Strongly polynomial algorithms and generalized flows

Summer School on Combinatorial Optimization
Hausdorff Center for Mathematics

László Végh
l.vegh@lse.ac.uk

Department of Mathematics
London School of Economics and Political Science

August 2018

Contents

Chapter 1

Minimum-cost flows

1.1 Strongly polynomial algorithms

We assume familiarity with the standard notion of Turing complexity. Consider a problem that is given by an input of N rational numbers in binary description. An algorithm for such a problem is *strongly polynomial*, if

(i) It only uses elementary arithmetic operations: addition, comparison, multiplication, and division;

(ii) the total number of arithmetic operations is bounded by $\mathrm{poly}(N)$;

(iii) the algorithm runs in *polynomial space*: that is, the size of the numbers occurring throughout the algorithm remain polynomial in the size of the input.

Above, for a rational number p/q with integers p and q, its *size* is defined as $\mathrm{size}(p/q) := \lceil \log_2(|p|+1)\rceil + \lceil \log_2(|q|+1)\rceil$, the number of bits needed for binary encoding. The size of a vector of rationals is the sum of the sizes of its components.

Alternatively, we can think of the strongly polynomial computation model as one applicable to the *real model of computation*. In this case, the input can be given by real numbers, and elementary arithmetic operations are carried out by oracles provided (with infinite precision). Thus, we omit requirement (iii). (However, for a realistic model, one needs to work with approximate computations with bounded error instead of infinite precision.)

Linear Programming Our main interest will be special cases of linear programming (LP). For $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, consider

$$\begin{aligned} \min\ & c^\top x \\ Ax &= b \\ x &\geq 0. \end{aligned} \tag{1.1}$$

Here, the input is given by $N = n \times m + n + m$ rational numbers; let L denote the total encoding length of the problem. In this context, an algorithm polynomial in the Turing model is called *weakly polynomial*. The first weakly polynomial algorithm for LP was Khachiyan's ellipsoid algorithm [9]. The number of arithmetic operations in the ellipsoid algorithm inherently depends on L, and the same holds for all subsequent polynomial time algorithms such as interior point methods. We will also use the term *pseudopolynomial* to refer to algorithms whose running time is polynomial in the value of the input numbers (instead of their sizes).

It is an outstanding open question is to *develop a strongly polynomial algorithm for LP*. This is listed by Fields medalist Smale not only as the main unsolved problem in linear programming theory, but as one of the most important challenges for the twenty-first century in the entire field of mathematics [18].

1.1.1 Examples

Repeated squaring Consider the problem where the input includes a single number $k = 2^\ell$, in a binary representation (that is, requiring ℓ bits), and the algorithm needs to output $2^k = 2^{2^\ell}$. The number of elementary arithmetic operations is polynomial as it can be obtained by repeated squaring. However, the space requirement to describe the output in standard binary representation is exponential in the input, and thus this algorithm does not run in polynomial space.

Euclidean algorithm Consider the Euclidean algorithm for computing the greatest common divisor of two integers a and b. This is also a weakly but not strongly polynomial time algorithm; notice that the number of arithmetic operations would need to be constant. As a consequence, in the strongly polynomial model we cannot simply replace rational numbers in the form p/q by co-prime representations.

Gaussian elimination The classical Gaussian elimination algorithm can be used to bring a matrix $A \in \mathbb{Q}^{m\times n}$ to the upper triangular form as follows. We construct a sequence $A_0 = A, A_1, \dots, A_r$ of matrices for $r = \mathrm{rk}(A) \le m$, such that

$$A_k = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix},$$

where B is a $k \times k$ strict upper triangular matrix. To obtain A_{k+1} from A_k, we permute the rows of D such that $d_{11} \neq 0$ (this is called the pivot element), and subtract a multiple of the first row of D from all other rows such that d_{j1} becomes 0 for all $j > 1$. Namely, we set

$$d'_{ij} := d_{ij} - \frac{d_{i1}}{d_{11}} d_{1j}. \tag{1.2}$$

It is easy to see that the number of elementary arithmetic operations is bounded by $O(m^2 n)$, however, the space complexity becomes a problematic issue. We could easily keep the size of numbers under control by maintaining co-prime representations, but that is not admissible as pointed out above.

To highlight the problem, assume that the input matrix is integer, and at every elimination step we maintain an integer matrix. This can be achieved by modifying the update formula (1.2) to

$$d'_{ij} := d_{ij} d_{11} - d_{i1} d_{1j}. \tag{1.3}$$

Using this formula, the size of the numbers could become exponential, as illustrated by the following example:

$$\begin{pmatrix} 2 & 0 & 0 & \dots & 0 \\ 1 & 2 & 0 & \dots & 0 \\ 1 & 1 & 2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 1 & 1 & 1 & \dots & 2 \end{pmatrix} \tag{1.4}$$

Nevertheless, Edmonds [3] has shown that the original update formula (1.2) yields a strongly polynomial algorithm. See also [7, Section 1.4] for the proof.

1.2 Network flow problems

Consider a directed graph $G = (V, E)$ with arc capacities $u : E \to \mathbb{Q}_+ \cup \{\infty\}$. We will use $n := |V|$ and $m := |E|$ throughout. For a set of nodes $S \subseteq V$, we let $\delta^+(S)$ and $\delta^-(S)$ denote the set of outgoing and incoming arcs, respectively; for a node $v \in V$ we let $\delta^+(v) := \delta^+(\{v\})$, $\delta^-(v) := \delta^-(\{v\})$. For any arc set $F \subseteq E$, we let $\overleftarrow{F} := \{ji : ij \in E\}$ denote the set of reverse arcs of F, and $\overleftrightarrow{F} := F \cup \overleftarrow{F}$.

For a vector $v : X \to \mathbb{R}$, we let $\mathrm{supp}(v) := \{i \in X : v_i \neq 0\}$, and for subset $Y \subseteq X$, we let $v(Y) := \sum_{i\in Y} v_i$. We let $\|v\|_1 := \sum_{i\in X} |v_i|$ and $\|v\|_\infty = \max_{i\in X} |v|$ denote the ℓ_1 and ℓ_∞ norms, respectively.

For a vector $f : E \to \mathbb{R}_+$, we define the *balance* at node $i \in V$ as

$$\nabla f_i := \sum_{ji \in \delta^-(i)} f_{ji} - \sum_{ij \in \delta^+(i)} f_{ij}.$$

Given source $s \in V$ and sink $t \in V$, the maximum flow problem is

$$\begin{aligned} \max \ & \nabla f_t \\ \nabla f_i &= 0 \quad \forall i \in V \setminus \{s,t\} \\ 0 \leq f &\leq u. \end{aligned}$$

A feasible f is called a *flow*, and the flow value is ∇f_t. An *$s-t$ cut* is a subset of nodes $S \subseteq V$ with $s \in S$, $t \notin S$, and the value of the cut is $u(\delta^+(S))$. Recall the maximum flow-minimum cut (MFMC) theorem:

Theorem 1.1. *Given a capacitated network (V, E, u) and $s, t \in V$, the maximum value of an $s-t$ flow equals the minimum value of an $s-t$ cut.*

Residual network Given a flow f, we define the *residual graph* $G_f = (V, E_f)$ such that $ij \in E_f$ if $ij \in E$ and $f_{ij} < u_{ij}$, or if $ji \in E$ and $f_{ji} > 0$. These two types of arcs will be called *forward* and *backward arcs*. The *residual capacity* of a forward arc ij is $u^f_{ij} := c_{ij} - f_{ij}$, and for a backward arc ij it is $u^f_{ij} := f_{ji}$. By sending δ units of flow on an arc ij in the residual graph we mean increasing f_{ij} by δ if ij is a forward arc and decreasing f_{ji} by δ if ij is a backward arc. Sending δ units of flow on a path/cycle in the residual graph means sending δ units on every arc of the path/cycle.

The Ford-Fulkerson algorithm To prove the MFMC theorem, Ford and Fulkerson showed that as long as f is not a maximum $s-t$ flow, there exists an $s-t$ path in the residual graph G_f. Consider any such $s-t$ path P, and let δ denote the minimum residual capacity along P. Then, we obtain a feasible flow f' by sending δ units of flow on P; this increases the flow value by δ. The Ford-Fulkerson algorithm starts from $f = 0$, and keeps repeating the above step as long as the current flow is not maximum.

If the capacities c are integers, then the algorithm maintains an integer flow throughout, and the flow value increases by one at every step. Hence, the number of iterations will be bounded by $u(\delta^+(s))$.

However, this running time is only pseudopolynomial. This can be verified on the example with $V = \{s,t,i,j\}$, $E = \{(s,i),(s,j),(i,t),(j,t),(i,j)$ with $u_{si} = u_{sj} = u_{it} = u_{jt} = U$ for a large integer U, and $u_{ij} = 1$. The algorithm may repeatedly use the paths $s-i-j-t$ and $s-j-i-t$ to send 1 unit of flow each time, giving a total of $2U$ iterations.

The Edmonds-Karp-Dinits algorithm The first strongly polynomial algorithms for the maximum flow problem was given by Dinits [2], and independently by Edmonds and Karp [4]. These amount to a special selection rule in the Ford-Fulkerson algorithm: always pick a *shortest* $s-t$ path in the residual graph. In the previous example, this would terminate in only 2 augmentations.

In general, they showed that the algorithm always terminates within $O(mn)$ path augmentations; see e.g. [1, Chapter 7.4] for the proof; this yields a total running time $O(nm^2)$ that can be further improved to $O(n^2 m)$.

There is a large literature of strongly polynomial maximum flow algorithms, see [1]. The current fastest running time $O(mn)$ was given by Orlin in 2013 [14], building on several previous results. If the capacities are small, there are much faster weakly polynomial algorithms available; we do not discuss them here.

