5 Network flows

Many problems can be formalized as *optimization* problems: among all possible solutions of a problem, find one which minimizes or maximizes a certain cost. For example, in Chapter 4 we have seen algorithms that find the shortest path between two vertices of a graph, among all possible such paths. Another example is a *bipartite matching* problem, in which we have to match the elements of one set with the elements of another set, assuming that each allowed match has a known cost. Among all possible ways of matching the two sets of elements, one is interested in a matching with minimum total cost.

In this chapter we show that many optimization problems can be *reduced* (in the sense already explained in Section 2.3) to a *network flow* problem. This polynomially solvable problem is a powerful model, which has found a remarkable array of applications. Roughly stated, in a network flow problem one is given a transportation network, and is required to find the optimal way of sending some content through this network. Finding a shortest path is a very simple instance of a network flow problem, and, even though this is not immediately apparent, so is the bipartite matching problem.

One of the most well-known network flow problems is the *maximum flow* problem (whose definition we recall on page 46). In this chapter we focus instead on a more general version of it, called the *minimum-cost flow* problem. In Section 5.1 we give an intuition of flow, and we show that, in a DAG, a flow is just a collection of paths. This basic observation will allow us to reduce the assembly problems in Chapter 15 to minimum-cost flow problems. In Sections 5.3 and 5.4 we will discuss matching and covering problems solvable in terms of minimum-cost flow, which will in their turn find later applications in the book.

We must emphasize that algorithms tailored for a particular optimization problem can have a better complexity than the solution derived from a network flow algorithm. Nevertheless, a practitioner may prefer a quickly obtainable and implementable solution, given the existing body of efficient network flow solvers. Alternatively, a theoretician may first show that a problem is solvable in polynomial time by a reduction to a network flow problem, and only afterwards start the quest for more efficient solutions.

5.1 Flows and their decompositions

In this chapter we make the assumption that the directed graphs, be they acyclic or not, have exactly one source and one sink, which will be denoted s and t, respectively.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

A path from s to t will be called an *s-t path*. Recall that all paths and cycles are directed.

For an intuition on what a minimum-cost flow problem is, consider a DAG with a cost associated with every arc. Suppose we want to find the optimal way of sending some content that continuously "flows" along the arcs of the DAG, from its source s to its sink t. If by optimal we mean that the cost of a possible solution is the sum of the costs of the arcs traversed by our content, then the optimal solution is just the shortest s-t path, in which costs are taken as lengths, the problem which we already solved in Section 4.1.2. Suppose, however, that each arc also has a *capacity*, that is, a limit on how much content it can be traversed by. This slight generalization renders it no longer solvable in terms of just one shortest-path problem, since our content must now flow along different s-t paths.

This brings us to the most basic meaning of *flow* in a DAG: a collection of s-t paths. Solving a minimum-cost flow problem amounts to solving simultaneously many shortest-path-like problems, tied together by different constraints, such as capacity constraints.

In this section we make the simplifying assumption that, for any two vertices x and y, if the arc (x, y) is present in the graph, then the reverse arc (y, x) is not present. Otherwise, we can simply *subdivide* (y, x), that its, we can remove (y, x), and then add a new vertex z together with the two arcs (y, z) and (z, x). Unless stated otherwise, graphs have n vertices and m arcs.

It will be convenient to regard a flow as a function assigning to every arc of the graph the amount of content it is being traversed by; we will call this content *the flow* of an arc. Clearly, a flow function arising from a collection of paths must satisfy the following condition:

> Flow conservation. The flow entering a vertex is equal to the flow exiting the vertex, with the exception of s and t.

Note that, if the flow conservation property holds, then the flow exiting s equals the flow entering t (Exercise 5.1). We can now give a formal definition of flow in a directed graph.

DEFINITION 5.1 *Given a directed graph G, a* flow *on G is a function $f : E(G) \to \mathbb{Q}_+$ satisfying the flow conservation property.*

Theorem 5.2 below shows that, in a DAG, collections of s-t paths and functions assigning flow values to the arcs of the DAG and satisfying the flow conservation property are equivalent notions. Namely, any flow in a DAG can be decomposed into a collection of weighted s-t paths, such that the flow on an arc equals the sum of the weights of the s-t paths using that arc (see Figure 5.1).

THEOREM 5.2 *Let f be a flow in DAG G. In time $O(nm)$ we can find $k \leq m$ s-t paths $P_1, \ldots, P_k$, each P_i having a weight $w_i > 0$, such that for every arc $(x, y) \in E(G)$ it holds that*

