve linear programs with a huge number of variables. It is dual to the cutting plane method. The principal idea, due to Ford and Fulkerson [2], works as follows: We want to solve an LP, called *master LP*, and consider a *restricted master LP*, which contains all constraints of the master LP, but only a subset of the variables. Then we solve a *pricing problem*, i.e., we decide whether there is a variable that is currently not contained in the restricted master LP, but might improve the objective function. If there are no such variables, it is guaranteed that the current solution of the restricted master LP is optimal for the whole problem. Otherwise, we add the variables and iterate.

### *3.2.1 Interlude: Duality Theorem for Linear Programming*

We need some facts from duality theory for linear programs. Consider the following linear program, the so-called *primal program*:

$$\begin{aligned} \max\ & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ s.t.\ & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. A vector $x \in \mathbb{R}^n$ is *feasible* for the primal program if $A\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$. The *dual program* is

$$\begin{aligned} \min\ & \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} \\ s.t.\ & A^{\mathrm{T}}\boldsymbol{y} \geq \boldsymbol{c} \\ & \boldsymbol{y} \geq \boldsymbol{0}. \end{aligned}$$

A vector $\boldsymbol{y} \in \mathbb{R}^m$ is *feasible* for the dual program if $A^{\mathrm{T}}\boldsymbol{y} \geq \boldsymbol{c}$ and $\boldsymbol{y} \geq \boldsymbol{0}$.

**Lemma 3.2** (Weak Duality)**.** *We have*

$$\max\{\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \,:\, A\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\} \leq \min\{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} \,:\, A^{\mathrm{T}}\boldsymbol{y} \geq \boldsymbol{c},\ \boldsymbol{y} \geq \boldsymbol{0}\},$$

*if both programs have a feasible solution.*

*Proof.* Let $\boldsymbol{x} \geq \boldsymbol{0}$ satisfy $A\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{y} \geq \boldsymbol{0}$ satisfy $A^{\mathrm{T}}\boldsymbol{y} \geq \boldsymbol{c}$, i.e., $\boldsymbol{x}$ and $\boldsymbol{y}$ are feasible for the primal and dual program, respectively. Then we have

$$\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \leq (A^{\mathrm{T}}\boldsymbol{y})^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{y}^{\mathrm{T}}A\boldsymbol{x} \leq \boldsymbol{y}^{\mathrm{T}}\boldsymbol{b},$$

because $\boldsymbol{x},\ \boldsymbol{y} \geq \boldsymbol{0}$. □

We define $\max \varnothing := -\infty$ and $\max\{\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \,:\, \boldsymbol{x} \in X\} := \infty$, if the set $X$ is *unbounded* with respect to $\boldsymbol{c}$ (and taking the maximum), i.e., there exists $\boldsymbol{x} \in X$ and $\boldsymbol{r} \in \mathbb{R}^n$ with $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{r} > 0$ such that $\boldsymbol{x} + \lambda\,\boldsymbol{r} \in X$ for all $\lambda \geq 0$. Similarly, we define $\min \varnothing := \infty$ and $\min\{\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \,:\, \boldsymbol{x} \in X\} := -\infty$ if $X$ is unbounded with respect to $\boldsymbol{c}$ (and taking the minimum), i.e., there exists $\boldsymbol{x} \in X$ and $\boldsymbol{r} \in \mathbb{R}^n$ with $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{r} < 0$ such that $\boldsymbol{x} + \lambda\,\boldsymbol{r} \in X$ for all $\lambda \geq 0$.

**Theorem 3.3** (Duality Theorem)**.**

$$\max\{\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \,:\, A\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\} = \min\{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} \,:\, A^{\mathrm{T}}\boldsymbol{y} \geq \boldsymbol{c},\ \boldsymbol{y} \geq \boldsymbol{0}\},$$

*if at least one of the programs has a feasible solution.*

**Remark 3.4.** From the duality theorem it follows that if the primal is unbounded, then the dual has no feasible solution and conversely. If they are both feasible their optimal solution values are equal. It can, however, happen that both programs have no feasible solution.