1.3 The minimum-cost flow problem

Consider a capacitated network as in the previous section, with arc costs $c : E \to \mathbb{Q}$ and node demands $b : V \to \mathbb{Q}$ such that $b(V) = 0$. We say that $f : E \to \mathbb{R}$ is a b-flow if $\nabla f = b$. The minimum-cost flow problem asks for minimum c-cost of a b-flow f satisfying the capacity constraints:

$$\begin{aligned} \min\ & c^\top f \\ \nabla f_i &= b_i \quad \forall i \in V \\ 0 \leq f &\leq u. \end{aligned} \tag{1.5}$$

To simplify the discussion, we will assume that all capacities are infinite, that is, we only have the nonnegativity constraint $f \geq 0$. This is without loss of generality (see homework exercise): every capacitated flow problem on n nodes and m arcs can be replaced by an equivalent uncapacitated instance on $n + m$ nodes and $2m$ arcs. We will focus on the uncapacitated form:

$$\begin{aligned} \min\ & c^\top f \\ \nabla f_i &= b_i \quad \forall i \in V \\ 0 &\leq f. \end{aligned} \tag{1.6}$$

The dual problem can be written as:

$$\begin{aligned} \max\ & b^\top \pi \\ \pi_j - \pi_i &\leq c_{ij} \quad \forall ij \in E \end{aligned} \tag{1.7}$$

Node potentials and negative cycles Given a vector $\pi : V \to \mathbb{R}$ (often called a *node potential*) we can define the modified cost function

$$c^\pi_{ij} := c_{ij} + \pi_i - \pi_j.$$

The following is a simple observation:

Claim 1.2. *For any cycle $C \subseteq E$, and any $\pi : V \to \mathbb{R}$, we have $c^\pi(C) = c(C)$.*

Node potentials provide a well-known characterization on whether a graph contains a negative cycle:

Lemma 1.3. *Given a directed graph $G = (V, E)$ and arc costs $c : E \to \mathbb{R}$, every cycle has a nonnegative cost if and only if there exists a node potential π such that $c^\pi \geq 0$.*

Proof. The existence of such a potential clearly implies that no negative cycle may exists using the previous claim. Conversely, assume there exists no negative cost cycle. For every node $i \in V$, let us define π to be the minimum-cost of a walk ending at i. Since all cycles have nonnegative costs, this is well-defined: the optimal walk P_i is a path. The definition implies that $\pi_j \leq c_{ij} + \pi_i$, that is, $c^\pi \geq 0$. □

There are several simple strongly polynomial algorithms for deciding whether a given graph contains a negative cost cycle. A simple (but not the most efficient) label correcting algorithm inspired by the above proof is the following. For $k = 0, 1, 2, \ldots, n$, and $i \in V$, we let $\pi_i^{(k)}$ denote the length of the minimum-cost walk of $\leq k$ arcs ending in i. We start by setting $\pi_i^{(0)} = 0$ for all $i \in V$, and in every iteration, we update

$$\pi_i^{(k+1)} := \min\{\pi_i^{(k)}, \min_{j \in \delta^-(i)} \pi_j^{(k)} + c_{ji}\}.$$

Let us compute these values for $k \leq n$. If $\pi^{(n)} = \pi^{(n-1)}$, then $\pi = \pi^{(n)}$ satisfies $c^\pi \geq 0$. Otherwise, we have a $j \in V$ with $\pi_j^{(n)} < \pi_j^{(n-1)}$; thus, there is a walk of length n ending at j that is shorter than any path ending at j, implying the existence of a negative cycle that can be easily recovered from the sequence of values computed during the algorithm.

Feasibility Deciding the feasibility of (1.6) reduces to a maximum flow computation (see exercise). Hoffman's circulation theorem provides a particularly simple characterization of feasibility.

Theorem 1.4. *The program (1.6) is feasible if and only if there exists no set $S \subseteq V$ with $\delta^-(S) = \emptyset$ and $b(S) > 0$.*

The *only if* direction is immediate: we have

$$f(\delta^-(S)) - f(\delta^+(S)) = \sum_{i \in S} \nabla f_i.$$

For a b-flow f and a set S with no incoming arcs, we see that $0 \geq \sum_{i \in S} \nabla f_i = b(S)$ must hold.

Flow decomposition For an arc set $H \subseteq E$, we let $\chi^H : E \to \mathbb{R}_+$ be the indicator flow defined as $\chi^H_e = 1$ if $e \in H$ and $\chi^H_e = 0$ otherwise.

Theorem 1.5. *Consider a vector $f : E \to \mathbb{R}_+$. Then, we can write*

$$f = \sum_{\ell=1}^{k} \lambda_\ell \chi^{P_\ell} + \sum_{\ell=1}^{k'} \mu_\ell \chi^{C_\ell},$$

where $k + k' \leq |\mathrm{supp}(f)|$, all P_ℓ's are paths, all C_ℓ's are cycles, and $\sum_{\ell=1}^{k} \lambda_\ell = \|\nabla f\|_1/2$. Further, if $k' = 0$, then $\|f\|_\infty \leq \|\nabla f\|_1/2$. Moreover, such a decomposition can be found in strongly polynomial time.

Note that since $\sum_{i \in V} \nabla f_i = 0$, we have $\|\nabla f\|_1 = 2 \sum_{i \in V : \nabla f_i > 0} \nabla f_i$.

Proof. The proof is by induction on $|\mathrm{supp}(f)|$. First, assume that $\nabla f = 0$. Let us select any ij with $f_{ij} > 0$; let $i_1 := i$ and $i_2 := j$. Since $\nabla f_{i_2} = 0$, there must be an outgoing arc $i_2 i_3$ with $f_{i_2 i_3} > 0$. Similarly, we construct a walk $i_1 i_2 i_3 \ldots$. The first time a vertex occurs for the second time, we have identified a cycle C_1 in $\mathrm{supp}(f)$. We let $\mu_1 := \min_{e \in C_1} f_e$ and $f' := f - \mu_1 \chi^{C_1}$. Thus, $|\mathrm{supp}(f')| < |\mathrm{supp}(f)|$, and we can use induction to obtain the decomposition.

If $\nabla f \neq 0$, then we use the same argument, starting from an i_1 such that $\nabla f_{i_1} < 0$; thus, there must be an outgoing arc $i_1 i_2 \in \mathrm{supp}(f)$. Using the same procedure as above, we either find a closed walk, or get stuck in a vertex i_t such that there are no outgoing arcs in $\mathrm{supp}(f)$, meaning $\nabla f_{i_t} > 0$. In this case, we have found a path $P_1 = i_1 i_2 \ldots i_t$, and set $\lambda_1 := \min_{e \in P_1} f_e$. We let $f' := f - \mu_1 \chi^{P_1}$ and use induction. Note that $\|\nabla f'\|_1 = \|\nabla f\|_1 - 2\lambda_1$.

In the absence of cycles in the decomposition, the bound $\|f\|_\infty \leq \|\nabla f\|_1/2 = \sum_{i=1}^{k} \mu_i$ is immediate. The above argument can be easily turned into an efficient algorithm. □

Interpreting the dual Let us now consider the dual program (1.7). For a feasible solution π, the constraints assert that $c^\pi \geq 0$. Note that it implies in particular that there are no negative cycles in G. In case there is a negative cost cycle, the primal (1.6) is unbounded. Complementary slackness yields that $c^\pi_{ij} = 0$ must hold whenever $f_{ij} > 0$.

In the uncapacitated formulation (1.6), the residual graph $G_f = (V, E_f)$ is of a simple form, namely $E_f = E \cup \overleftarrow{\mathrm{supp}(f)}$. For a reverse arc ij we define the cost as $c_{ij} := -c_{ji}$. With this convention, Lemma 1.3 yields the following:

Lemma 1.6. *The nonnegative b-flow f is optimal to (1.6) if and only if the residual graph G_f contains no negative cycles.*

Note that one direction is obvious: if G_f includes a negative cycle, then we can obtain a better solution by sending a small amount of flow on this cycle. Using this optimality condition, we can obtain two natural algorithm schemes for the minimum-cost flow problem.

1.3.1 The cycle cancelling algorithm

In the cycle cancelling algorithm, we start with a feasible nonnegative b-flow f. A b-flow f can be obtained via a maximum flow computation (as mentioned above). We refine this slightly to obtain a good starting solution: we compute the flow decomposition of f as in Theorem 1.5. Consider the cycles in the decomposition. If any of them is negative, then (1.6) is unbounded (why?). Otherwise, we remove all cycle flows; the resulting flow will be the starting flow for the algorithm.

In every iteration, we check whether E_f contains a negative cycle. If not, then we can conclude that the current f is optimal. Otherwise, we select a negative cycle $C \subseteq E_f$, and let δ to be the smallest residual capacity of any arc on C. We change f by sending δ units of flow around C. Clearly, this changes the cost $c^\top f$ by $\delta c(C) < 0$.

Algorithm 1 Cycle Cancelling Algorithm

Input: Graph $G = (V, E)$, node demands $b : V \to \mathbb{Q}$, with $b(V) = 0$, costs $c : E \to \mathbb{Q}$.
Output: A minimum-cost b-flow.

```
1: Compute an initial $b$-flow $f$. Remove all cycles from the decomposition.
2: **repeat**
3:     Decide if $E_f$ contains a negative cost cycle $C$.
4:     **if** no negative cycle exists **then return** $f$.
5:     **else** $\delta \leftarrow \min_{e \in C} u_e^f$. Send $\delta$ units of flow around $C$.
6: **until** $E_f$ contains no negative cycles.
```

Theorem 1.7. *Assume all node demands and arc costs are integers, with $|c_{ij}| \le C$ for all $ij \in E$. Then the cycle cancelling algorithm terminates in $mC\|b\|_1$ iterations.*

Proof. Using Theorem 1.5, we see that the initial flow has $\|f\|_\infty \le \|b\|_1/2$; therefore, its cost is at most $mC\|b\|_1/2$. Similarly, we see that there exists an optimal flow with a decomposition containing no cycles; therefore the optimum value can be lower bounded by $-mC\|b\|_1/2$.

For integer b, we can assume that the initial flow is integer, and it remains integer throughout. The cost decreases by at least one in every iteration; the claimed bound follows. □

This bound is only pseudopolynomial. There are examples showing that the generic cycle cancelling algorithm can run for an exponential number of iterations. For irrational input, it may not even terminate.

1.3.2 The successive shortest path algorithm

The cycle cancelling algorithm proceeds by constructing a sequence of feasible solutions to (1.6) of decreasing objective value. We now exhibit an algorithm that maintains instead feasible dual solutions to (1.7), but proceeds with infeasible solutions to (1.6). Throughout, we maintain a nonnegative flow $f \ge 0$ that may violate the node demand constraints. If $\nabla f_i > b_i$, then i is a *excess node*, and if $\nabla f_i < b_i$, then i is a *deficient node*. The key invariant maintained throughout is the optimality condition in Lemma 1.6:

$$\text{The residual graph } G_f \text{ contains no negative cost cycles.} \tag{1.8}$$

We initialize $f = 0$; thus, $G_f = E$. If (1.8) is violated, then we can conclude that the instance is unbounded or infeasible. Otherwise, in every iteration, we select a shortest path P in G_f from an excess node r to a deficient node q. We send δ units of flow along P where δ is the minimum of the excess of r, the deficiency of q, and the minimum residual capacity of the arcs in P. Such a shortest path can be found via Dijkstra's algorithm; we describe a simple (but not the most efficient) variant in Algorithm 2. Given a node potential π, we say that an arc $e \in E_f$ is *tight* if $c_e^\pi = 0$. The key observation is the following:

Lemma 1.8. *Throughout the successive shortest path algorithm, the invariant (1.8) is maintained.*

Algorithm 2 Successive Shortest Path Algorithm

Input: Graph $G = (V, E)$, node demands $b : V \to \mathbb{Q}$, with $b(V) = 0$, costs $c : E \to \mathbb{Q}$.

