Lecture Notes: Selected Topics in Algorithms

Palash Dey
Indian Institute of Technology, Kharagpur
palash.dey@cse.iitkgp.ac.in

Statutory warning: This is a draft version and may contain errors. If you find any error, please send an email to the author.

Contents

Notation: $\mathbb{N} = \{0, 1, 2, ...\}$ denotes the set of natural numbers, $\mathbb{R}$ denotes the set of real numbers. For a set $\mathcal{X}$, its power set is denoted by $2^{\mathcal{X}}$.

Chapter 1

Matching in Graphs

1.1 Minimum Weight Perfect Matching

We need the following result from linear programming whose proof is available in any standard textbook on linear programming.

Theorem 1 (Complementary Slackness). *Let $A = (a_{ji})_{1 \leqslant j \leqslant m, 1 \leqslant i \leqslant n} \in \mathbb{Q}^{m \times n}$ be any $m \times n$ rational matrix, $c = (c_i)_{i \in [n]} \in \mathbb{Q}^n$, $b = (b_j)_{j \in [m]} \in \mathbb{Q}^m$ any rational vectors, $x = (x_i)_{i \in [n]}$ and $y = (y_j)_{j \in [m]}$. Consider the standard primal and dual pair of linear programs:*

$$\begin{aligned} &\textit{minimize} \quad c \cdot x && &&\textit{maximize} \quad b \cdot y \\ &\textit{subject to} \quad Ax \geqslant b, && &&\textit{subject to} \quad A^t y \leqslant c \\ &\qquad\qquad x \geqslant 0, && &&\qquad\qquad y \geqslant 0, \end{aligned}$$

Let x^* and y^* be feasible solutions of primal and dual linear programs respectively. Then x^* and y^* are optimal solutions of their respective linear programs if and only if they satisfy complementary slackness conditions:

$$\forall i \in [n], \textit{ either } x_i = 0 \textit{ or } \sum_{j=1}^{m} a_{ij} y_j = c_j \textit{ (or both)}$$

$$\forall j \in [m], \textit{ either } y_j = 0 \textit{ or } \sum_{i=1}^{n} a_{ij} x_i = b_i \textit{ (or both)}$$

In the minimum weight perfect matching problem, we are given a weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, w : \mathcal{E} \longrightarrow \mathbb{Q})$ and the goal is to compute a minimum-weight perfect matching of $\mathcal{G}$. We now describe an algorithm for computing a minimum-weight perfect matching og $\mathcal{G}$. The following integer linear program is a formulation of the minimum weight perfect matching problem.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in \mathcal{E}} w_e x_e \\ \text{s.t.} \quad & \sum_{e \text{ incident on } v} x_e = 1, \quad \forall v \in \mathcal{V} \\ & x_e \in \{0, 1\}, \quad \forall e \in \mathcal{E} \end{aligned}$$

We can see that every feasible solution of the above ILP is the characteristic vector of a perfect matching.

1.2 Edmond's Blossom Algorithm

Given an unweighted graph $G(V, E)$, the maximum cardinality matching problem asks to compute a matching in G of maximum cardinality. Let us begin with working towards an upper bound on the size (number of edges) of a maximum cardinality matching. Let $U \subseteq V$ be any arbitrary subset of V. We denote the number of components of odd cardinality in $G[V \setminus U]$ by $o(V \setminus U)$. We observe that any component of odd cardinality cannot be perfectly matched within itself (that is, using only the edges whose both endpoints are in that component). Any such odd component C needs at least one partner outside C and such a partner can come only from U (why?). Hence, every matching of G necessarily leaves at least $o(V \setminus U) - |U|$ vertices of G unmatched. Let M be any matching of maximum cardinality; size of the matching M, denoted by $|M|$, is the number of edges in it. Then we have

$$|M| \leqslant \frac{1}{2}(|V| - (o(V \setminus U) - |U|)) = \frac{1}{2}(|V| + |U| - o(V \setminus U)).$$

Since the above inequality holds for every $U \subseteq V$, we have

$$|M| \leqslant \min_{U \subseteq V} \frac{1}{2}(|V| + |U| - o(V \setminus U)) \tag{1.1}$$

The celebrated Tutte-Berge Theorem says that the above inequality is actually tight.

Theorem 2 (Tutte-Berge Theorem)**.** *For any maximum cardinality matching M of an undirected graph G, we have*

$$|M| = \min_{U \subseteq V} \frac{1}{2}(|V| + |U| - o(V \setminus U)).$$

We now prove this result by exhibiting a polynomial-time algorithm, thanks to Edmond, which computes a matching M and a subset $S \subseteq V$ such that $|M| = \frac{1}{2}(|V| + |S| - o(V \setminus S))$ which is enough to prove the Tutte-Berge Theorem due to Equation (1.1). The algorithm starts with empty matching and iteratively either finds another matching of size one more that the size of the current matching or conclude that the current matching is a maximum cardinality matching. To proceed, we focus on the following questions.

1. Given a matching M which is not of maximum cardinality, how do we compute another matching M' such that $|M'| > |M|$?

2. Given a matching M, how to certify that M is a matching of maximum cardinality?

Let us begin with the first question. The concept involved here is known as an M-augmenting path. A path $P = (x_0, x_1, \ldots, x_k)$ is called an M-augmenting path if (i) x_0 and x_k are unmatched (a.k.a free, exposed, etc.) vertices and (ii) $\{x_{2i-1}, x_{2i}\} \in M$ for every $1 \leqslant i \leqslant k/2$; that is, edges in the path alternate from matching and out of matching, beginning and ending with unmatched vertices. If we have an M-augmenting path P, then we can use P to obtain another matching M' with $|M'| = |M| + 1$: $M' = M \Delta P$[1]. Hence, if we have found an M-augmenting path, we can use it to obtain another matching of size one more than the size of M. Does an M-augmenting path guaranteed to exist if M is not a matching of maximum cardinality? The answer is yes! To see this, let M be any matching and M' be another matching such that $|M'| > |M|$. Then we will show that there exists an M-augmenting path. For that, let us consider the subgraph $H = M \Delta M'$; it is a collection of vertex disjoint cycles and paths where the edges alternate between M and M'. In the

[1]We slightly abuse the notation P to also denote the set of edges in the path P. For two sets X and Y, we define $X \Delta Y = (X \setminus Y) \cup (Y \setminus X)$; this is also known as symmetric difference between X and Y.

cycles, we have the same number of edges from $\mathcal{M}$ and $\mathcal{M}'$. Among the paths, only $\mathcal{M}$-augmenting paths have more edges from $\mathcal{M}'$ than $\mathcal{M}$. Hence, if we do not have any $\mathcal{M}$-augmenting path in $\mathcal{H}$, then we have $|\mathcal{M}| \leqslant |\mathcal{M}'|$ (why?) contradicting our assumption that $|\mathcal{M}'| > |\mathcal{M}|$. Hence, one of the disjoint path must be an $\mathcal{M}$-augmenting path. Hence, an $\mathcal{M}$-augmenting path is guaranteed to exist if $\mathcal{M}$ is not a maximum cardinality matching. This finishes our answer to the first question. This is also known as Berge's Theorem.