### *3.2.2 Column Generation for MCF*

Let us describe the column generation method for the multicommodity flow problem formulation (3.6). We denote by (MCF) the master LP (3.6) and its dual by (D-MCF). Let $\mathcal{P}' \subseteq \mathcal{P}$ be a subset of all paths and consider the restricted master LP w.r.t. $\mathcal{P}'$:

$$\begin{aligned}
(\mathrm{MCF}') \quad \max \ & \sum_{p \in \mathcal{P}'} f_p & & \\
s.t. \ & \sum_{p \in \mathcal{P}'_a} f_p \leq u_a & & \text{for all } a \in A \\
& f_p \geq 0 & & \text{for all } p \in \mathcal{P}'.
\end{aligned}$$

A feasible solution $(f'_p)_{p\in\mathcal{P}'}$ for (MCF$'$) yields a feasible solution $(f_p)_{p\in\mathcal{P}}$ for (MCF) by setting $f_p = f'_p$ for $p \in \mathcal{P}'$ and $f_p = 0$ for $p \in \mathcal{P} \setminus \mathcal{P}'$. It follows that the optimal solution value $v(\text{MCF}')$ of (MCF$'$) is not larger than the optimal solution value of (MCF), i.e., $v(\text{MCF}') \leq v(\text{MCF})$. We want to decide whether the solution $(f'_p)$ is optimal for (MCF). If this is not the case, we want to add additional variables to (MCF$'$) that help improve the solution value. How can we do that?

We use the *dual restricted master LP*:

$$\begin{aligned}
(\text{D-MCF}') \quad \min & \sum_{a\in A} u_a\,\mu_a & \\
s.t. & \sum_{a\in p} \mu_a \geq 1 & \text{for all } p \in \mathcal{P}' \\
& \mu_a \geq 0 & \text{for all } a \in A.
\end{aligned}$$

Note that $\mathbf{0}$ is feasible for (MCF) and (MCF$'$), and they are bounded (their flow value is not larger than $\sum_{a\in A} u_a < \infty$). Hence, it follows from the duality theorem that $v(\text{D-MCF}')$, $v(\text{D-MCF}) < \infty$ and

$$v(\text{D-MCF}') = v(\text{MCF}') \leq v(\text{MCF}) = v(\text{D-MCF}). \tag{3.7}$$

Let $\boldsymbol{\mu}$ be a solution to (D-MCF$'$) and imagine that the dual constraints

$$\sum_{a\in p} \mu_a \geq 1 \tag{3.8}$$

are even satisfied for all $p \in \mathcal{P}$ (and not only for $p \in \mathcal{P}'$ as guaranteed by the formulation of (D-MCF$'$)). Then $\boldsymbol{\mu}$ is feasible (and hence optimal) for (D-MCF). Therefore, in this case

$$v(\text{D-MCF}) = v(\text{D-MCF}'),$$

and equality holds in (3.7) throughout. This shows that our solution $(f'_p)$ to (MCF$'$) was in fact optimal for the original problem (MCF). Hence, we are done in this case. If there exists a path $p$ for which (3.8) is violated, we add the corresponding variable to (MCF$'$) and repeat the process.

Deciding whether (3.8) is satisfied for all paths is the so-called *pricing problem* and can efficiently be done as follows. We compute a shortest path w.r.t. to the weights $\boldsymbol{\mu} \geq \mathbf{0}$ for each commodity $(s_i, t_i)$, e.g., by Dijkstra's algorithm (see, for instance, Korte and Vygen [5]); this is possible since the weights are nonnegative. If the shortest paths are all at least of weight 1, we know that (3.8) is satisfied for all paths, otherwise we have found a path that violates (3.8). See Algorithm 1.

We have observed that the pricing problem can be solved in polynomial time. Hence the big question is how many iterations this algorithm will take.