Output: A minimum-cost b-flow, or the conclusion that (1.6) is infeasible or unbounded.

```
 1: $f_e \leftarrow 0$ for all $e \in E$.
 2: Compute $\pi : V \to \mathbb{Q}$ such that $c^\pi_e \geq 0$ for all $e \in E$.
 3: if no such $\pi$ exists then return a negative cycle in $E$.
 4: repeat
 5:     $R \leftarrow \{i \in V : \nabla f_i > b_i\}$.                                  ▷ excess nodes
 6:     $Q \leftarrow \{i \in V : \nabla f_i < b_i\}$.                                  ▷ deficient nodes
 7:     $S \leftarrow \{i \in V : \exists$ a tight path from $i$ to $Q$ in $E_f\}$.
 8:     while $S \cap R = \emptyset$ do
 9:         $\varepsilon \leftarrow \min\{c^\pi_{ij} : ij \in E_f, i \notin S, j \in S\}$.
10:         if $\varepsilon = \infty$ then return infeasible instance.
11:         else
12:             $\pi_i \leftarrow \pi_i + \varepsilon$ for all $i \in S$.
13:             $S \leftarrow \{i \in V : \exists$a tight path from $i$ to $Q$ in $E_f\}$.
14:     Select a tight path $P$ from a node $r \in R$ to a node in $q \in Q$.
15:     $\delta \leftarrow \min\left\{\nabla f_r - b_r, b_q - \nabla f_q, \min_{e\in P} u^f_e\right\}$. Send $\delta$ units of flow on $P$.
16: until $f$ is a $b$-flow.
17: return $f$.
```

Proof. This can be verified by maintaining the potential π as in Algorithm 2 that satisfies $c^\pi_e \geq 0$ for all $e \in E_f$. Whenever a shortest path is found, all its arcs are tight: $c^\pi_e = 0$ for all $e \in P$. When we send flow along P, the only new arcs appearing in E_f can be reverse arcs of some arcs in P, namely, if $ij \in P \cap E$, and $f_{ij} = 0$ before sending the flow, then ji enters E_f as a new reverse arc. However, in this case, $c^\pi_{ij} = 0$, and therefore $c^\pi_{ji} = -c^\pi_{ij} = 0$. Therefore, the same π verifies (1.8) after the flow update. □

Theorem 1.9. *If all node demands are integers, then the Successive Shortest Path Algorithm terminates in $\|b\|_1/2$ iterations. The running time of the algorithm is $O(\|b\|_1(m + n \log n))$*

Proof. Let

$$\mathrm{Ex}(f) := \sum_{i\in V} (\nabla f_i - b_i)^+ = \|\nabla f - b\|_1/2.$$

denote the total excess of f. Initially, $\mathrm{Ex}(f) = \|b\|_1/2$. The flow remains integer throughout, and in every path augmentation, $\mathrm{Ex}(f)$ decreases by $\delta \geq 1$. The bound on the number of iterations follows. Using the efficient implementation of Dijkstra's algorithm with Fibonacci heaps (which is different from the variant described in Algorithm 2), every iteration takes $O(m + n \log n)$ time.

Finally, we show that if no shortest path can be found, that is, $\varepsilon = \infty$ at some point of the algorithm, then the set S testifies that no b-flow may exist. Indeed, we have $b_i \geq \nabla f_i$ for every $i \in S$, with strict inequality for all $i \in Q$. Also, $f_{ij} = 0$ on all arcs entering and leaving S. Therefore $b(S) > \sum_{i\in S} \nabla f_i = 0$. The set S has positive total demand, but there are no arcs entering S, showing that no b-flow exists according to Theorem 1.4. □

1.4 The Goldberg-Tarjan algorithm

The cycle cancelling algorithm allows for selecting arbitrary cycles. Goldberg and Tarjan [6] have found a simple and elegant selection rule that ensures *strongly polynomial* termination. We now introduce the algorithm and provide the weakly polynomial analysis. The strongly polynomial bound will be shown in Section 2.1.1.

The *mean cost* of a cycle $C \in \overset{\leftrightarrow}{E}$ is defined as $c(C)/|C|$. The Goldberg-Tarjan (GT) algorithm is the special implementation of the cycle cancelling algorithm where we always select a cycle C in E_f that has the *minimum-mean cost.* Let us now discuss how this cycle can be found.

1.4.1 Finding a minimum-mean cycle

Consider the following pair of primal and dual LPs:

$$\begin{aligned} \min\ & c^\top x & \qquad \max\ & -\varepsilon \\ \nabla x_i &= 0 \quad \forall i \in V & \pi_j - \pi_i - \varepsilon &\le c_{ij} \quad \forall ij \in E \\ \sum_{e \in E} x_e &= 1 & & \\ x &\ge 0. & & \end{aligned} \tag{1.9}$$

Lemma 1.10. *The support of a basic optimal solution to the primal LP in (1.9) gives a minimum-mean cycle.*

Proof. For every cycle $C \subseteq E$, setting $x_e := 1/|C|$ for $e \in C$ and $x_e := 0$ otherwise gives a solution to the primal LP of cost $c(C)/|C|$. Consider the optimal primal solution in (1.9), which is a circulation (0-flow). As in Theorem 1.5, this can be written as a nonnegative combination of incidence vectors of cycles. Clearly, a basic feasible solution must be supported on a single cycle C. Thus, a basic optimal solution corresponds to a minimum-mean cycle. □

The dual program in (1.9) also gives a very useful interpretation. The minimum cycle mean will be $-\varepsilon$, where ε is the smallest value such that there exists a potential π satisfying $c_e^\pi \ge -\varepsilon$ for all $e \in E$. By Lemma 1.3, this is equivalent to that the cost function c' defined by $c'_e = c_e + \varepsilon$ does not contain any negative cycles.

The LPs in (1.9) can be solved in strongly polynomial time, by a modification of the negative cycle detection algorithm, see the exercises.

1.4.2 The weakly polynomial analysis

The weakly polynomial running time guarantee follows from the next lemma. For a given b-flow f, let $\varepsilon(f)$ denote the absolute value of the minimum-mean cycle value, that is, the negative of the optimum value in (1.9).

Lemma 1.11. *The value of $\varepsilon(f)$ is monotone decreasing throughout the GT algorithm. Let f be the flow at a certain iteration and f' the flow m iterations later. Then, $\varepsilon(f') \le \left(1 - \frac{1}{n}\right)\varepsilon(f)$. Consequently, within nm iterations, $\varepsilon(f)$ decreases at least by a factor 2.*

Proof. For the first part, let f be the flow at a certain iteration, and let π be the dual optimal solution to (1.9) for the graph $G_f = (V, E_f)$. Thus, $c_e^\pi \ge -\varepsilon$ for every $e \in E_f$, and by complementary slackness, equality holds for all arcs on the cycle C selected in this iteration. The new arcs entering E_f can be arcs on $\overleftarrow{C}$; but any such new arc will have $c_e^\pi = \varepsilon$. Consequently, the potential π shows that the cycle mean is $\ge -\varepsilon$ after the iteration.

For the second part, let $\bar{f} := f^{(t)}$ denote the flow at the t-th iteration of the GT algorithm, and let $\bar{\pi}$ denote a dual optimal solution to (1.9) for $G_{\bar f}$. Let $\varepsilon := \varepsilon(\bar f)$.

Let $\bar{E} \subseteq E_{\bar f}$ denote the set of arcs such that $c_e^{\bar\pi} < 0$. Clearly, $-\varepsilon \le c_e^{\bar\pi}$ for every $e \in E_f$, and using complementary slackness, the t-th iteration selects a cycle $\bar{C}$ with all arcs $e \in \bar{C}$ satisfying $c_e^{\bar\pi} = -\varepsilon$, in particular, $e \in \bar{E}$.

Let $t' > t$ be the first iteration such that the cycle $C_{t'}$ selected at iteration t' uses an arc outside $\bar{E}$. We claim that $t' \le t + m$, and that $\varepsilon(f^{(t')}) \le \left(1 - \frac{1}{n}\right)\varepsilon(f^{(t)})$.

Consider any iteration τ between $t \le \tau < t'$. We claim that in this iteration, no arc new $e \in \overleftrightarrow{E}$ is added to E_f such that $c_e^{\bar\pi} < 0$, and at least one such arc is removed. Let C_τ be the cycle cancelled in this iteration. By the choice of t', every $e \in C_\tau$ has $c_e^{\bar\pi} < 0$, and any new arcs entering the residual graph are in $\overleftarrow{C}_\tau$, thus, they have a positive reduced cost. Further, at least one negative reduced cost are is removed from E_f, namely those that get saturated on the cycle C_τ.

The above argument also gives $c^{\bar{\pi}}_e \geq -\varepsilon$ for every arc e in the residual graph at iteration t'; in particular, this holds for all arcs of $C_{t'}$. Further, at least one arc $e' \in C_{t'}$ satisfies $c^{\bar{\pi}}_{e'} \geq 0$. Consequently, the mean cost is

$$\frac{c(C_{t'})}{|C_{t'}|} \geq -\frac{(|C_{t'}|-1)\varepsilon}{|C_{t'}|} \geq -\left(1-\frac{1}{n}\right)\varepsilon.$$

□

Theorem 1.12. *For an integer cost function $c : E \to \mathbb{Z}$, with $c_e \geq -C$ for all $e \in E$, the GT algorithm terminates in $O(nm\log(nC))$ iterations.*

Proof. The mean of any cycle is $\geq -C$, therefore $\varepsilon(f) \leq C$ at the initialization. According to Lemma 1.11, within $O(nm\log(nC))$ the value of ε decreases below $1/n$. Since all costs are integer, this means that the cycle mean at this point is nonnegative. □

1.5 The Edmonds-Karp scaling algorithm

The running time of the successive shortest paths (SSP) algorithm is pseudopolynomial: it depends on $\|b\|_1$ for integer demands b. The first weakly polynomial algorithm for the minimum-cost flow problem was given by Edmonds and Karp in 1972 [4]. This introduced the influential idea of *scaling* to turn the SSP algorithm into a polynomial algorithm.

The main caveat of the SSP algorithm is that the minimum residual capacity of a shortest path could be very small. The variant described below will be a variant of the SSP algorithm that always augments by amounts that are commensurate with the excess $\mathrm{Ex}(f)$. The algorithm is guided by a scaling factor Δ, which will be selected as decreasing powers of 2.

The algorithm is divided into Δ phases. In a Δ-phase, the deficiency of every node is at most 2Δ, and there are some nodes with deficiency at least Δ. We let Q denote the set of such nodes; if this is empty, we move on to the next phase by dividing Δ by 2.

Otherwise, we find a shortest path from an excess node to a node in Q as in the SSP algorithm, and send Δ units of flow along this path.