Theorem 3 (Berge's Theorem)**.** *A matching $\mathcal{M}$ is a maximum cardinality matching in a graph $\mathcal{G}$ if and only if there is no $\mathcal{M}$-augmenting path in $\mathcal{G}$.*

From algorithmic perspective, all we need is a recipe which is guaranteed to find an $\mathcal{M}$-augmenting path when one exists (which is exactly when $\mathcal{M}$ is not a matching of maximum cardinality). Actually, given a matching $\mathcal{M}$ that is not a maximum cardinality matching, it is enough to find a matching $\mathcal{M}'$ (that could be $\mathcal{M}$ also) with $|\mathcal{M}| = |\mathcal{M}'|$ and an $\mathcal{M}'$-augmenting path to obtain another matching of cardinality one more than the cardinality of $\mathcal{M}$.

Algorithm for Finding an Augmenting Path

Let us start looking for an $\mathcal{M}$-augmenting path. Since we are looking for an $\mathcal{M}$-augmenting path, it makes sense to start our search from an unmatched vertex since every $\mathcal{M}$-augmenting path starts at some unmatched vertex. To make our search systematic, we perform the following modified version of breadth first search (BFS). We begin with marking all unmatched vertices "even" (denoting the fact that these vertices are at even, namely zero, level in the BFS forest that we are building) and enqueueing them like standard BFS; other vertices are unlabeled. An important difference between our modified BFS and standard BFS is that we enqueue only vertices marked "even" which are precisely the vertices at an even distance from the root of some tree in the forest that we are maintaining. In every iteration, we check if the queue is empty or not. If the queue is empty, then we declare that the matching $\mathcal{M}$ is a maximum cardinality matching. Otherwise, we dequeue a (that is already marked "even") vertex, say u, and process every edge $\{u, v\}$ incident on u. We have the following possibilities:

1. (easy) If the vertex v is marked "odd," then we simply ignore this edge. That is, we simply ignore edges between an even vertex and another vertex which is already marked "odd."

2. (easy) If the vertex v is unmarked, we observe that v must be a matched vertex; let $\mathcal{M}(v)$ be its mate (matched vertex). In this case, we add the edges $\{u, v\}$ and $\{v, \mathcal{M}(v)\}$ in the forest and mark v and $\mathcal{M}(v)$ "odd" and "even" respectively. Hence, every vertex which is marked "odd" has exactly one child in the forest we are maintaining.

3. (easy) If the vertex v is marked "even" and belongs to some other tree (that is, not the tree where u belongs) in the forest that we are maintaining, then we have obtaining an $\mathcal{M}$-augmenting path — root of u's tree $\dashrightarrow u \to v \dashrightarrow$ root of v's tree. Hence, we are done in this case; remember our goal is to find an $\mathcal{M}$-augmenting path if one exists.

4. If the vertex v is marked "even" and belongs to the same tree as u. In this case, then we have found a structure like Figure 1.1 which is called a flower. A flower has a cycle of odd length called a blossom and an even length alternating path called stem from an unmatched vertex a_k. In our modified BFS, the root of the tree of u serves the role of a_k which is an unmatched vertex. Note that, the stem could be empty but not the blossom. If the flower obtained has no stem, then we are good, otherwise we

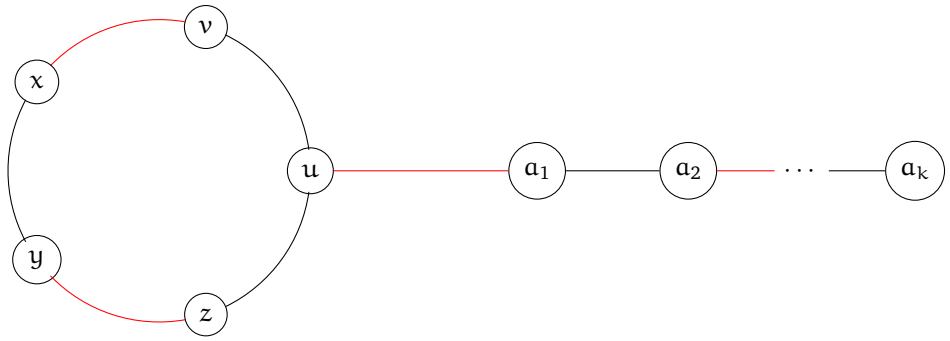

Figure 1.1: Schematic diagram of a flower. Red edges are matching edges. The cycle is called blossom and the path between u and a_k is called stem.

remove the matched edges in the stem from $\mathcal{M}$ and add the remaining edges in the stem in $\mathcal{M}$ thereby obtaining another matching $\mathcal{M}'$ of the same size as $\mathcal{M}$, but we now have a flower with empty stem.

Let $\mathcal{B}$ be that flower which is just a blossom; $\mathcal{B}$ has exactly one unmatched edge. How do we proceed when we have found a blossom $\mathcal{B}$ (mentioning for the last time that $\mathcal{B}$ does not have any stem)? We collapse the vertices of $\mathcal{B}$ into one (super)node, mark that (super)node unmatched, and restart our task of finding an $(\mathcal{M}'/B)$-augmenting path in the resulting graph. The collapsing operation is formally called contracting the vertex set $\mathcal{B}$ and resultant graph is denoted by $\mathcal{G}/\mathcal{B}$.

Proof of Correctness

We next show that there is an $\mathcal{M}$-augmenting path in $\mathcal{G}$ if and only if there is an $(\mathcal{M}/\mathcal{B})$-augmenting path in $\mathcal{G}/\mathcal{B}$. In one direction, let a path $\mathcal{P}$ between x and y be an $\mathcal{M}$-augmenting path in $\mathcal{G}$. Since both x and y are unmatched, let us assume without loss of generality that $x \neq u$. Let z be the first vertex in $\mathcal{P}$ from x which belongs to $\mathcal{B}$; if no such z exists, then $\mathcal{P}$ is an $(\mathcal{M}/\mathcal{B})$-augmenting path in $\mathcal{G}/\mathcal{B}$. We observe that the prefix of the path $\mathcal{P}$ from x to z is an $(\mathcal{M}/\mathcal{B})$-augmenting path in $\mathcal{G}/\mathcal{B}$ since z becomes the supernode in $\mathcal{G}/\mathcal{B}$ which is unmatched in $\mathcal{G}/\mathcal{B}$.