**Algorithm 1** Column Generation for MCF

1: **repeat**
2: Solve (MCF$'$) for $\mathcal{P}'$
3: Let $\boldsymbol{\mu}$ be the dual variables for (D-MCF$'$)
4: **for** $i = 1, \ldots, k$ **do**
5: Find a shortest $(s_i - t_i)$-path w.r.t. weights $(\mu_a)$
6: If weight of shortest path $p$ is less than 1, add $p$ to (MCF$'$)
7: **end for**
8: **until** no path has been added

Do we need to generate exponentially many paths (variables) to solve (MCF) to optimality in the worst case? The following general theorem shows that if one takes care in the implementation, i.e., one uses the ellipsoid method to solve the LPs, one only needs polynomially many iterations.

**Theorem 3.5.** *If the pricing problem for a linear program with a fixed number of constraints, but arbitrarily many variables (not counting for the input length), can be solved in polynomial time, the LP can be solved in polynomial time.*

*Proof.* Column generation is the dual process to a cutting plane method. The pricing problem corresponds to a so-called *separation problem*. Here, for a class of inequalities, one has to decide whether all inequalities are satisfied by a given point or to produce a violated inequality.

From the results of Grötschel, Lovász, and Schrijver [4] on the ellipsoid method, it follows that a cutting plane algorithm runs in polynomial time if and only if the separation problem can be solved in polynomial time. This translates to the statement of the theorem. □

**Corollary 3.6.** *The path formulation* (3.6) *for the MCF can be solved in polynomial time.*

**Remark 3.7.** In fact, our presentation above was in terms of the separation problem for inequalities (3.8). We showed that this separation problem can be solved in polynomial time. The result of Grötschel, Lovász, and Schrijver, mentioned in the proof of Theorem 3.5, is often referred to as (polynomial time) equivalence between optimization and separation.

On can even show that one only needs $|A|$ many paths in $\mathcal{P}'$ to find an optimal solution for (MCF). Hence, one only needs very few variables compared to the edge formulation.

**Lemma 3.8.** *There exists a set of paths $\mathcal{P}' \subseteq \mathcal{P}$ of size $|\mathcal{P}'| = |A|$ such that $v(\text{MCF}') = v(\text{MCF})$.*

*Proof.* Consider a basic feasible optimal solution to (MCF). Since there are $|A|$ many constraints, the basis is of this size. Hence, at most $|A|$ components (i.e., paths) have a nonzero entry. Therefore, it suffices to let $\mathcal{P}'$ consist of the corresponding paths. □

**Remark 3.9.** The problem remains, however, to find the optimal variables. This might take a long time, since in practice it often happens that the inequalities corresponding to some paths are violated, yet adding the variable does not improve the objective function value, i.e., the variables are zero in the master LP.

**Remark 3.10.** One should note that one can also develop a column generation method for the edge formulation of MCF, since also there most variables will be zero. However, since there are more constraints in the formulation, the size of the bases are larger and hence a larger number of variables can be nonzero.

### *3.2.3 Alternative View 1: Reduced Costs*

Column generation can also be viewed as a pivot step in the reduced simplex algorithm. We again explain it for the example of multicommodity flows.

Consider the path formulation of MCF in matrix form:

$$\begin{aligned} \text{(MCF)} \quad & \max \mathbb{1}^{\mathrm{T}} \boldsymbol{y} \\ & s.t.\ M\boldsymbol{y} \leq \boldsymbol{u} \\ & \boldsymbol{y} \geq \mathbf{0}. \end{aligned}$$

Here, $\mathbb{1}$ is the vector of all ones, and $M \in \mathbb{R}^{m \times n}$ is the constraint matrix of the capacity constraints. We add slack variables $\boldsymbol{s} \in \mathbb{R}^m$ to obtain the equivalent formulation in standard form:

$$\begin{aligned} & \max \mathbb{1}^{\mathrm{T}} \boldsymbol{y} \\ & s.t.\ M\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{u} \\ & \boldsymbol{y}, \boldsymbol{s} \geq \mathbf{0}. \end{aligned}$$