Algorithm 3 Edmonds-Karp Scaling Algorithm

Input: Graph $G = (V, E)$, node demands $b : V \to \mathbb{Z}$, with $b(V) = 0$, costs $c : E \to \mathbb{Q}$.
Output: A minimum-cost b-flow, or the conclusion that (1.6) is infeasible or unbounded.

```
 1: $f_e \leftarrow 0$ for all $e \in E$.
 2: $\Delta \leftarrow 2^k$ for the smallest $k$ such that $2\Delta \geq |b_i|$ for all $i \in V$.
 3: Compute $\pi : V \to \mathbb{Q}$ such that $c^\pi_e \geq 0$ for all $e \in E$.
 4: if no such $\pi$ exists then return a negative cycle in $E$.
 5: repeat
 6:     $R \leftarrow \{i \in V : \nabla f_i > b_i\}$.                        ▷ excess nodes
 7:     $Q \leftarrow \{i \in V : \nabla f_i \leq b_i - \Delta\}$.          ▷ nodes with large deficiency
 8:     while $Q = \emptyset$ do $\Delta \leftarrow \Delta/2$ ; recompute $Q$.      ▷ New $\Delta$-phase.
 9:     Update $\pi$ and find a tight path $P$ from a node $r \in R$ to a node in $q \in Q$ or conclude
    infeasibility as in Algorithm 2.
10:     Send $\Delta$ units of flow on $P$.
11: until $f$ is a $b$-flow.
12: return $f$.
```

Theorem 1.13. *Assume all node demands are integers of absolute value $\leq B$. Then the Edmonds-Karp scaling algorithm finds a minimum-cost flow or concludes that the problem is infeasible or unbounded in time $O(n(m + n\log n)\log B)$.*

Proof. First, let us show that we can always send Δ units of flow along the path P, that is, we do not violate the residual capacities. This follows by showing that in the Δ phase, the flow on every arc is an integer multiple of Δ. The proof is by induction: this clearly holds in the first phase, and is maintained when moving to the next phase, because we select the new value of Δ as $\Delta/2$. Therefore, the flow f remains nonnegative throughout.

The bound $b_i - \nabla f_i \le 2\Delta$ is also maintained. Initially, this is how Δ was selected. A path augmentation could turn the starting excess node into a deficient node, but in this case the deficiency will be $< \Delta$. We decrease Δ once $b_i - \nabla f_i < \Delta$ for all i; thus, $b_i - \nabla f_i \le 2\Delta$ for the new value. Let us consider the potential

$$\Xi(f) := \sum_{i \in V} \max\left\{ \left\lfloor \frac{b_i - \nabla f_i}{\Delta} \right\rfloor, 0 \right\}.$$

By the above, $\Xi(f) \le 2n$ throughout. In every path augmentation between r and q, ∇f_i remains unchanged for all nodes different from r and q. The term corresponding to r is 0 both before and after the path augmentation, and the term for q is initially ≥ 1 and decreases by 1.

Thus, within the same Δ phase, $\Xi(f)$ cannot increase but it decreases at every path augmentation. Therefore the number of path augmentations is at most $2n$ in every Δ-phase.

At the end of the phase $\Delta = 1$, the algorithm terminates with an optimal solution, since the flow and b are integer, and $Q = \emptyset$. Thus, the total number of phases is $\log_2 B$, and the total number of path augmentations in $O(n \log B)$. $\square$

This algorithm is inherently *not* strongly polynomial: Δ always decreases by a factor 2, hence we need to go through $\log B$ phases. (Instead of always halving, we could immediately jump to the smallest power of 2 such that $Q \neq \emptyset$, but this is does not suffice from strong polynomiality.)

Chapter 2

Techniques for strongly polynomial algorithms

We now exhibit a general approach that enables to turn weakly polynomial or even pseudopolynomial flow algorithms into strongly polynomial ones. The main progress is measured by showing that, within a strongly polynomial number of iterations, we can learn the value of a variable in every optimal solution to (1.6), or that a constraint must be binding in every optimal solution to (1.7). Thus, we can reduce the size of the network by contracting or deleting arcs.

2.1 Primal variable fixing

Lemma 2.1 (Tardos)**.** *Consider a b-flow f, labels $\pi : V \to \mathbb{R}$ and $\varepsilon > 0$ such that $c^\pi_e \geq -\varepsilon$ for all $e \in E_f$. If for some $e \in E$, $c^\pi_e > n\varepsilon$, then $f^*_e = f_e$ must hold for every optimal solution f^* to (1.6).*

Note that for $\varepsilon = 0$, f and π are primal and dual optimal solutions, and the statement follows by complementary slackness.

Proof. Let f^* be any optimal solution, and let us define the flow $h : \overset{\leftrightarrow}{E} \to \mathbb{R}$ as

$$h_{ij} := \begin{cases} f^*_{ij} - f_{ij}, & \text{if } ij \in E, f_{ij} \leq f^*_{ij} \\ f_{ji} - f^*_{ji}, & \text{if } ji \in E, f_{ji} > f^*_{ji}. \end{cases}$$

Since f and f^* are both b-flows, then h must be a circulation (that is, a 0-flow). If $f^*_e \neq f_e$, then $h_e > 0$ or $h_{\overleftarrow{e}} > 0$; let us assume $h_e > 0$; the case $h_{\overleftarrow{e}} > 0$ follows analogously. Thus, there must be a cycle C in $\mathrm{supp}(h)$ containing e. Then,

$$c(C) = c^\pi(C) \geq c^\pi_e + \sum_{e' \in C} c^\pi_{e'} > n\varepsilon - (n-1)\varepsilon = \varepsilon > 0.$$

Let $\overleftarrow{C}$ denote the reverse cycle. Then, $c(\overleftarrow{C}) = -c(C) < 0$. However, $\overleftarrow{C} \subseteq E_{f^*}$, giving a contradiction to Lemma 1.6. □

Once we can infer the value of an arc ij in the optimal solution, we can delete ij from the graph, and modify the demands b_i and b_j accordingly. An optimal solution to the modified instance immediately extends to an optimal solution to the original instance.

Tardos's algorithm proceeds as follows: every major iteration deletes at least one arc from the graph. To achieve this, we can construct a rounded version of the problem where all costs are small integers. Every algorithm that has even pseudopolynomial dependence on the costs (and does not depend on the capacities) can solve such an instance optimally. One example is the out-of-kilter algorithm, see [1, Section 9.9]. From the output, one can identify an arc as in Lemma 2.1.

Instead of giving the details, we show how the number of iterations in the GT algorithm is strongly polynomial using a slight generalization of this lemma.

2.1.1 The Goldberg-Tarjan algorithm

We formulate an extension of Lemma 2.1. Recall the notion of $\varepsilon(g)$ from Section 1.4.

Lemma 2.2. *Consider two b-flows f and g, $\pi : V \to \mathbb{R}$ and $\varepsilon > 0$ such that $c^\pi_e \geq -\varepsilon$ for all $e \in E_f$. If for some $e \in E$, $c^\pi_e > n\varepsilon + n\varepsilon(g)$, then $f_e = g_e$ must hold.*

Theorem 2.3. *For arbitrary (even real valued) costs, the GT algorithm terminates in $O(nm^2 \log n)$ iterations.*

Proof. Let $T = nm(\lceil \log n\rceil + 1)$. At iteration t, let $E_t \subseteq E$ denote the set of arcs e such that f_e does not change in any subsequent iteration of the GT algorithm. We show that if $t' \geq t + T$, then $E_{t'} \supsetneq E_t$. This implies the bound, since the total number of arcs is m.

Let f and f' be the flow at times t and t' respectively, and π and π' the corresponding potentials. Let $\varepsilon := \varepsilon(f)$ and $\varepsilon' := \varepsilon(f')$. According to Lemma 1.11, $\varepsilon' \leq \varepsilon/2n$. Consider the cycle C that was cancelled in iteration t. This has

$$-|C|\varepsilon = c(C) = c^{\pi'}(C)$$

Consequently, one of the arcs $ij \in C$ has $c^{\pi'}_{ij} \leq -\varepsilon \leq 2n\varepsilon'$. Note that $ij \in E_f \setminus E_{f'}$, meaning that $ji \in E$ and $f'_{ji} = 0$. Further, $c^{\pi'}_{ji} > 2n\varepsilon'$.

Let f'' be the flow at any iteration $t'' > t'$. Then, $\varepsilon(f'') \leq \varepsilon'$. Lemma 2.2 shows that $f''_{ji} = f'_{ji}$. Consequently, $ji \in E_{t'}$, but $ji \notin E_t$, completing the proof. □

2.2 Dual constraint fixing

Recall the notation $\mathrm{Ex}(f) = \sum_{i \in V} (\nabla f_i - b_i)^+$ for the total excess.

Lemma 2.4. *Consider an $f \geq 0$ such that G_f does not contain any negative cycles, and an arc $k\ell \in E$ such that $f_{k\ell} > \mathrm{Ex}(f)$. Then, $c_{k\ell} = \pi_\ell - \pi_k$ must hold for every optimal solution π to (1.7).*

Proof. Let us select an optimal solution f^* to (1.6) such that $\|f - f^*\|_1$ is minimal. We show that $\|f - f^*\|_\infty < Ex(f) < f_{k\ell}$. Therefore, $f^*_{k\ell} > 0$, and the claim follows by complementary slackness.

Let us define the flow $h : \overleftrightarrow{E} \to \mathbb{R}$ as in the proof of Lemma 2.1, namely,

$$h_{ij} := \begin{cases} f^*_{ij} - f_{ij}, & \text{if } ij \in E, f_{ij} \leq f^*_{ij} \\ f_{ji} - f^*_{ji}, & \text{if } ji \in E, f_{ji} > f^*_{ji}. \end{cases}$$

We claim that $\mathrm{supp}(h)$ does not contain any cycles. Indeed, if $C \in \mathrm{supp}(h)$, then $C \in E_f$ and the reverse cycle $\overleftarrow{C} \in E_{f^*}$. The assumption that G_f does not contain negative cycles and the optimality of f^* implies that $c(C) = 0$. Then, we can change f^* by subtracting $\epsilon\chi^C$ for some $\varepsilon > 0$ to obtain another optimal solution that is closer to f in 1-norm, a contradiction to the choice of f^*.

Let us apply the flow decomposition Theorem 1.5 to h. By the above, the decomposition contains only paths, and therefore $\|h\|_\infty \leq \|\nabla h\|_1/2 = \mathrm{Ex}(f)$ follows. Thus, $|f_{k\ell} - f^*_{k\ell}| < \mathrm{Ex}(f) < f_{k\ell}$, implying $f^*_{k\ell} > 0$. □

Contraction How can we use the information that for some $k\ell \in E$, $c_{k\ell} = \pi_\ell - \pi_k$ is guaranteed for every dual optimal solution to (1.7)? This means that the optimum solution in (1.6) does not change if we remove the nonnegativity constraint for the variable $f_{k\ell}$. Thus, we could contract the nodes k and ℓ into a single node with demand $b_k + b_\ell$; every feasible primal solution in the contracted instance extends to the original graph (with $f_{k\ell}$ possibly negative).