In the other direction, let $\mathcal{Q}$ be an $(\mathcal{M}/\mathcal{B})$-augmenting path in $\mathcal{G}/\mathcal{B}$. If $\mathcal{Q}$ does not pass through the supernode, then $\mathcal{Q}$ is an $\mathcal{M}$-augmenting path in $\mathcal{G}$ too. So let us assume that $\mathcal{Q}$ passes through the supernode. Since the supernode is unmatched in $\mathcal{G}/\mathcal{B}$, it must be one of the end vertices in $\mathcal{Q}$. Let z_1 be the neighbor of the supernode in $\mathcal{Q}$ and z_2 the neighbor of z_1 in $\mathcal{B}$ in $\mathcal{G}$; observe that $\{z_1, z_2\} \notin \mathcal{M}$ since the supernode is unmatched in $\mathcal{G}/\mathcal{B}$. If z_2 is u which is an unmatched vertex in $\mathcal{G}$ under $\mathcal{M}$, then replacing the supernode with u gives us an $\mathcal{M}$-augmenting path in $\mathcal{G}$. Otherwise, we replace the supernode in $\mathcal{Q}$ with z_2 and concatenate the $\mathcal{M}$-alternating from z_2 to u of even length in $\mathcal{B}$ to obtain an $\mathcal{M}$-alternating path in $\mathcal{G}$.

An important point to note that an $\mathcal{M}$-augmenting path in $\mathcal{G}$ can be computed from an $(\mathcal{M}/\mathcal{B})$-augmenting path in $\mathcal{G}/\mathcal{B}$ in $\mathcal{O}(m)$ time. Recall that we have already reduced our problem of computing a matching of maximum cardinality to the problem of computing an $\mathcal{M}$-augmenting path from a given matching $\mathcal{M}$.

What remains to show to prove the correctness of the algorithm is that when the queue is found to be empty, the current matching $\mathcal{M}$ is indeed a maximum cardinality matching. We observe that when the queue is empty, every edge of the graph falls into one of three types: (i) edges present in the BFS forest, (ii) edges discovered during modified BFS but ignored since their other end points are already marked "odd", and (iii)

edges not discovered by the modified BFS algorithm. We observe that all type (i) and (ii) edges have one end point marked "even" and the other end point marked "odd." All type (iii) edges have both their end points unmarked. Hence, all the vertices marked "even" becomes isolated in $\mathcal{G}[\mathcal{V} \setminus \mathcal{U}]$; that is $o(\mathcal{V} \setminus \mathcal{U}) = |\text{Even}|$ where Even is the set of vertices marked "even." On the other hand,

$$|\mathcal{M}| = |\text{Odd}| + \frac{1}{2}(|\mathcal{V}| - |\text{Even}| - |\text{Odd}|) = \frac{1}{2}(|\mathcal{V}| + |\text{Odd}| - |\text{Even}|) = \frac{1}{2}(|\mathcal{V}| + |\mathcal{U}| - o(\mathcal{V} \setminus \mathcal{U})).$$

Notice that, $\frac{1}{2}(|\mathcal{V}| + |\mathcal{U}| - o(\mathcal{V} \setminus \mathcal{U}))$ is an upper bound on the cardinality of any matching in $\mathcal{G}$. Hence, $\mathcal{M}$ is a matching of maximum cardinality. Hence, when the algorithm does not find any augmenting path or flower, then the current matching is indeed a matching of maximum cardinality. This concludes the proof of correctness of Edmond's algorithm. This also proves Tutte-Berge Theorem.

Runtime of Blossom Algorithm

Finally, we analyze the runtime of this algorithm. Since the maximum cardinality matching can have at most $\frac{n}{2}$ edges, we can augment the current matching at most $\frac{n}{2}$ times. Between two consecutive augmentations, how many times we can find a blossom (and thus shrink it and restart our modified BFS)? At most $\frac{n}{2}$ times since contracting a blossom reduces the number of vertices by at least two. Each run of our modified BFS can be implemented in $\mathcal{O}(m)$ time. Hence, the overall runtime of our algorithm is $\mathcal{O}(mn^2)$.

Chapter 2

Integrality of Polyhedra

We begin with introducing some terminologies that we will use going forward.

Definition 1 (Convex combination)**.** *A convex combination of $v_1, \dots, v_k \in \mathbb{R}^n$ is $\sum_{i=1}^{k} \lambda_i v_i$ for any $\lambda_1, \dots, \lambda_k \in \mathbb{R}$ with $\lambda_i \geqslant 0$ for every $i \in [k]$ and $\sum_{i=1}^{k} \lambda_i = 1$.*

Definition 2 (Convex hull)**.** *The convex hull of a finite set $S \subset \mathbb{R}^n$ is the set of all convex combinations of the points in S.*

The following proposition is easy to see and prove.

Proposition 1. *For any finite subset $S \subset \mathbb{R}^n$ and $w \in \mathbb{R}^n$, we have*

$$\max\{w^T x : x \in S\} = \max\{w^T x : x \in \textit{conv.hull}(S)\}.$$

A hyperplane is the solution set of $w^T x = t$ for any $w \in \mathbb{R}^n$ and $t \in \mathbb{R}$. We will also use the following proposition which is easy to see and can be proved using Farkas' Lemma.

Proposition 2. *For any finite subset $S \subset \mathbb{R}^n$ and $v \in \mathbb{R}^n \setminus \textit{conv.hull}(S)$, there exists a hyperplane $w^T x = t$ that separates v from conv.hull(S). That is, $w^T x \leqslant t$ for every $x \in \textit{conv.hull}(S)$ and $w^T v > t$.*

Definition 3 (Polyhedron)**.** *A polyhedron is the solution set of a system of linear inequalities. That is, any matrix $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ defines a polyhedron, namely $\{x \in \mathbb{R}^n : Ax \leqslant b\}$.*

Definition 4 (Valid and supporting hyperplane,)**.** *A hyperplane $w^T x = t$ is called a valid hyperplane for a polyhedron $\mathcal{P}$ if $\mathcal{P} \subseteq \{x \in \mathbb{R}^n : w^T x \leqslant t\}$. A valid hyperplane $w^T x = t$ is called a supporting hyperplane for $\mathcal{P}$ if $\mathcal{P} \cap \{x \in \mathbb{R}^n : w^T x = t\} \neq \emptyset$.*

Definition 5 (Face and vertex)**.** *A face of a polyhedron is the intersection of a polyhedron with its some supporting hyperplane. A point $v \in \mathcal{P}$ is called a vertex if $\{v\}$ is a face, that is, $\{v\} = \mathcal{P} \cap \{x \in \mathbb{R}^n : w^T x = t\}$ for some $w \in \mathbb{R}^n$ and $t \in \mathbb{R}$.*

We call a polyhedron $\mathcal{P}$ bounded if there exists a vector $b \in \mathbb{R}^n$ such that $\mathcal{P} \subset \{x \in \mathbb{R}^n : -b \leqslant x \leqslant b\}$. A bounded polyhedron is called a *polytope*. We will assume the following propositions which are easy to see and prove.