Setting $A := [M\ I] \in \mathbb{R}^{m \times (n+m)}$, where $I$ is the identity matrix of size $m \times m$, we get

$$\begin{aligned} & \max \boldsymbol{c}^{\mathrm{T}} \boldsymbol{z} \\ & s.t.\ A\boldsymbol{z} = \boldsymbol{u} \\ & \boldsymbol{z} \geq \mathbf{0}, \end{aligned}$$

where $\boldsymbol{c} = (\mathbb{1}, \mathbf{0})$ and $\boldsymbol{z} = (\boldsymbol{y}, \boldsymbol{s})$. Let now $B \subseteq \{1, \ldots, n+m\}$ be a basis for $A$, i.e., $|B| = m$ and $A_B$ (the submatrix of $A$ obtained by taking the

columns indexed by $B$) is invertible. The *reduced costs* corresponding to $B$ are defined by

$$r_j := c_j - \boldsymbol{c}_B^{\mathrm{T}} A_B^{-1} \boldsymbol{A}_j \qquad \text{for } j \in N := \{1, \ldots, n+m\} \setminus B,$$

where $\boldsymbol{c}_B$ is the subvector of $\boldsymbol{c}$ indexed by $B$ and $\boldsymbol{A}_j$ is the $j$th column of $A$. The vector $\boldsymbol{c}_B^{\mathrm{T}} A_B^{-1} =: \boldsymbol{\mu}^{\mathrm{T}}$ is the dual solution corresponding to $B$.

In the reduced simplex algorithm the pivot step chooses $j \in N$ with $r_j > 0$. Such a column might increase the objective function value if pivoted into the new basis $B$. For the MCF this means that

$$0 < r_j = c_j - \boldsymbol{\mu}^{\mathrm{T}} \boldsymbol{A}_j.$$

If $j \in \{1, \ldots, n\}$, the $j$th column of $A$ is the incidence vector of a path $p$ and $c_j = 1$. This leads to

$$0 < r_j = 1 - \sum_{a \in p} \mu_a \qquad \Leftrightarrow \qquad \sum_{a \in p} \mu_a < 1,$$

which is exactly the condition that we derived above for $p$ being a candidate path. For $j \in \{n+1, \ldots, n+m\}$, we get

$$0 < r_j = 0 - \boldsymbol{\mu}^{\mathrm{T}} \boldsymbol{e}_{j-n} = -\mu_{j-n}.$$

Here, $\boldsymbol{e}_i$ is the $i$th unit vector. Since by definition $\boldsymbol{\mu} \geq \boldsymbol{0}$ this condition is never fulfilled.

Summarizing, the revised simplex can be viewed as a special case of column generation. The pivot choice of a new column entering the basis corresponds to the pricing problem.

### *3.2.4 Alternative View 2: Dantzig-Wolfe Decomposition*

Dantzig-Wolfe decomposition can be seen as a general way to derive models that are suitable to column generation. Generating columns will correspond to generating vertices or rays of a particular polyhedron. As above, we explain the method on the example of MCF.

We start with the MCF in its edge formulation:

$$\begin{aligned} \max\ & \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x} \\ s.t.\ & N\boldsymbol{x} = \boldsymbol{0} \\ & U\boldsymbol{x} \leq \boldsymbol{u} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned}$$

Here, $N\boldsymbol{x} = \boldsymbol{0}$ codes flow conservation and $U\boldsymbol{x} \leq \boldsymbol{u}$ the capacity constraints. We now take the subset of the constraints corresponding to the flow conservation and nonnegativity constraints and consider the corresponding polyhedron:

$$P := \{\boldsymbol{x} \geq \boldsymbol{0} \,:\, N\boldsymbol{x} = \boldsymbol{0}\}.$$

This polyhedron has a single vertex $\mathbf{0}$ (and hence is a cone) and a finite number of rays $\boldsymbol{r}^1, \ldots, \boldsymbol{r}^s$. By the Minkowski-Weyl theorem (see Schrijver [6]) we can write

$$P = \text{cone}\{\boldsymbol{r}^1, \ldots, \boldsymbol{r}^s\} := \{\lambda_1 \boldsymbol{r}^1 + \cdots + \lambda_s \boldsymbol{r}^s \,:\, \lambda_1, \ldots, \lambda_s \geq 0\}.$$