This contraction can be defined as follows. Let k also denote the image of k and ℓ in the contracted instance; we set the demand $b'_k := b_k + b_\ell$. We leave the incoming and outgoing arcs incident to k unchanged. For every $\ell j \in E$ in the original graph, we add an arc $kj \in E$ with cost $c'_{kj} := c_{k\ell} + c_{\ell j}$; for every arc $j\ell \in E$, we add $jk \in E$ with cost $c'_{jk} := c_{j\ell} - c_{k\ell}$. If parallel arcs are created, we only keep the one of lower cost.

Given a dual optimal solution in the contracted instance, we can straightforwardly extend it to the original instance. Then, we can also obtain a primal optimal solution in strongly polynomial time.

Lemma 2.5. *Assume we are given a dual optimal solution to (1.7). Then, a primal optimal solution to (1.6) can be obtained by a single maximum flow computation.*

2.2.1 Orlin's strongly polynomial algorithm

Using Lemma 2.4, Orlin [13] showed that the Edmonds-Karp algorithm can be turned into a strongly polynomial one. The running time in $O(n \log n(m + n \log n))$ for the uncapacitated form (1.6), and $O(m \log n(m + n \log n))$ for the capacitated form (1.5). This is still the best running time for the minimum-cost flow problem. Here, we present a simpler and less efficient variant; the full algorithm is described in [1, Section 10.2].

The overall scheme Whereas the algorithm uses primal flow augmentations, the true goal is to obtain a dual optimal solution. We repeatedly run the Edmonds-Karp algorithm for a strongly polynomial number of iterations, until we can apply Lemma 2.4 to obtain an arc $k\ell \in E$ that can be contracted. Then, we restart the algorithm in the contracted instance. The overall algorithm terminates once all node demands become $b_i = 0$, in which case $f = 0$ is an optimal solution. Thus, any potential with $c^\pi \geq 0$ is an optimal dual solution. We blow this back to an optimal dual solution in the original graph. Then, the optimal primal solution to (1.6) in the original graph can be obtained using a single maximum flow computation as in Lemma 2.5.

Abundant arcs from the Edmonds-Karp algorithm Let us now consider the Edmonds-Karp algorithm. Recall from the proof of Theorem 1.13 that in the Δ-phase, the deficiency of any node is $\leq 2\Delta$. Therefore, the total deficiency is $\leq 2n\Delta$, and thus $\mathrm{Ex}(f) \leq 2n\Delta$ (since the total excess equals the total deficiency).

Hence, if in the Δ-phase we can find an arc $k\ell \in E$ with $f_{k\ell} > 2n\Delta$, then Lemma 2.4 is applicable. We call such an arc an *abundant arc*. We show the following.

Lemma 2.6. *Within the first $O(\log n)$ scaling phases of the Edmonds-Karp algorithm, the flow must contain an abundant arc.*

Proof. Let $B := \max_{i \in V} |b_i|$, and let $s \in V$ be the node giving the maximum value. The initial value of Δ satisfies $B/2 \leq \Delta \leq B$. Hence, in $O(\log n)$ phases, we obtain a value of Δ such that $2n(n+1)\Delta < |b_s|$.

First, assume that $b_s > 0$. Then, since $\nabla f_s \geq b_s - 2\Delta$, we must have $\nabla f_s > 2n^2\Delta$. Then, one of the incoming arcs $js \in \delta^-(s)$ must carry at least $f_{js} > 2n\Delta$, that is, js is abundant.

For the case $b_s < 0$, we know that $\nabla f_s \leq b_s + \mathrm{Ex}(f) \leq b_s + 2n\Delta$, that is, $\nabla f_s < -2n^2\Delta$. Now, one of the outgoing arcs $sj \in \delta^+(s)$, must be abundant. □

Algorithm 4 Orlin's Algorithm

Input: Graph $G = (V, E)$, node demands $b : V \to \mathbb{Z}$, with $b(V) = 0$, costs $c : E \to \mathbb{Q}$.
Output: A minimum-cost b-flow, or the conclusion that (1.6) is infeasible or unbounded.

```
1: while $b \neq 0$ do
2:     Run the Edmonds-Karp scaling algorithm for $O(\log n)$ iterations, until an abundant arc
       $ij \in E$ is identified, or the algorithm terminates detecting infeasibility or unboundedness.
3:     Contract($ij$).
4: Compute $\pi$ in the current contracted graph such that $c^\pi_e \geq 0$ for all $e \in E$.
5: Expand $\pi$ to a dual optimal solution in the original graph.
6: Compute a $b$-flow $f$ using only tight arcs for $\pi$.
7: return $f$.
```

2.3 Combinatorial linear programs

Tardos gave a far-reaching extension of her minimum-cost flow result to the general LP setting [19]. Consider an LP in the standard form, along with its dual:

$$\begin{aligned} \min\ & c^\top x & \qquad \max\ & b^\top y \\ Ax &= b & y^\top A &\le c \\ x &\ge 0 & & \end{aligned} \tag{2.1}$$

Assume $A \in \mathbb{Z}^{m\times n}$ is an integer matrix, and let us define a bound on the maximum subdeterminant.

$$\Delta(A) \ge \max\{|\det(B)| : B \text{ is a square submatrix of } A\}.$$

Note that for totally unimodular matrices, such as the one in the minimum-cost flow problem, we can use $\Delta(A) = 1$. There are no general methods known to determine the tight value of $\Delta(A)$; we can use the general estimation $\Delta(A) = n^{n/2}|B|^n$ obtained from Hadamard's bound, where $B = \max|a_{ij}|$.

Theorem 2.7 (Tardos [19])**.** *For $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, the LP (2.1) can be solved in time poly$(n, m, \log \Delta(A))$.*

Many combinatorial problems can be described by LPs where the entries in the constraint matrix are small integers. For example, even though the multicommodity flow LP is non-integral, the constraint matrix still contains only $+1$'s and -1's. LPs given by an integer constraint matrix where $\max_{i,j}|a_{ij}| = \text{poly}(n, m)$ are called *combinatorial LPs*. For such LPs, $\log \Delta(A) = \text{poly}(n, m)$. Thus, Tardos's result gives a strongly polynomial algorithm for combinatorial LPs. Let us note that Vavasis and Ye [21] gave a generalization of Theorem 2.7 using an interior point method.

Theorem 2.7 provides a meta-algorithm. It requires in the input a weakly polynomial LP algorithm (such as the ellipsoid or interior point methods), and turns it into a strongly polynomial one using variable fixing. Here, we only present one important step of the argument, that highlights how the dependence on $\Delta(A)$ comes into play. We let A_i denote the i-th column of the matrix A.

Lemma 2.8 (Tardos [19])**.** *Let x be any feasible solution to (2.1). Further, let $y \in \mathbb{R}^m$ a vector and $\varepsilon > 0$ such that $y^\top A_i \le c_i + \varepsilon$, and whenever $y^\top A_i < c_i$, we have $x_i = 0$. For every index k such that $y^\top A_k \le c_k - n\varepsilon\Delta(A)$, we must have $x^*_k = 0$ for every optimal solution x^* to (2.1).*

Note that this is a generalization of (a weaker form of) Lemma 2.1. For $\varepsilon = 0$, x and y satisfy complementary slackness and therefore they are primal and dual optimal solutions to (2.1). The statement gives a relaxation where the dual constraints might be violated by ε. In the proof, we will need the following simple statement.

Lemma 2.9. *Let $\bar{A} \in \mathbb{Z}^{m\times n}$, and for a $1 \le k \le n$, let $e_k \in \mathbb{Z}^n$ be the k-th unit vector and $c \in \mathbb{R}^n$. Then, if the system $\bar{A}z = 0$, $z \ge e_k$ is feasible, then there for every basic feasible solution z we have $\|z\|_\infty \le \Delta(\bar{A})$.*

Proof. Let us select z as a basic feasible solution. This corresponds to the unique solution of a system $Cz' = d$, where $C = \binom{B}{e_k'^\top}$ is a nonsingular matrix and $d = e'_t$. Here, B is a $(t-1)\times t$ submatrix of $\bar{A}$, and e'_k and e'_t are the t dimensional k-th and t-th unit vectors for some $k \le t$. (That is, the last equality in the system is $z'_k = 1$; note that every basic feasible solution must include this as a defining constraint.)

Using Cramer's rule, the i-th component is $z'_i = \det(C^{(i)})/\det(C)$, where $C^{(i)}$ is the matrix obtained from C by replacing the i-th column by d. Since d is a unit vector, we see that $\det(C^{(i)}) \le \Delta(\bar{A})$. Further, $\det(C) \ge 1$ due to the integrality of $\bar{A}$, and thus we have that $z'_i \le \Delta(\bar{A})$, showing the claim. □

Proof of Lemma 2.8. Consider any optimal solution x^* to (2.1), and let us define

$$h_i := \begin{cases} x^*_i - x_i, & \text{if } x_i \le x^*_i, \\ x_i - x^*_i, & \text{if } x_i > x^*_i. \end{cases}$$

Let $S := \{i : 1 \leq i \leq n, x_i > x_i^*\}$. Let $\bar{A}$ denote the matrix we obtain from A by multiplying every column A_i for $i \in S$ by -1; let $\bar{c}$ be the cost function obtained from c by negating the same indices. Then, $\bar{A}h = 0$, $h \geq 0$, and $\bar{c}^\top h = c^\top x^* - c^\top x \leq 0$.

Assume for a contradiction that there exists an index k such that $y^\top A_k < d_k + n\varepsilon\Delta$, yet $x_k^* > 0$. For such indices, $x_k = 0$ was required. Consequently, $h_k > 0$ and $k \notin S$.

Let $\Delta := \Delta(A)$. Thus, for sufficiently large $\alpha > 0$, $\bar{z} = \alpha h$ is a solution to the system

$$\bar{A}\bar{z} = 0, \quad \bar{z} \geq e_k, \quad \bar{z}_i = 0 \text{ if } x_i = x_i^*$$

Let us consider a basic feasible solution z to this system. By Lemma 2.9, $\|z\|_\infty \leq \Delta(\bar{A}) = \Delta(A)$. We claim that $\bar{c}^\top z \leq 0$. Indeed, $\bar{c}^\top z > 0$ would contradict the optimality of x^* (why?). Using $\bar{A}z = 0$, we can write

$$0 \geq \bar{c}^\top z = (\bar{c}^\top - y^\top \bar{A})z.$$

We claim that $\bar{c}_i - y^\top \bar{A}_i \geq -\varepsilon$ for all i, where $\bar{A}_i$ is the i-th column of $\bar{A}$. Indeed, if $i \notin S$, then $\bar{A}_i = A_i$ and $\bar{c}_i = c_i$, and this follows from the assumption $y^\top A_i \leq c_i + \varepsilon$. If $i \notin S$, that is $\bar{A}_i = -A_i$, $\bar{c}_i = -c_i$, then $x_i > x_i^* \geq 0$, therefore by the assumption on x we must have $y^\top A_i \geq c_i$. Consequently, $\bar{c}_i - y^\top \bar{A}_i \geq 0$ for all such indices. Since $\|z\|_\infty \leq \Delta$, we have $(\bar{c}_i - y^\top \bar{A}_i)z_i \geq -\varepsilon\Delta$ for every i.