Proposition 3. *For a polyhedron $\mathcal{P}$,*

1. *a point $v \in \mathbb{R}^n$ is a vertex of $\mathcal{P}$ if and only if v cannot be written as a convex combination of points in $\mathcal{P} \setminus \{v\}$.*

2. *a point $v \in \{x \in \mathbb{R}^n : Ax \leqslant b\}$ is a vertex if and only if there exists a subsystem $A'x \leqslant b'$ of $Ax \leqslant b$ such that $\{v\} = \{x \in \mathbb{R}^n : A'x = b'\}$ and the rank of A' is n.*

3. *A polytope is the convex hull of its vertices.*

4. *If $\mathcal{P}$ is a polytope, then for every vector $w \in \mathbb{R}^{\mathcal{E}}$, there exists a vertex $\overline{x}$ of the polytope $\mathcal{P}$ which maximizes $w^T x$ among points $x \in \mathcal{P}$.*

With this background, we now study the matching polytope. We focus on the perfect matching polytope without loss of generality. This is due to the following reduction from the maximum weight matching problem to the maximum weight perfect matching problem. Let $(\mathcal{G}, w)$ be an arbitrary instance of maximum weight matching. To construct an equivalent instance of maximum weight perfect matching, we take two disjoint copies of $\mathcal{G}$ (along with its edges), call them $\mathcal{G}_1$ and $\mathcal{G}_2$, and an edge between u_1 and u_2 with weight 0 where u_1 and u_2 respectively are the copies of the vertex $u \in \mathcal{G}$ in $\mathcal{G}_1$ and $\mathcal{G}_2$. Let us call the resulting graph $\mathcal{G}'$. Clearly $\mathcal{G}'$ has a perfect matching, namely $\{\{u_1, u_2\} : u \in \mathcal{V}[\mathcal{G}]\}$. It is easy to prove that $\mathcal{G}$ has a matching of weight w if and only if $\mathcal{G}'$ has a perfect matching of weight $2w$. Hence, we have the following result. We denote the set of edges incident on any vertex v by $\delta(v)$.

Theorem 4. *The maximum weight matching problem many-to-one reduces to the maximum weight perfect matching problem.*

The LP relaxation of the natural ILP for the maximum weight perfect matching problem is the following.

$$\max \sum_{e \in \mathcal{E}} w_e x_e$$

$$\text{subject to: } \sum_{e \in \delta(v)} x_e = 1, \forall v \in \mathcal{V} \tag{2.1}$$

$$x_e \geqslant 0, \forall e \in \mathcal{E} \tag{2.2}$$

The perfect matching polytope is the convex hull of the characteristic vectors encoding the perfect matchings of the graph under consideration. The variable vector is $(x_e)_{e \in \mathcal{E}}$. The characteristic vector of a perfect matching $\mathcal{M}$ is $(\mathbb{1}(e \in \mathcal{M}))_{e \in \mathcal{E}}$. We now present Birkhoff's theorem which states that the polytope defined by Equations (2.1) and (2.2) is the perfect matching polytope for bipartite graphs.

Theorem 5 (Birkhoff's Theorem)**.** *Let $\mathcal{G}$ be a bipartite graph. Then the perfect matching polytope of $\mathcal{G}$ is defined by Equations* (2.1) *and* (2.2)*.*

Proof. The characteristic vector of every perfect matching satisfies Equations (2.1) and (2.2). Hence, the perfect matching polytope is contained in the polytope defined by Equations (2.1) and (2.2). For the other direction, let x be a non-integral vertex of the polytope defined by Equations (2.1) and (2.2). We define $F = \{e \in \mathcal{E} : 0 < x_e < 1\} \neq \emptyset$. Since $\sum_{e \in \delta(v)} x_e = 1$, every vertex v that meets some edge in F, the vertex v meets at least two edges in F. Hence, $\mathcal{G}[F]$ has a cycle $\mathcal{C}$ which must be an even cycle since $\mathcal{G}$ is a bipartite graph. Let $(d_e)_{e \in \mathcal{E}}$ be the vector where $d_e = 0$ for every edge not in $\mathcal{C}$ and alternates between 1 and -1 along the edges in $\mathcal{C}$, traversed in any direction. Let us define $\varepsilon = \min\{x_e, 1 - x_e : e \in F\}$.

Then we have both $x + \varepsilon d$ and $x - \varepsilon d$ belongs to the polytope defined by Equations (2.1) and (2.2) and $x = \frac{1}{2}(x+\varepsilon d)+\frac{1}{2}(x-\varepsilon d)$. However, this contradicts out assumption that x is a vertex of the polytope defined by Equations (2.1) and (2.2). □

Theorem 5 clearly needs the assumption that the graph $\mathcal{G}$ is bipartite. This is so because the unique solution of Equations (2.1) and (2.2) for any odd cycle graph is $\left(\frac{1}{2}\right)_{e\in\mathcal{E}}$. For this reason, the polytope defined by Equations (2.1) and (2.2) is called the *fractional matching polytope,* abbreviated as FPM($\mathcal{G}$). Our next result characterizes the vertices of FPM($\mathcal{G}$).

Theorem 6. *Let $\mathcal{G}$ be any graph and x be any point in FPM($\mathcal{G}$). Then x is a vertex of FPM($\mathcal{G}$) if and only if $x_e \in \left\{0, 1, \frac{1}{2}\right\}$ for all $e \in \mathcal{E}$ and the edges with $x_e = \frac{1}{2}$ form vertex disjoint cycles.*

Proof. For the if part, let $x' \in \mathbb{R}^{\mathcal{E}}$ be a point FPM($\mathcal{G}$) such that the edges with $x'_e = \frac{1}{2}$ form vertex disjoint cycles. We define $(w_e)_{e\in\mathcal{E}}$ as

$$w_e = \begin{cases} -1 & \text{if } x_e = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then $\{x'\} = \text{FPM}(\mathcal{G}) \cap \{x \in \mathbb{R}^{\mathcal{E}} : w^T x = 0\}$.