It turns out that in our case the rays $\boldsymbol{r}^i$ correspond to incidence vectors of paths and cycles in the digraph. The above decomposition corresponds to the path decomposition mentioned in Section 3.1.2. More precisely, the rays have the following structure

$$(\boldsymbol{r}^i)^{\mathrm{T}} := [0, \ldots, 0, (\tilde{\boldsymbol{r}}^i)^{\mathrm{T}}, 0, \ldots, 0].$$

Here, $\tilde{\boldsymbol{r}}^i$ correspond to entries of some commodity $j$ and is a multiple of an $(s_j - t_j)$-path or a cycle. Because of the flow conservation constraints, all nonzero entries in $\tilde{\boldsymbol{r}}^i$ have the same size $w_i$. Because cycles do not change the flow value in the model below, we can assume that they do not appear among the $\boldsymbol{r}^i$.

We now use the above description to represent a feasible $\boldsymbol{x}$ and plug it into the original formulation. This yields:

$$\begin{aligned}
\max\ & \lambda_1 \boldsymbol{c}^{\mathrm{T}} \boldsymbol{r}^1 + \cdots + \lambda_s \boldsymbol{c}^{\mathrm{T}} \boldsymbol{r}^s \\
s.t.\ & \lambda_1 N \boldsymbol{r}^1 + \cdots + \lambda_s N \boldsymbol{r}^s = \mathbf{0} \\
& \lambda_1 U \boldsymbol{r}^1 + \cdots + \lambda_s U \boldsymbol{r}^s \leq \boldsymbol{u} \\
& \lambda_1 \boldsymbol{r}^1 + \cdots + \lambda_s \boldsymbol{r}^s \geq \mathbf{0} \\
& \lambda_1, \ldots, \lambda_s \geq 0.
\end{aligned}$$

Since by construction, $N\boldsymbol{r}^i = \mathbf{0}$ and $\boldsymbol{r}^i \geq \mathbf{0}$, we can drop the first and third class of constraints to get

$$\begin{aligned}
\max\ & \lambda_1 \boldsymbol{c}^{\mathrm{T}} \boldsymbol{r}^1 + \cdots + \lambda_s \boldsymbol{c}^{\mathrm{T}} \boldsymbol{r}^s \\
s.t.\ & \lambda_1 U \boldsymbol{r}^1 + \cdots + \lambda_s U \boldsymbol{r}^s \leq \boldsymbol{u} \\
& \lambda_1, \ldots, \lambda_s \geq 0.
\end{aligned}$$

We can now set $y_i = w_i \lambda_i$ (see above for a definition of $w_i$). These new variables denote the flow on the path $p_i$ corresponding to $\boldsymbol{r}^i$. If we now analyze the structure of $\boldsymbol{c}$ and $U$ be note that we have recovered the path formulation of MCF:

$$\begin{aligned}
\max\ & y_1 + \cdots + y_s \\
s.t.\ & \sum_{i : a \in p_i} y_i \leq u_a \qquad \text{for all } a \in A \\
& y_1, \ldots, y_s \geq 0,
\end{aligned}$$

This is equivalent to (3.6).

Therefore, Dantzig-Wolfe decomposition derives the path formulation from the edge formulation of MCF. Furthermore, the pricing problem corresponds to a generation of rays in $P$.

Let us close this chapter with a note on the general method of Dantzig-Wolfe decomposition and its relation to solving integer programs.