On the other hand, $k \notin S$, and we have $n\varepsilon\Delta \leq \bar{c}_k - y^\top \bar{A}_k$. Using $z_k \geq 1$ and the above bounds, we get a contradiction from

$$0 \geq (\bar{c}^\top - y^\top \bar{A})z \geq n\varepsilon\Delta - (n-1)\varepsilon\Delta > 0.$$

□

Chapter 3

The generalized flow problem

3.1 Formulations

In the generalized flow problem setting, besides node demands, arc capacities and costs, the arcs are also equipped with positive *gain factors* $\gamma : E \to \mathbb{R}_{>0}$. The flow gets multiplied by the factor $\gamma_e > 0$ while traversing arc e.

These factors may represent some kind of physical change, such as leakage in a fluid network. Another way to view them is that the nodes may represent different types of entities, for example, currencies and commodities, and the gain factors correspond to exchange rates between them. See [1, Chapter 15] for several applications of the model.

The generalized flow maximization problem Similarly to (1.6), we use a capacitated form with node demands. Several variants have been studied in the literature, e.g. it is common to include arc capacities, but all these can be transformed to the form we use. We are given a directed graph $G = (V, E)$ with a specified sink node $t \in V$, node demands $b : V \setminus \{t\} \to \mathbb{R}$, and gain factors $\gamma : E \to \mathbb{R}_{>0}$. We let f_{ij} denote the flow entering arc ij at i; the flow arriving at j is $\gamma_{ij} f_{ij}$. Thus, the net flow at node i is defined as

$$\nabla f_i := \sum_{e \in \delta^-(i)} \gamma_e f_e - \sum_{e \in \delta^+(i)} f_e.$$

The generalized flow maximization problem can be formulated as

$$\begin{aligned} \max \quad & \nabla f_t \\ \text{s.t.} \quad & \nabla f_i \geq b_i \quad \forall i \in V \setminus \{t\} \\ & f \geq 0. \end{aligned} \tag{3.1}$$

The minimum-cost generalized flow problem Assume that we are also given an arc cost $c : E \to \mathbb{R}$. In the minimum-cost version, there is no need to distinguish a sink; it is customary to require equality in the node constraints.

$$\begin{aligned} \min \quad & c^\top f \\ \text{s.t.} \quad & \nabla f_i = b_i \quad \forall i \in V \\ & f \geq 0. \end{aligned} \tag{3.2}$$

3.1.1 Linear programs with two nonzeros per column

Let us consider an LP of the form

$$\begin{aligned} \min \ & c^\top x \\ Ax &= b \\ x &\geq 0, \end{aligned}$$

such that the constraint matrix $A \in \mathbb{Q}^{n \times m}$ contains at most two nonzero entries per column. Let $\mathcal{M}_2(n, m)$ denote the set of all such matrices, and LP_2 the class of LPs defined by such a matrix.

This is a class of LP's that do not fit into the combinatorial LP framework by Tardos. Still, they form a natural subclass of LP. Note that by allowing 3 nonzero entries per column, every LP can be modelled (see exercise). The minimum-cost generalized flow problem falls into LP_2, and the converse is also true:

Theorem 3.1 (Hochbaum[8])**.** *Solving LPs in the class LP_2 can be strongly polynomially reduced to the minimum-cost generalized flow problem.*

Let us now consider the primal feasibility problem $Ax = b$, $x \geq 0$ for a matrix $A \in \mathcal{M}_2(n, m)$.

Theorem 3.2 ([22])**.** *Finding a feasible solution to $Ax = b$, $x \geq 0$ for a matrix $A \in \mathcal{M}_2(n, m)$ can be strongly polynomially reduced to a generalized flow maximization problem.*

The rest of this chapter is dedicated to presenting a strongly polynomial algorithm for generalized flow maximization. Before moving to that, let us also consider the dual feasibility problem $A^\top y \leq c$ for $A \in \mathcal{M}_2(n, m)$. This means that we need to solve a system of linear inequalities with at most two nonzero variables per inequality.

Theorem 3.3 (Megiddo [10])**.** *There exists a strongly polynomial algorithm for finding a feasible solution to a system of linear inequalities with at most two nonzero variables per inequality.*

Megiddo's paper [10] introduced the very notion of strongly polynomial algorithms (although he used the term *genuinely polynomial*). The algorithm uses a variant of Megiddo's parametric search idea that is quite different from the variable fixing techniques discussed in these notes.

To summarize, for LP_2 (or equivalently, for the minimum-cost generalized flow problem) we currently have strongly polynomial algorithms for finding both primal and dual feasible solutions. However, *finding a strongly polynomial algorithm for the minimum-cost circulation problem* remains an open question.

3.2 Basic concepts

Cycle types For a generalized flow f, we define the residual graph E_f as for standard flows. For a reverse arc $ij \in \overleftarrow{E}$, we set $\gamma_{ij} := 1/\gamma_{ji}$. Consider a cycle $C \in \overleftrightarrow{E}$ and a node $i \in V(C)$. If we send δ units of flow around C, then $\delta \prod_{e \in C} \gamma_e$ units arrive back. Let $\gamma(C) := \prod_{e \in C} \gamma_e$. If $\gamma(C) = 1$, then C is called a *unit gain cycle*. If $\gamma(C) > 1$, then C is a *flow generating cycle*, and if $\gamma(C) < 1$, then C is a flow absorbing cycle.

Assume we have a feasible solution f to (3.1) such that E_f contains a flow generating cycle. Then, we can increase ∇f_i by a positive amount by an arbitrary $i \in V(C)$. If further there exists a residual path P from i to the sink t, then we can send a positive amount to t; note that the amount can increase or decrease along the path due to the gain factors. A flow generating cycle C and a path P connecting $V(C)$ and t is called a *generalized augmenting path (GAP)*. If a GAP exists, then f cannot be optimal. We will show the converse using duality.

3.2.1 Assumptions

Before writing the dual of (3.1), let us make some assumptions. These are without loss of generality: the general case can be reduced to instances satisfying these assumptions.

Assumption 3.4. *Assume that for every $i \in V$ there exists an $i - t$ path in E.*

To satisfy this assumption, we may add new *it* arcs with very small gain factors such that they would not have any "real" use in an optimal solution. Under this assumption, an optimal solution to (3.1) must satisfy $\nabla f_i = b_i$ for all $i \in V \setminus \{t\}$. This assumption naturally requires the next one.

Assumption 3.5. *The arc set E does not contain any flow generating cycles.*

Indeed, given Assumption 3.4, if there were a flow generating cycle anywhere in E, then we would obtain an uncapacitated GAP. Thus, the instance would be unbounded or infeasible.

In the algorithms we are going to present, we will make one more assumption:

Assumption 3.6. *Assume that an initial feasible solution is given to* (3.1).

This is a natural assumption in some special cases (e.g. if the uncapacitated instance was obtained from a capacitated instance with 0 demands).

In the general case we can use an approach similar to the two phase simplex method. Once we have an algorithm for generalized flow maximization starting from a feasible solution, we can use it to obtain a feasible solution by applying it to an auxiliary instance.

3.2.2 The dual program and relabellings

Under Assumption 3.4, the dual of (3.1) can be transformed into the following form:

$$\begin{aligned} \max \quad & \mu_t \sum_{j\in V\setminus\{t\}} \frac{b_j}{\mu_j} \\ \text{s.t.} \quad & \mu_j \geq \gamma_{ij}\mu_i \quad \forall ij \in E \\ & \mu > 0 \end{aligned} \tag{3.3}$$

We note that the dual variable for node i would be μ_t/μ_i. Due to Assumption 3.4, no dual value can be 0.

Relabellings We interpret the dual solutions as *relabellings.* These are the basic vehicle of all combinatorial algorithms. A vector $\mu : V \to \mathbb{R}_{>0}$ is called a *labelling.* We define the *relabelled flow* as

$$f^{\mu}_{ij} := \frac{f_{ij}}{\mu_i} \quad \forall ij \in E.$$

The multiplier μ_i can be interpreted as a change of the unit of measurement at node i; for example, a node representing pounds would be denominated in pennies. An equivalent problem instance is obtained by defining

$$\gamma^{\mu}_{ij} := \gamma_{ij} \cdot \frac{\mu_i}{\mu_j}, \qquad \nabla f^{\mu}_i := \frac{\nabla f_i}{\mu_i}, \qquad \text{and} \qquad b^{\mu}_i := \frac{b_i}{\mu_i}.$$

Then the feasibility of μ to (3.3) is equivalent to $\gamma^{\mu}_e \leq 1$ for all $e \in E$. We call an arc $e \in E$ *tight* with respect to μ, if $\gamma^{\mu}_e = 1$.

3.2.3 Fitting pairs and optimality

Let $f \in \mathbb{R}^E_+$ and $\mu \in \mathbb{R}^V_{>0}$. We say that (f, μ) is a *fitting pair*, if μ is feasible to (3.3), and $f_e > 0$ implies $\gamma^{\mu}_e = 1$. We also say that f fits μ or that μ fits f. Equivalently, (f, μ) is a fitting pair if the entire $\operatorname{supp}(f)$ is tight with respect to μ. Fitting captures complementary slackness. Clearly, we have the following:

Lemma 3.7. *Let (f, μ) be a fitting pair such that $\nabla f_i = b_i$ for all $i \in V \setminus \{t\}$. Then f is an optimal solution to* (3.1) *and μ is an optimal solution to* (3.3).

For a fitting pair (f, μ), $\gamma^{\mu}_e = 1$ for all $e \in \operatorname{supp}(f)$. Consequently, f^{μ} is a regular flow; this is true for any pair of primal and dual optimal solutions.

Lemma 3.8. *Given a dual optimal solution to* (3.3)*, a primal optimal solution to* (3.1) *can be found by a (standard) maximum flow computation.*

Proof. According to Lemma 3.7, finding f^{μ} is equivalent to finding a feasible (standard) flow with node demands b^{μ} and using only tight arcs w.r.t. μ. □

3.2.4 Cancelling flow generating cycles

Note that a cycle $C \subseteq \overleftrightarrow{E}$ is flow generating if and only if it is a negative cost cycle for the cost function $c_e := -\log \gamma_e$. Consequently, we can adapt several results for standard flows to the generalized flow setting. Lemma 1.3 immediately implies the following:

Lemma 3.9. *The program (3.3) is feasible if and only if there exists no flow generating cycle. Further, given a flow f, there exists a labelling μ fitting f if and only if G_f contains no flow generating cycles.*

The negative cycle detection subroutine described after Lemma 1.3 can be used to decide whether a flow generating cycle exists. However, one has to be careful to avoid working with irrational costs. For this purpose, we can develop multiplicative variants of the algorithm.