For the only if part, let us first prove that every vertex of FPM($\mathcal{G}$) is half integral. Towards that, we construct a bipartite graph $\mathcal{G}'$ from $\mathcal{G}$ by taking two disjoint copies of $\mathcal{V}[\mathcal{G}]$ (not the edges) and for every edge $\{u, v\} \in \mathcal{E}[\mathcal{G}]$, we add the edges $\{u_1, v_2\}$ and $\{u_2, v_1\}$ in $\mathcal{G}'$ where u_1, u_2, v_1, and v_2 are the copies of the vertices u and v in $\mathcal{G}'$. For any weight vector $w = (w_e)_{e\in\mathcal{E}[\mathcal{G}]}$, we define $w' = (w'_e)_{e\in\mathcal{E}[\mathcal{G}']}$ as $w'(\{u_1, v_2\}) = w'(\{u_2, v_1\}) = w(\{u, v\})$ for every vertex $\{u, v\} \in \mathcal{E}[\mathcal{G}]$ and call it the weight vector for $\mathcal{G}'$ corresponding to the weight vector w. Let $\bar{x} \in \text{FPM}(\mathcal{G})$ be a vertex which is not half-integral. Then there exists a vector $w \in \mathbb{R}^{\mathcal{E}}$ such that $\{\bar{x}\} = \text{argmax}_{x\in\text{FPM}(\mathcal{G})} w^T x := \{x \in \text{FPM}(\mathcal{G}) : w^T x \geqslant w^T y, \forall y \in \text{FPM}(\mathcal{G})\}$. Let w' be the weight vector corresponding to w and $x' \in \text{FPM}(\mathcal{G}')$ be a vertex that maximizes $w'^T x$ among points in FPM($\mathcal{G}'$). From Birkhoff's Theorem, we know that x' is integral. We define $x'' \in \text{FPM}(\mathcal{G})$ as $x''_{\{u,v\}} = \frac{1}{2}\left(x_{\{u_1,v_2\}} + x_{\{u_2,v_1\}}\right)$ for every edge $\{u, v\} \in \mathcal{E}[\mathcal{G}]$. It is easy to prove that x'', as defined above, is a point in FPM($\mathcal{G}$). Since x' is integral, x'' is half-integral. Moreover, we have

$$w^T \bar{x} = w'^T x' = w^T x''.$$

Since $\bar{x}$ is not half-integral but x'' is half-integral, we have $\bar{x} \neq x''$. However, this contradicts our assumption that $\{\bar{x}\} = \text{argmax}_{x\in\text{FPM}(\mathcal{G})} w^T x$. □

Thus Equations (2.1) and (2.2) does not capture the matching polytope if the graph contains odd cycles. We will show next that the following constraints along with Equations (2.1) and (2.2) captures the perfect matching polytope. For any subset $S \subseteq \mathcal{V}$ of vertices, we denote the set of edges with exactly one end point in S by $\delta(S)$.

$$\sum_{e\in\delta(S)} x_e \geqslant 1, \forall S \subset \mathcal{V}, 3 \leqslant |S| \leqslant |\mathcal{V}|, |S| \text{ is an odd integer.} \tag{2.3}$$

Theorem 7. *The perfect matching polytope of any graph $\mathcal{G}$ is defined by Equations (2.3), (2.1) and (2.2).*

Proof. Let $\mathcal{Q}(\mathcal{G})$ be the polytope defined by Equations (2.3), (2.1) and (2.2). The characteristic vector of any perfect matching satisfies Equations (2.3), (2.1) and (2.2). Thus PM($\mathcal{G}$) $\subseteq \mathcal{Q}(\mathcal{G})$.

For the other direction, let G be a graph with the smallest sum of the number of vertices and the number of edges such that $Q(G) \nsubseteq \mathrm{PM}(G)$. Then G must be connected since otherwise there exists a connected component of G which provides a smaller counter-example.

Let x be a vertex of $Q(G)$ which does not belong to $\mathrm{PM}(G)$. We claim that $x_e \in (0,1)$ for every edge $e \in E[G]$. If we have any edge e with $x_e = 0$, then we can delete this edge thereby obtaining a smaller counter-example. If we have any edge with $x_e = 1$, then we can delete this edge and both its end points thereby obtaining a smaller counter example.

Also $|V|$ is an even integer since otherwise $Q(G) = \emptyset$, $\mathrm{PM}(G) = \emptyset$, and thus we have $Q(G) = \mathrm{PM}(G)$. Since $\sum_{e \in \delta(v)} x_e = 1$, the degree of every vertex of G is at least 2. If $|E| = 2|V|$, that is, the degree of every vertex of G is 2, then G is an even cycle since it is connected and $|V|$ is an even integer. However, in this case, G is bipartite and thus $Q(G) = \mathrm{PM}(G)$ thanks to Theorem 5. So let us assume from here on that $|E| > 2|V|$. Since x is a vertex of $Q(G)$, there exists $|E|$ ineuqalities among Equations (2.3), (2.1) and (2.2) which hold with equality for x. Hence, there exists an odd set S with $3 \leqslant |S| \leqslant |V|$ such that $\sum_{e \in \delta(S)} x_e = 1$. By the minimality of G, we have $Q(G/S) = \mathrm{PM}(G/S)$ and $Q(G/(V \setminus S)) = \mathrm{PM}(G/(V \setminus S))$. Let the vector x when restricted to G/S and $G/(V \setminus S)$ be x' and x'' respectively. Hence, x' and x'' can be written as a convex combination of perfect matchings in G/S and $G/(V \setminus S)$ respectively. Since x is a vertex of $Q(G)$, it is rational and thus both x' and x'' are rational. Thus there exists an integer k such that $x' = \frac{1}{k}\sum_{i=1}^{k} \chi(M'_i)$ and $x'' = \frac{1}{k}\sum_{i=1}^{k} \chi(M''_i)$. Note that the integer k is the same in both the expressions. This is so because both x' and x'' assign the same value to every edge in $\delta(S)$.

Let $\delta(S) = \{e_1, \ldots, e_h\}$. Notice that all the edges in $\delta(S)$ are incident on the shrunk vertex in both G/S and $G/(V \setminus S)$. Hence, every matching in $\{M'_i : i \in [k]\}$ and $\{M''_i : i \in [k]\}$ includes exactly one edge from $\delta(S)$. Moreover, we have $\sum_{e \in \delta(S)} x_e = \sum_{e \in \delta(S)} x'_e = \sum_{e \in \delta(S)} x''_e = 1$. Hence, the number of matchings in $\{M'_i : i \in [k]\}$ that include the edge e_j is the same as the number of matchings in $\{M''_i : i \in [k]\}$ that include e_j for every $j \in [h]$. Thus, by renaming, we can assume without loss of generality that M'_i and M''_i have the same edge from $\{e_1, \ldots, e_h\}$. Then $M_i = M'_i \cup M''_i$ is a perfect matching in G for every $i \in [k]$. Thus, we have $x = \frac{1}{k}\sum_{i=1}^{k} \chi(M_i)$ which shows that $x \in \mathrm{PM}(G)$ contradicting our assumption that $Q(G) \nsubseteq \mathrm{PM}(G)$. □

Chapter 3

Min-Cost Flow

In the min-cost flow problem, we are given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where every edge $e \in \mathcal{E}$ has a capacity $u_e \in \mathbb{R}_{>0}$ and cost $c_e \in \mathbb{R}$. Every vertex has a demand $b_v \in \mathbb{R}$. The goal is to compute a flow $x : \mathcal{E} \longrightarrow \mathbb{R}_{\geqslant 0}$ that respects capacity constraint at every edge, meets the demand at every vertex, and minimizes the total cost. These requirements can be encoded by the following linear program.

$$\begin{aligned} \min \sum_{e \in \mathcal{E}} c_e x_e & \\ \text{subject to: } \sum_{wv \in \mathcal{E}} x_{wv} - \sum_{vw \in \mathcal{E}} x_{vw} &= b_v, \forall v \in \mathcal{V} \\ 0 \leqslant x_e &\leqslant u_e, \forall e \in \mathcal{E} \end{aligned}$$

If the capacity of every edge in the min-cost flow problem is infinity, then we call it *transshipment problem*. Interestingly, the min-cost flow problem reduces to the transshipment problem. The reduction is shown in Figure 3.1. The correctness of the reduction follows from the fact that any edge vw of a min-cost flow solution carries a flow of x if and only if the edge vt (and sw) of the corresponding transshipment instance solution carries a flow of x. We also observe that the integral feasible solutions of min-cost flow are in one-to-one correspondence with the integral feasible solutions of the reduced transshipment instance.