Consider an integer program

$$\begin{aligned} \max\ & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ s.t.\ & A\boldsymbol{x} \leq \boldsymbol{b} \\ & D\boldsymbol{x} \leq \boldsymbol{d} \\ & \boldsymbol{x} \in \mathbb{Z}^n_+. \end{aligned} \tag{3.9}$$

Again, we chose a subset of the constraints, include the integrality constraints, and take the convex hull:

$$P := \mathrm{conv}\{\boldsymbol{x} \in \mathbb{Z}^n_+ \,:\, A\boldsymbol{x} \leq \boldsymbol{b}\}.$$

This again is a polyhedron, and hence, by the Minkowski-Weyl theorem, we can write

$$P = \mathrm{conv}\{\boldsymbol{v}^1, \ldots, \boldsymbol{v}^t\} + \mathrm{cone}\{\boldsymbol{r}^1, \ldots, \boldsymbol{r}^s\},$$

where

$$\mathrm{conv}\{\boldsymbol{v}^1, \ldots, \boldsymbol{v}^t\} := \{\mu_1\boldsymbol{v}^1 + \cdots + \mu_t\boldsymbol{v}^t \,:\, \sum_{i=1}^{t} \mu_i = 1,\ \mu_1, \ldots, \mu_t \geq 0\}.$$

Substituting $\boldsymbol{x}$ using this representation and removing the constraints $A\boldsymbol{x} \leq \boldsymbol{b}$ (which are automatically fulfilled) yields the *master problem*:

$$\begin{aligned} \max\ & \sum_{i=1}^{t} \mu_i \boldsymbol{c}^{\mathrm{T}}\boldsymbol{v}^i + \sum_{i=1}^{s} \lambda_i \boldsymbol{c}^{\mathrm{T}}\boldsymbol{r}^i \\ s.t.\ & \sum_{i=1}^{t} \mu_i D\boldsymbol{v}^i + \sum_{i=1}^{s} \lambda_i D\boldsymbol{r}^i \leq \boldsymbol{d} \\ & \sum_{i=1}^{t} \mu_i = 1 \\ & \sum_{i=1}^{t} \mu_i \boldsymbol{v}^i + \sum_{i=1}^{s} \lambda_i \boldsymbol{r}^i \in \mathbb{Z}^n_+ \\ & \mu_1, \ldots, \mu_t, \lambda_1, \ldots, \lambda_s \geq 0. \end{aligned} \tag{3.10}$$

Note that we have to keep the constraint that $\boldsymbol{x}$ has to be integral, if we want to solve the integer program by branch-and-bound methods; for solving the LP-relaxation it of course can be removed.

The idea of including the integrality requirement into $P$ is to improve the value of the LP-relaxation, since the feasible set is smaller. However, if

$$P' := \text{conv}\{\boldsymbol{x} \in \mathbb{R}^n_+ \,:\, A\boldsymbol{x} \leq \boldsymbol{b}\}$$

is integral, i.e., has only integral vertices, then it turns out that this "does not help":

**Theorem 3.11.** *Let $P'$ be an integral polyhedron. Then the value of the LP-relaxation of (3.10) is equal to the value of the LP-relaxation of (3.9).*

The proof is not hard, but we will omit it here.

### *3.2.5 Final Remarks on Column Generation*

In the above presentation we explained column generation at the example of the multicommodity flow problem only. It should, however, not be too hard to generalize the approach, once the main features have become clear.

Much more could be said about column generation, especially about solving integer programs with column generation. This will play a role when solving vehicle and duty scheduling problems. The article of Desrosiers and Lübbecke [1] can be a starting point for further reading, as well as the other articles in this book.

# Bibliography


[1] J. Desrosiers and M. E. Lübbecke, *A primer in column generation*, in Column Generation, G. Desaulniers, J. Desrosiers, and M. M. Solomon, eds., Springer-Verlag, New York, 2005, ch. 1, pp. 1–32.

[2] L. R. Ford, jr. and D. R. Fulkerson, *A suggested computation for maximal multi-commodity network flows*, Manage. Sci. **5** (1958), pp. 97–101.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[4] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Algorithms and Combinatorics 2, Springer-Verlag, Heidelberg, 2nd ed., 1993.

[5] B. Korte and J. Vygen, *Combinatorial optimization. Theory and algorithms*, Algorithms and Combinatorics 21, Springer, Berlin, 2nd ed., 2002.

[6] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, Chichester, 1986.