The Goldberg-Tarjan algorithm can be adapted to cancel all flow generating cycles in the residual graph of a flow f. Instead of minimum-mean cycles, we cancel cycles that maximize the geometric mean of the gain, that is, $\gamma(C)^{1/|C|}$. The weakly polynomial analysis in Section 1.4 immediately extends to the multiplicative setting. While the strongly polynomial analysis in Section 2.1.1 does not carry over, Radzik [15] has shown, using a different argument, that this algorithm is also strongly polynomial. We will refer to this as the Goldberg-Tarjan-Radzik or GTR subroutine.

There is however an important difference between the two settings: when cancelling all flow generating cycles, we create additional excess at some nodes of the graph (this is also the reason why the analysis in Section 2.1.1 does not work).

3.2.5 Onaga's algorithm

We now describe Onaga's combinatorial algorithm from 1967 [12]. This can be seen as a natural counterpart of the successive shortest path algorithm. We start with an initial feasible solution as in Assumption 3.6, and cancel all flow generating cyles using the GTR algorithm. Then, as long as there is a node with positive excess, i.e. $\nabla f_i > b_i$, we use a *highest gain path* from an excess node to the sink t in the residual graph G_f, and send the largest amount of flow that does not violate the capacity constraints. Note that a path is highest gain if and only if it is a shortest path with respect to the costs $c_e = -\log \gamma_e$. As for the SSP, if we only augment using highest gain paths, then no flow generating cycle may be created in G_f.

This will be immediate from the description below. After the GTR algorithm, by Lemma 3.9, we have a fitting pair (f, μ). We will maintain a fitting pair throughout. In order to find a highest gain path, we let S be the set of nodes that can reach t on a tight path. As long as it does not include any excess nodes, we uniformly increase all labels μ_i for $i \notin S$ until a new tight arc entering S appears. Note that this change does not affect γ^{μ}_{ij} if $i, j \notin S$ or if $i, j \in S$.

Thus, we obtain a tight path P from an excess node to t. We send $\delta > 0$ amount of *relabelled flow* along P; that is, if $ij \in P$, then we increase f_{ij} by $\delta\mu_i$ so that f^{μ}_{ij} increases by δ. Note that it is immediate that no flow generating cycle may appear: (f, μ) remains a fitting pair after the flow augmentation.

A remarkable difference compared to the SSP algorithm is that while we modify the labelling in order to find the augmenting path, the relabelled flow itself is changing: namely, f^{μ}_{ij} decreases if μ_i increases. Thus, even though we may see high excesses at the beginning of an iteration, this may become arbitrarily smaller by the time an augmenting path is identified.

3.2.6 The Fat Path Algorithm

The first polynomial time combinatorial algorithms (two of them) were given by Goldberg, Plotkin and Tardos [5]. One of these, the Fat Path Algorithm, is a natural adaptation of the Edmonds-Karp scaling to the generalized flow setting.

The algorithm comprises scaling phases with a parameter Δ that decreases at the end of every phase by a factor 2. Inside the Δ-phase, the algorithm sends Δ units of relabelled flow on a highest gain Δ-fat path, that is, a path that has residual relabelled capacity at least Δ. Algorithm 5 can be modified to find such a path by ignoring all arcs in the residual graph that have relabelled capacity

Algorithm 5 Onaga's Algorithm

```
Input: Graph $G = (V, E)$, sink $t \in V$, node demands $b : V \setminus \{t\} \to \mathbb{Q}$, gain factors $\gamma : E \to \mathbb{Q}_{>0}$,
    initial feasible solution $f^0$ to (3.1).
Output: A generalized flow of maximum value.
 1: $f \leftarrow f^0$.
 2: Cancel all flow generating cycles using the GTR algorithm.
 3: Obtain a labelling $\mu$ that fits $f$.
 4: repeat
 5:     $R \leftarrow \{i \in V : \nabla f_i > b_i\}$.                          ▷ excess nodes
 6:     $S \leftarrow \{i \in V : \exists$ a tight path from $i$ to $t$ in $E_f\}$.
 7:     while $S \cap R = \emptyset$ do
 8:         $\alpha \leftarrow \max\{\gamma^\mu_{ij} : ij \in E_f, i \notin S, j \in S\}$.
 9:         $\mu_i \leftarrow \alpha\mu_i$ for all $i \notin S$.
10:         $S \leftarrow \{i \in V : \exists$a tight path from $i$ to $t$ in $E_f\}$.
11:     Select a tight path $P$ from a node $r \in R$ to $t$.
12:     $\delta \leftarrow \min\left\{\nabla f^\mu_r - b^\mu_r, \min_{e \in P \cap \overleftarrow{E}} f^\mu_e\right\}$. Send $\delta$ units of relabelled flow on $P$.
13: until $\nabla f_i = b_i$ for all $i \in V \setminus \{t\}$.
14: return $f$.
```

$< \Delta$; note that the residual capacities may decrease during the label changes. We omit the precise description here.

An additional difficulty is that we may create new flow generating cycles. Thus, the algorithm repeatedly needs to call the GTK subroutine at the beginning of every Δ-phase; it can be shown that the total increase of the relabelled excess remains bounded by $O(m\Delta)$.

3.3 A strongly polynomial algorithm

The paper [5] was followed by a number of weakly polynomial algorithms. Among them, the best running times are the $O(m^{1.5}n^2 \log(nB))$ interior point method by Vaidya [20]; and the $O(mn(m + n \log n) \log B)$ combinatorial algorithm by Radzik [16]. See Shigeno's survey [17] for an overview. The first strongly polynomial was given in 2014 [22] in time $O(n^3m^2)$. We now outline much simpler algorithm in [11] that runs in time $O(mn(m + n \log n) \log(n^2/m))$ for the uncapacitated formulation (3.1).

3.3.1 Abundant arcs and plentiful nodes

Lemma 2.4 easily extends to the generalized flow setting, using the flow decomposition for generalized flows. We define the total relabelled excess as

$$\mathrm{Ex}(f, \mu) := \sum_{i \in V \setminus \{t\}} \max\left\{\nabla f^\mu_i - b^\mu_i, 0\right\} \tag{3.4}$$

Lemma 3.10. *Let f be feasible to (3.1), and let (f, μ) be a fitting pair. If $f^\mu_{ij} > \mathrm{Ex}(f, \mu)$, then, $\gamma^{\mu^*}_{ij} = 1$ must hold for every optimal solution μ^* to (3.3).*

Such an arc will be called an *abundant arc* w.r.t. the pair (f, μ). Once we find a fitting pair with such an arc (i, j), we can reduce the instance by contracting (i, j). As in Orlin's algorithm, our goal will be to find an optimal solution to the dual (3.3). Once an optimal dual is found, we use Lemma 3.8 to compute a primal optimal solution to (3.1).

The contractibility of abundant arcs was already used in [16]. The main difficulty is that in the natural weakly polynomial algorithms such as the Fat Path algorithm, no proof is known for the existence of an abundant arc. E.g. in the Fat Path algorithm, the total relabelled excess can be $\Theta(m\Delta)$; but it could be possible that the relabelled flow remains smaller on every arc throughout the algorithm. In [22], a new scaling variant, called *continuous scaling* was used to overcome this problem. In what follows, we describe the algorithm in [11].

Let us say that a node i is *plentiful*, if $|b_i^\mu| > 2n^2$. The following lemma follows easily from Lemma 3.10, by looking at the arcs in $\delta^-(i)$ if $b_i > 0$, or the arcs in $\delta^+(i)$ if $b_i < 0$.

Lemma 3.11. *Let (f,μ) be a fitting pair such that f is feasible to to (3.1). Assume further that* $\mathrm{Ex}(f,\mu) \le 2n$. *Then, if a plentiful node exists, then (f,μ) admits an abundant arc.*

Our goal will be to obtain an abundant arc using this lemma. While Onaga's algorithm simultaneously maintains feasibility and fitting pairs, these two properties appear to be incompatible for polynomial time algorithms. E.g., the Fat Path algorithm needs to relax the fitting property and may create flow generating cycles in every scaling phase. The algorithm in [22] also maintains feasibility and gives a Δ-relaxation of the notion of fitting pairs.

Safe labellings A key trick in [11] is to maintain the fitting property but relax feasibility instead. Throughout the algorithm, we work with a fitting pair (f,μ), but f may have some nodes $i \in V \setminus \{t\}$ with $\nabla f_i < b_i$. Without any further conditions, such a flow is not very helpful, e.g. Lemma 3.10 cannot be extended even if we take the amount of violations into account.

Our additional requirement is that μ must be a *safe* labelling, meaning that there exists a feasible solution g to (3.1) that fits μ. Given a labelling μ deciding safety reduces to a a standard flow problem - one can use Hoffman's theorem(Theorem 1.4).

We need another property of standard flows, known as the *linking property*. We leave the proof as an exercise.

Theorem 3.12. *Consider a directed graph $G = (V,E)$, and $p, b : V \to \mathbb{R}$, $p \le b$. Assume there exists a (standard) flow $f \ge 0$ such that $\nabla f \ge p$, and there exists another flow $f' \ge 0$ with $\nabla f' \le b$. Then, there exists a flow $f'' \ge 0$ that simultaneously satisfies $p \le \nabla f'' \le b$.*

When applied for the relabelled flows, we obtain the following:

Lemma 3.13. *Let (f,μ) be a fitting pair such that μ is a safe labelling and $f \ge 0$. Then there exists a $g \ge 0$ that is feasible to (3.1) such that g fits μ, and $Ex(g,\mu) \le Ex(f,\mu)$.*

3.3.2 The algorithm

Invariants Nodes with positive and negative demands will play different roles. We let

$$V^+ := \{i \in V \setminus \{t\} : b_i > 0\}, \quad V^- := \{i \in V \setminus \{t\} : b_i < 0\}.$$

Throughout the algorithm, we maintain a fitting pair (f,μ) satisfying the following properties

- The labelling μ is safe.
- The relabelled flow f^μ is integer valued.
- For every node $i \in V \setminus \{t\}$, $\nabla f_i^\mu - b_i^\mu \le 2$.
- For every node $i \in V^-$, $\nabla f_i^\mu - b_i^\mu > -1$.

Note that the lower bounds are only imposed for V^-. The integrality property is remarkable in the context of generalized flows. Due to the gain factors, this problem is inherently non-integral; e.g. finding an integer valued generalized flow is an NP complete problem.

Initialization Let us assume that we have some fitting pair $(\bar{f},\bar{\mu})$ with $\bar{f}$ feasible to (3.1) to start with. Assumption 3.6 provides a feasible solution. We could cancel all flow generating cycles using the GTK subroutine and obtain a label $\bar{\mu}$ fitting the flow $\bar{f}$ at this point using Lemma 3.9. The algorithm [11] avoids using GTK via a two-phase simplex type approach.