Theorem 8. *The min-cost flow problem polynomial-time many-to-one reduces to the transshipment problem. Moreover, the integral solutions of the min-cost flow instance are in one-to-one correspondence with the integral solutions of the reduced transshipment instance. Any algorithm for the transshipment problem with runtime* $T(m, n, b, c)$ *can be used to obtain an algorithm for the min-cost flow problem with runtime* $\mathcal{O}(T(m, n, b, c, u))$.

Thanks to Theorem 8, we focus our attention on the transshipment problem from now on.

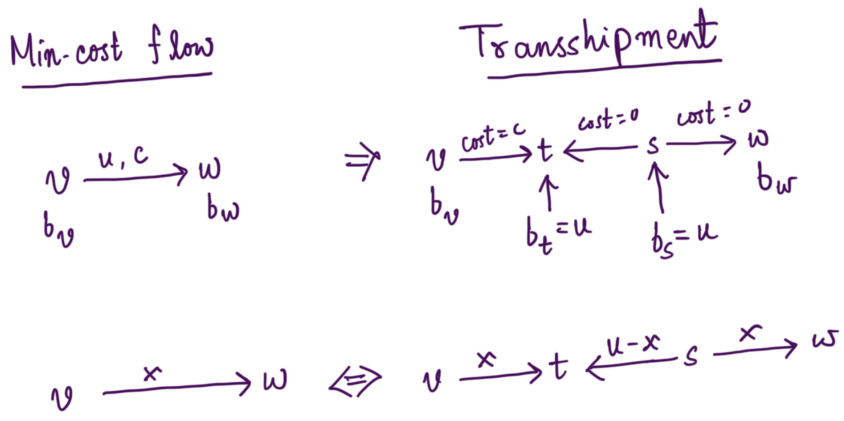

Figure 3.1: Reduction from min-cost flow to transshipment.

The linear program for the transshipment problem is the following.

$$\begin{aligned} \min \sum_{e \in \mathcal{E}} c_e x_e \\ \text{subject to: } \sum_{wv \in \mathcal{E}} x_{wv} - \sum_{vw \in \mathcal{E}} x_{vw} = b_v, \forall v \in \mathcal{V} \\ x_e \geqslant 0, \forall e \in \mathcal{E} \end{aligned}$$

The dual of the above linear program is the following.

$$\begin{aligned} \max \sum_{v \in \mathcal{V}} b_v y_v \\ \text{subject to: } y_w - y_v \leqslant c_{vw}, \forall vw \in \mathcal{E} \end{aligned}$$

The condition $y_w - y_v \geqslant c_{vw}$ is equivalent to $c_{vw} + y_v - y_w \leqslant 0$. We define $\bar{c}_{vw}$ to be $c_{vw} + y_v - y_w$ and call it the *reduced cost of edge* vw. Hence, the dual constraints are $\bar{c}_{vw} \geqslant 0$ for every edge vw of the input graph $\mathcal{G}$. Given a primal solution x, we want to find conditions for the optimality of x. The complementary slackness conditions will give us the optimality criteria which are as follows.

$$\forall vw \in \mathcal{E}[\mathcal{G}], x_{vw} > 0 \Longrightarrow \bar{c}_{vw} = 0$$

We call an edge $vw \in \mathcal{E}[\mathcal{G}]$ *tight* with respect to a dual solution y if $\bar{c}_{vw} = 0$. Hence, we have proved the following.

Lemma 1. *A primal feasible solution x is optimal if and only if there exists a dual solution y such that every edge that carries any flow must be tight. That is*

$$vw \in \mathcal{E}[\mathcal{G}], x_{vw} > 0 \Longrightarrow \bar{c}_{vw} = 0.$$

Moreover, every pair (x, y) of optimal primal and dual solutions satisfies the above conditions.

We define the residual graph $\mathcal{G}(x)$ for a given network $\mathcal{G}$ and flow x as follows. For every edge $vw \in \mathcal{E}[\mathcal{G}]$, there is an edge $vw \in \mathcal{E}[\mathcal{G}(x)]$ in the residual graph with cost c_{vw} and capacity ∞. These edges are called

forward edges. For every edge $vw \in \mathcal{E}[\mathcal{G}]$ with $x_{vw} > 0$, there is an edge $wv \in \mathcal{E}[\mathcal{G}(x)]$ in the residual graph with cost $-c_{wv}$ and capacity x_{vw}. We denote the cost of any edge $e \in \mathcal{E}[\mathcal{G}(x)]$ in the residual graph by c'_e. We now give an equivalent characterization for the optimality of a primal solution x.

Lemma 2. *A feasible primal solution x is optimal if and only if there is no negative cost cycle in $\mathcal{G}(x)$.*

Proof. Clearly, if there is a negative cost cycle $\mathcal{C}$ in $\mathcal{G}(x)$, then we can use $\mathcal{C}$ to have another feasible primal solution with cost less than x.

For the other direction, let us assume that there is no negative cost cycle in $\mathcal{G}(x)$. We will prove that x is optimal. Towards that, we introduce a new vertex r in $\mathcal{G}(x)$ and add an edge from r to v for every vertex $v \in \mathcal{G}(x)$ with cost 0. Let us call this resulting graph $\mathcal{G}'(x)$. We define y_v to be the distance from r to v in $\mathcal{G}'(x)$. This is well-defined since $\mathcal{G}(x)$ does not have any negative cost cycle. Let $vw \in \mathcal{E}[\mathcal{G}]$ be any edge of $\mathcal{G}$ with $x_{vw} > 0$. Then we have $wv \in \mathcal{E}[\mathcal{G}(x)]$ with $c'_{wv} = -c_{vw}$. Hence, we have the following.

$$y_w - c_{vw} \geqslant y_v \Rightarrow \bar{c}_{vw} \leqslant 0$$

We also have a forward edge $vw \in \mathcal{E}[\mathcal{G}(x)]$ with cost c_{vw}. Hence, we have the following.

$$y_v + c_{vw} \geqslant y_w \Rightarrow \bar{c}_{vw} \geqslant 0$$

Hence, we have $\bar{c}_{vw} = 0$ which proves optimality of x thanks to Lemma 1. □

If the primal has an optimal solution and the cost of every edge is integral, then the distances in $\mathcal{G}'(x)$, defined in the proof of Lemma 2, are also integral. This proves the following.

Theorem 9. *If the primal has an optimal solution and the cost c is integral, then the dual has an integral optimal solution.*

We now characterize existence of an optimal primal solution.

Theorem 10. *Suppose the primal is feasible. Then the primal has an optimal solution if and only if there is no negative cost cycle in $\mathcal{G}$.*

Proof. Clearly, if there exists a negative cost cycle in $\mathcal{G}$, then there is no primal solution. For the other direction, let us assume that $\mathcal{G}$ does not have any negative cost cycle and the primal is feasible. We will prove that the primal has an optimal solution. For that, it is enough to show dual feasibility. Towards that, we introduce a new vertex r in $\mathcal{G}$ and add an edge from r to v for every vertex $v \in \mathcal{G}$ with cost 0. Let us call this resulting graph $\mathcal{G}'$. We define y_v to be the distance from r to v in $\mathcal{G}'$. Then, for any edge $vw \in \mathcal{E}[\mathcal{G}]$, we have the following

$$y_v + c_{vw} \geqslant y_w \Rightarrow \bar{c}_{vw} \geqslant 0$$

that proves dual feasibility. □

Thanks to Theorem 10 and the fact that the primal feasibility can be checked in polynomial time by using any maximum flow algorithm, we will assume from now on that the transshipment instance we are working on is feasible and has an optimal solution.

3.1 Primal Algorithm

We now present our first algorithm for the transshipment problem. And the algorithm is immediate from Lemma 2. That is, we start with any feasible solution x. If $\mathcal{G}(x)$ has any negative cost cycle $\mathcal{C}$, then we augment x using $\mathcal{C}$ thereby obtaining another feasible solution with cost lower than x. If the demand b is integral, then we know that there exists an integral feasible solution which we can compute using any standard maximum flow algorithm, for example, Edmond-Karp or push-relabel. We notice that above iterative algorithm maintains integrality of the solution. Hence, we have the following.

Theorem 11. *If the demand is integral, then there exists an integral optimal solution for the transshipment problem and thus for min-cost flow problem.*

This algorithm follows the algorithm design technique called primal algorithm. Here we begin with a feasible primal solution and iteratively work towards optimality.

We now analyze the run time of the algorithm. We assume that the demand is integral which guarantees termination of our algorithm. The cost of the initial solution obtained using, say Edmond-Karp algorithm, is at most $\mathcal{O}\left(\left(\sum_{e\in\mathcal{E}[\mathcal{G}]}|c_e|\right)\cdot\left(\sum_{v\in\mathcal{V}}|b_v|\right)\right)$. Hence, the runtime is $\mathcal{O}\left(\left|\left(\sum_{e\in\mathcal{E}[\mathcal{G}]}|c_e|\right)\cdot\left(\sum_{v\in\mathcal{V}}|b_v|\right)-\mathrm{OPT}\right|\cdot(m+n)+n^3\right)$ which is not polynomial of the input size. We next see another algorithm which has better run time.

3.2 Primal-Dual Algorithm

In the algorithm design framework of primal-dual algorithms, we maintain a primal "infeasible solution" x and a dual feasible solution y such that (x,y) pair satisfy the optimality condition. For the transshipment problem, Lemma 2 provides the optimality condition. Then we iteratively make the primal infeasible solution x "more and more feasible." The algorithm terminates when the primal solution x is feasible.

So the first step of our primal-dual algorithm is to pick a primal "infeasible solution" x and a dual feasible solution y such that (x,y) pair satisfy the optimality condition. The optimality criteria for the transshipment problem is that only those edges having zero reduced cost can carry flow. We choose x to be zero (that is, all zero vector). We note that x is infeasible because it may not meet the demand requirements of all the vertices. For this x, we choose y as in the proof of Lemma 2. Hence, (x,y) satisfy the optimality criteria. We now try to meet the demands of the vertices iteratively. The algorithm terminates when the demand of all the vertices are met.

We call a vertex r an x-source if $b_v < \sum_{wv\in\mathcal{E}[\mathcal{G}]}x_{wv}-\sum_{vw\in\mathcal{E}[\mathcal{G}]}x_{vw}$. Similarly, we call a vertex x-sink if $b_v > \sum_{wv\in\mathcal{E}[\mathcal{G}]}x_{wv}-\sum_{vw\in\mathcal{E}[\mathcal{G}]}x_{vw}$. In every iteration of our algorithm, we send some amount of flow from some x-source to some x-sink. Towards that, we again consider the residual graph $G(x)$ but this time the edges are weighted with the reduced costs. That is, if $vw\in\mathcal{E}[\mathcal{G}(x)]$ is a forward edge, then its cost is $\bar{c}_{vw}$; if it is a backward edge, then its cost is $-\bar{c}_{vw}$. Let us call this graph $\mathcal{G}'(x)$. Since (x,y) satisfies optimality conditions, the weight of every edge in $\mathcal{G}'(x)$ is non-negative. For every vertex $v\in\mathcal{V}$, we define σ_v to be the distance of v from the set of x-source vertices; that is σ_v is the minimum distance of v from any x-source vertex. Note that, σ_v will be ∞ if the vertex v is unreachable from every x-source vertices. Also notice that the distance function is well-defined because the cost of every edge of $\mathcal{G}'(x)$ is non-negative.

We notice that if we augment flow along with any path from some x-source to some x-sink that uses only zero-cost edge, then the updated x and the current y together satisfy optimality criteria. In general, we pick

an x-sink s with the smallest σ_s value; let P_s be the corresponding shortest path. We augment x using the augmenting path P_s; let x' be the new flow. We define $y'_v = y_v + \min\{\sigma_v, \sigma_s\}$. We claim that (x', y') also satisfies the optimality conditions. Let $\bar{c}'$ be the new reduced cost. That is, for every edge $vw \in \mathcal{E}[\mathcal{G}]$, if $x'_{vw} > 0$, then $\bar{c}'_{vw} = 0$.