At this point, $\bar{\mu}$ is safe as testified by $\bar{f}$. To obtain the integrality of the relabelled flow as well as the upper and lower bounds on the excess, we first set $\mu := \alpha\bar{\mu}$ such that $0 \le \nabla\bar{f}^\mu_i - b^\mu_i \le 1$ holds for every $i \in V \setminus \{t\}$. Now the standard flow $\tilde{g} = \bar{f}^\mu$ satisfies that $b^\mu_i \le \nabla\tilde{g} \le b^\mu_i + 1$ for all $i \in V \setminus \{t\}$. Using the well-known integrality of the flow polyhedron, there must exists an integer flow $g \ge 0$ such that

$$\lfloor b^\mu_i \rfloor \le \nabla g \le \lceil b^\mu_i \rceil + 1 \quad \forall i \in V \setminus \{t\}$$

Now, defining $f_{ij} := \mu_i g_{ij}$ will provide a flow that fits μ, f^μ is integral, and $-1 < \nabla f^\mu_i - b^\mu_i < 2$ for all $i \in V \setminus \{t\}$.

Finding a plentiful node Algorithm 6 provides the subroutine for finding a plentiful node. Once a plentiful node is obtained, we use Lemmas 3.13 and 3.11 to obtain an abundant arc. We then contract such an arc and thereby reduce the size of the instance. We terminate once all demands are $b_i = 0$. In this case, $f = 0$ will be an optimal primal solution along with any labelling feasible to (3.3). As in Orlin's algorithm, we expand this dual solution to a dual optimal solution in the original graph. Lemma 3.8 then returns an optimal primal solution to (3.1).

Consider now (f,μ) satisfying the above invariants. We wish to update f by augmenting on a highest gain path. We use a modification of Onaga's algorithm, however, there is a crucial difference in how the labels are updated.

Besides t, the sink set Q comprises all nodes with $\nabla f_i < b_i$; we wish to find an augmenting path ending in Q. We let S denote the set of nodes that can reach Q on a tight path. Once S includes a node r with $\nabla f^\mu_i \ge b^\mu_i + 1$, then we send 1 unit of relabelled flow from r to Q. Note that integrality plays a vital role here: it ensures that the residual path has enough capacity to send a full unit. This flow update maintains integrality.

Algorithm 6 PRODUCE-PLENTIFUL-NODE

Input: Fitting pair (f,μ) with $f^\mu \in \mathbb{Z}^E_+$, in a network with $b \neq \emptyset$.
Output: Fitting pair (f,μ) with $f^\mu \in \mathbb{Z}^E_+$, such that there exists at least one plentiful node in V.

```
1:  while there are no plentiful nodes do
2:      Augmentation part of the iteration
3:      $Q \leftarrow \{t\} \cup \{j \in V \setminus \{t\} : \nabla f^\mu_i < b^\mu_i\}$.
4:      $S \leftarrow \{i \in V : \exists$ a tight $i$-$Q$-path in $E_f\}$.
5:      while there exists a node $i \in S$ with $\nabla f^\mu_i \ge b^\mu_i + 1$ do
6:          Augment $f^\mu$ by sending 1 unit from $i$ to a vertex in $Q$ along tight arcs.
7:          Update $Q$ and $S$.

8:      Label update part of the iteration
9:      $\alpha_0 \leftarrow \min_{e\in\delta^-(S)} 1/\gamma^\mu_e$.
10:     Select the largest $\alpha \le \alpha_0$ such that $\nabla f^\mu_i \le \alpha b^\mu_i + 2$ for all $i \in V \setminus \{t\}$ or until $|b^\mu_i|\alpha \ge 2n^2 + 1$
    for some $i \in V \setminus \{t\}$.
11:     $f_{ij} \leftarrow f_{ij}/\alpha$ for all $i, j \in S$.
12:     $\mu_i \leftarrow \mu_i/\alpha$ for all $i \in S$.
13: return $(f,\mu)$.
```

Assume now S does not contain any node with large relabelled excess. In Onaga's algorithm, we multiplied all labels outside S to obtain a tight arc entering S. We now do the opposite: we divide all labels inside S by a factor $\alpha > 1$ (including the label μ_t). Further, we also *change the flow* inside S: for every arc ij with $i, j \in S$, we divide f_{ij} by the same factor α. (This is the most important difference compared to [22]).

Note that such an update maintains the same relabelled flow values f_e^μ for all arcs. Indeed, $f_e = 0$ if $e \in \delta^-(S) \cup \delta^+(S)$ (why?). Thus, flow integrality is maintained.

However, the excess of a node may change. Since the relabelled flow is unchanged, ∇f_i^μ remains the same for all $i \in V \setminus \{t\}$. However, for $i \in S$, the relabelled demand b^μ is divided by a factor α.

This flow update has opposite effect for nodes in V^+ and in V^-. For $i \in V^+$, the relabelled demand increases, and therefore the relabelled excess decreases; this is the reason why we must allow violation of feasibility. For $i \in V^-$, the relabelled demand decreases and thus the relabelled excess may grow.

The value of α is selected as the largest value such that the properties $\gamma_e^\mu \le 1$ for all $e \in \delta^-(S)$, and that $\nabla f_i^\mu \le b_i^\mu + 2$ for all $i \in V^-$ are maintained. We further cap the value of α when a new plentiful node would appear.

3.3.3 Analysis

From the discussion above, we already see that most invariants are maintained in the path augmentations step: (f, μ) remains a fitting pair, the excesses of the nodes remain upper bounded by 2, and the excesses of nodes in V^- remain lower bounded by -1. However, the key *safety* property must be showed.

Lemma 3.14. *Throughout Algorithm 6, μ remains a safe labelling.*

Proof. The labelling μ is only changed when we divide all values in S by α. Let μ' be the updated label such that $\mu'_i = \mu_i$ if $i \notin S$ and $\mu'_i = \mu_i/\alpha$ if $i \in S$.

Let $g \ge 0$ be the flow before the update that testified the safety of μ, that is (g, μ) is a fitting pair and $\nabla g_i \ge b_i$ for all $i \in V \setminus \{t\}$. Note that g may not fit μ', since $g_e > 0$ could be possible for $e \in \delta^+(S)$. These are arcs that can be tight w.r.t. to μ but not w.r.t. μ'.

However, we can combine f and g into a flow h that is feasible and fits μ' as follows. We let

$$h_{ij} := \begin{cases} f_{ij} & \text{if } i, j \notin S, \\ g_{ij} & \text{if } i, j \in S, \\ 0 & \text{if } ij \in \delta^-(S) \cup \delta^+(S). \end{cases}$$

We leave it as an exercise to verify that h is feasible and fits μ. □

The key of the running time analysis is showing that a small number of augmentations leads to obtaining a plentiful node.

Lemma 3.15. *Within $O(n^3)$ path augmentations, Algorithm 6 finds a plentiful node.*

Proof. First, assume that $V^- = \emptyset$. In this case, the label updates can only decrease the excess of any node. Since the initial excess is < 2, there can be just a single augmentation starting from any node $i \in V \setminus \{t\}$. Thus, within $O(n)$ augmentations, we will have $\nabla f_i < b_i + 1$ for every $i \in V \setminus \{t\}$. At this point, the algorithm encounters $S = V$, and we select an α such that $|b_i^\mu|\alpha = 2n^2 + 1$, that is, a new plentiful node is created.

Let us now assume that $V^- \ne \emptyset$. We consider two potentials

$$\Phi := \sum_{i \in V^-} \nabla f_i^\mu - b_i^\mu, \quad \Psi := \sum_{i \in V^-} -b_i^\mu.$$

Clearly, once $\Psi \ge 2n^3$, one of the nodes in V^- must be abundant. Let us now track these two potentials throughout the algorithm. Ψ may change at the label update steps, and can only increase. Further, at every label update step, Ψ increases by the same amount as Φ increases.

Let us call a path augmentation *helpful*, if it starts from a node $r \in V^-$ and ends in a node $q \in Q \setminus V^-$; all other augmentations are called unhelpful. The number of unhelpful augmentations is bounded by $2n$. Indeed, a node outside V^- can serve as a starting point at most once, and a node in V^- can serve as an endpoint at most once (why?).

Throughout, we have $-n \le \Psi \le 2n$ due to upper and lower bounds on the excesses. In every helpful augmentation, Ψ decreases by one. As argued above, the total increase in Ψ equals the total increase in Φ; and Φ can increase by at most $2n^3$ until a plentiful node is found. Thus, the total number of helpful path augmentations is bounded by $2n^3 + 3n$. □

Improvements One can modify the algorithm so that a path augmentation takes the same time $O(m + n \log n)$ as Dijsktra's algorithm. Since we terminate in $\leq n - 1$ contraction, this would yield a running time bound $O(n^4(m + n \log n))$. By defininig the plentiful threshold differently, the $O(n^3)$ bound in the lemma can be decreased to $O(nm)$, leading to an improved $O(n^2 m(m + n \log n))$.

Further, [11] gives an amortized analysis over all contraction phases that yields the running time $O(mn(m + n \log n) \log(n^2/m))$.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., feb 1993.

[2] E. A. Dinits. Algorithm for solution of a problem of maximum flow in a network with power estimation (in Russian). *Doklady Akademii Nauk SSSR 194*, 1970. English translation: Soviet Mathematics Doklady 11 (1970) 1277-1280.

[3] J. Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards B*, 71:241–245, 1967.

[4] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[5] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16(2):351, 1991.

[6] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM (JACM)*, 36(4):873–886, 1989.

[7] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

[8] D. S. Hochbaum. Monotonizing linear programs with up to two nonzeroes per column. *Operations Research Letters*, 32(1):49–58, 2004.

[9] L. G. Khachiyan. A polynomial algorithm in linear programming (in Russian). *Doklady Akademiia Nauk SSSR 224*, 224:1093–1096, 1979. English translation: Soviet Mathematics Doklady 20, 191-194.

[10] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12(2):347–353, 1983.

[11] N. Olver and L. A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 100–111. ACM, 2017.

[12] K. Onaga. Optimum flows in general communication networks. *Journal of the Franklin Institute*, 283(4):308–327, 1967.

[13] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[14] J. B. Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774. ACM, 2013.

[15] T. Radzik. Approximate generalized circulation. *Technical Report93-2, Cornell Computational Optimization Project, Cornell University*, 1993.

[16] T. Radzik. Improving time bounds on maximum generalised flow computations by contracting the network. *Theoretical Computer Science*, 312(1):75–97, 2004.

[17] M. Shigeno. A survey of combinatorial maximum flow algorithms on a network with gains. *Journal of the Operations Research Society of Japan*, 47:244–264, 2004.

[18] S. Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20(2):7–15, 1998.

[19] É. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, pages 250–256, 1986.

[20] P. M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 332–337. IEEE, 1989.

[21] S. A. Vavasis and Y. Ye. A primal-dual interior point method whose running time depends only on the constraint matrix. *Mathematical Programming*, 74(1):79–120, 1996.

[22] L. A. Végh. A strongly polynomial algorithm for generalized flow maximization. *Mathematics of Operations Research*, 42(2):179–211, 2017.