Claim 1. *For every edge $vw \in \mathcal{E}[\mathcal{G}]$, if $x'_{vw} > 0$, then $\bar{c}'_{vw} = 0$.*

Proof. If $vw \in P_s$, then $\sigma_v \leqslant \sigma_s, \sigma_w \leqslant \sigma_s, \sigma_v + \bar{c}_{vw} = \sigma_w, y'_v = y_v + \sigma_v, y'_w = y_w + \sigma_w$. Then we have the following.

$$\begin{aligned}
\bar{c}'_{vw} &= c_{vw} + y'_v - y'_w \\
&= c_{vw} + y_v - y_w + \min\{\sigma_v, \sigma_s\} - \min\{\sigma_w, \sigma_s\} \\
&= \bar{c}_{vw} + \sigma_v - \sigma_s \\
&= 0
\end{aligned}$$

Let us assume now that $vw \notin P_s$. Then $x'_{vw} > 0 \Rightarrow x_{vw} > 0 \Rightarrow \bar{c}_{vw} = 0$. Also, both the edges vw and wv belongs to $\mathcal{G}'(x)$. Hence, we have $\sigma_v = \sigma_w$. Then we have the following.

$$\begin{aligned}
\bar{c}'_{vw} &= c_{vw} + y'_v - y'_w \\
&= c_{vw} + y_v - y_w + \min\{\sigma_v, \sigma_s\} - \min\{\sigma_w, \sigma_s\} \\
&= 0
\end{aligned}$$

□

Since we send some amount of flow from some x-source to some x-sink in every iteration, our algorithm terminates if the demand vector is integral which we assume from now on. Moreover, the number of iterations is at most $\mathcal{O}\left(\sum_{v \in \mathcal{V}} |b_v - f_{x_0}(v)|\right)$ where $f_{x_0}(v)$ is the flow into v minus the flow out v in the initial solution x_0 and thus the overall runtime is $\mathcal{O}\left(n \log n \sum_{v \in \mathcal{V}} |b_v - f_{x_0}(v)|\right) = \mathcal{O}\left(n \log n \sum_{v \in \mathcal{V}} |b_v|\right)$. Hence, the runtime is pseudo-polynomial in the input size.

This algorithm is called the primal-dual algorithm with least cost augmenting paths since we use a least cost path to augment the current flow.

3.3 Dual Scaling Algorithm

We now use a scaling technique and use the primal-dual algorithm described above to obtain a polynomial-time algorithm. For that we assume that there exists a vertex $r \in \mathcal{V}[\mathcal{G}]$ with $b_r = 0, rv \notin \mathcal{E}[\mathcal{G}], vr \in \mathcal{E}[\mathcal{G}], c_{rv} = 0$ for every $v \in \mathcal{V}[\mathcal{G}] \setminus \{r\}$; that is r has only incoming edges from all other vertices. We can ensure this by adding a new vertex r and adding an edge from r to every other vertex with cost zero. Any feasible solution x of the original instance can be extended to a solution in the modified instance by setting $x_{vr} = 0$ for every original vertex v and vice versa.

Scaling a transshipment instance $(\mathcal{G}, c, b)$ by a factor Δ is defined to be the instance $(\mathcal{G}, c, b')$ where $b'(v) = \left\lfloor \frac{b_v}{\Delta} \right\rfloor$ for every vertex $v \in \mathcal{V} \setminus \{r\}$ and $b'(r) = -\sum_{v \in \mathcal{V} \setminus \{r\}} b'(v)$. The following lemma can be proved using max-flow min-cut theorem which characterizes feasibility of a flow instance. For $S \subseteq \mathcal{V}$, we define $b(S) = \sum_{v \in S} b_v$.

Lemma 3. *A min-cost flow instance $(\mathcal{G}, c, b)$ is feasible if and only if (i) $b(V) = 0$ and (ii) $b(S) \leqslant 0$ for every subset $S \subset V$ such that there is no incoming edge to S.*

We now show that the scaled instance has an optimal solution if the original instance has an optimal solution.

Lemma 4. *If $(\mathcal{G}, c, b)$ has an optimal solution, then the scaled instance $(\mathcal{G}, c, b')$ has an optimal solution for every scaling factor $\Delta > 0$.*

Proof. Since both the instances have the same cost function and there is no negative cost cycle in the original instance, there is no negative cost cycle in the scaled down instance. Hence, we only need to prove feasibility of the scaled down instance to prove existence of an optimal solution in the scaled down instance. Clearly, we have $b'(V) = 0$. Let $S \subset V$ such that there is no incoming edge to S. Since there is no incoming edge to S, $r \notin S$. We now have

$$b'(S) = \sum_{v \in S} \left\lfloor \frac{b_v}{\Delta} \right\rfloor \leqslant \frac{1}{\Delta} \sum_{v \in S} b_v \leqslant 0$$

where the last inequality follows from the feasibility of the original instance and Lemma 3. □

We now describe our algorithm. This is the first polynomial time algorithm for the min-cost flow problem thanks to Edmond and Karp. For a non-negative integer k, we denote the scaled instance by the scaling factor 2^k by $(\mathcal{G}, c, b^k)$. We note that $(\mathcal{G}, c, b^0)$ is the original input instance. The idea is to use an optimal primal-dual solution pair of the instance $(\mathcal{G}, c, b^{k+1})$ to construct an initial solution pair for the primal-dual algorithm for the instance $(\mathcal{G}, c, b^k)$ which is close to optimal thereby guaranteeing that the primal-dual algorithm does not take too much time to find an optimal solution for $(\mathcal{G}, c, b^k)$. Concretely, we have the following lemma whose proof is immediate from the fact that both the instances has the same cost functions and $y' = y$.

Lemma 5. *Let (x, y) be a primal and dual pair of optimal solutions for $(\mathcal{G}, c, b^{k+1})$. Then $(x', y') = (2x, y)$ is a pair of primal (possibly infeasible) solution and dual feasible solution that satisfies the optimality criteria, that is, $\forall e \in \mathcal{E}[\mathcal{G}], x'_e > 0 \Rightarrow \overline{c}'_e = 0$.*

Hence, (x', y') is a valid initial pair of solutions for the primal-dual algorithm for $(\mathcal{G}, c, b^k)$. Moreover, we have $b^k_v = 2b^{k+1}_v$ or $b^k_v = 2b^{k+1}_v + 1$ for every $v \in \mathcal{V} \setminus \{r\}$ and $b^k_r \leqslant 2b^{k+1}_r$. Hence, we have $\sum_{v \in \mathcal{V}} |b_v - f_{x'}(v)| = \mathcal{O}(n)$ and thus the primal-dual algorithm for $(\mathcal{G}, c, b^{k+1})$ with starting solution pair (x', y') makes $\mathcal{O}(n)$ iterations. We now explain our full algorithm.

Let K be the smallest integer such that $b_v \leqslant 2^K$ for every vertex v. We observe $(\mathcal{G}, c, b^K)$ has total demand at most n and thus the primal-dual algorithm on this instance take at most $\mathcal{O}(n)$ time. From the optimal solution pair for $(\mathcal{G}, c, b^K)$, we construct a valid solution pair for $(\mathcal{G}, c, b^{K-1})$ and use the primal-dual algorithm again to compute an optimal solution pair for $(\mathcal{G}, c, b^{K-1})$ which, as we have explained, takes $\mathcal{O}(n)$ iterations of the primal-dual algorithm and thus takes $\mathcal{O}(n^2 \log n)$ time. We repeat this process to obtain an optimal solution for $(\mathcal{G}, c, b^0)$ which is the input instance. Hence, the depth of the recursion tree is $\mathcal{O}(n \log B)$ where $B = \sum_{v \in V} |b_v|$. This gives an overall run time of $\mathcal{O}(n^2 \log n \log B)$ which is polynomial in the input size.

Bibliography