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

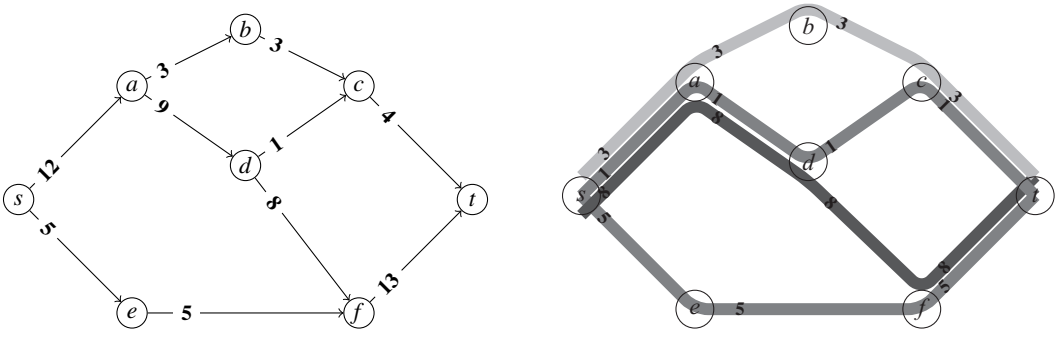


Figure 5.1 On the left, a flow f on a DAG G. On the right, the decomposition of f into four paths, (s, a, b, c, t), (s, a, d, c, t), (s, a, d, f, t), and (s, e, f, t), having weights 3, 1, 8, and 5, respectively.

$$f(x, y) = \sum_{P_i : (x,y) \in P_i} w_i.$$

Moreover, if f has only integer values, then the computed $w_1, \ldots, w_k$ are integers.

Proof At step i of the algorithm, we select an s-t path P_i and take its weight w_i to be the minimum flow value on its arcs (the value w_i is also called the *bottleneck* of P_i). We then remove P_i from G, in the sense that we subtract w_i from the flow value of all arcs of P_i, which results in a new flow function (that is, still satisfying the flow conservation property). If P_i is chosen such that its bottleneck is positive, then at least one more arc of G has flow value 0. Therefore, we can decompose any flow in G into $k \leq m$ paths.

In order to compute each P_i in $O(n)$ time, at the start of the algorithm we can remove the arcs of G with null flow value. As long as G has at least one arc, by the flow conservation property there exists an s-t path P, which we can find by a visit of G starting from s (since G is acyclic this visit must end in t). When we subtract w_i from the flow value of an arc on P_i, we also check whether its flow value becomes null, in which case we remove it from G. Since for every vertex in P_i we subtract w_i both from its in-coming flow and from its out-going flow, the result is still a flow function.

The fact that $w_1, \ldots, w_k$ are integers if f is integer-valued follows from the fact that at every step we are subtracting an integer value from an integer-valued flow. □

The proof of Theorem 5.2 can be extended to show that if a flow f is in a graph that is not necessarily acyclic, then f can be decomposed as a collection of at most m weighted paths or cycles (Exercise 5.3). See also Insight 5.1. In practice, we might be interested in decomposing a flow into the minimum number of components. However this is an NP-hard problem, as our next theorem shows.

Insight 5.1 Decomposing a flow in a DAG into a few paths in practice

One heuristic for decomposing a flow f in a DAG G into a few paths is to use the same strategy as in the proof of Theorem 5.2, but at each step to select

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

the *s-t* path of maximum *bottleneck*, which path can be computed in time $O(m)$ (Exercise 5.5).

If an approximate solution is satisfactory, then one can decompose into a few paths only a fraction of the flow. This idea is based on the observation that, if f has value q, and the values of f on all arcs are multiples of an integer x, then f can be decomposed into exactly q/x paths each of weight x (Exercise 5.2). Therefore, one can round all values of a flow to the nearest multiple of an appropriately chosen integer, and then decompose the resulting flow. For example, through appropriate choices, it can be shown that, if the minimum number of paths into which a flow f of value q can be decomposed is k, then one can decompose $1/3$ of f into at most k paths in polynomial time; see the literature section.

THEOREM 5.3 *Given a flow f in a DAG G, the problem of decomposing f into the minimum number of weighted s-t paths is NP-hard.*

Proof We reduce from the NP-hard subset sum problem, in which we are given a set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers, together with positive integers b and k, and we are asked whether there exists a subset $S \subseteq A$ such that $|S| \leq k$ and $\sum_{a \in S} a = b$. We denote by $\overline{b}$ the complementary value $\left(\sum_{a \in A} a\right) - b$.

For an instance (A, b, k) of the subset sum problem, we construct the DAG $G_{A,b}$, and the flow f in $G_{A,b}$, as follows (see also Figure 5.2):

- $V(G_{A,b}) = \{s, x_1, \ldots, x_n, y, z_1, z_2, t\}$;
- for every $i \in \{1, \ldots, n\}$, we add the arcs (s, x_i) and (x_i, y) to $G_{A,b}$, setting $f(s, x_i) = f(x_i, y) = a_i$;
- we add the arcs (y, z_1) and (z_1, t) to $G_{A,b}$, setting $f(y, z_1) = f(z_1, t) = b$; and
- we add the arcs (y, z_2) and (z_2, t) to $G_{A,b}$, setting $f(y, z_2) = f(z_2, t) = \overline{b}$.

We show that the subset sum problem admits a solution to an input (A, b, k) if and only if the flow f in $G_{A,b}$ can be decomposed into n weighted paths.

For the forward implication, let S be a solution to an input (A, b, k). For $i \in \{1, \ldots, n\}$, set

$$P_i = \begin{cases} s, x_i, y, z_1, t, & \text{if } a_i \in S, \\ s, x_i, y, z_2, t, & \text{if } a_i \notin S. \end{cases}$$

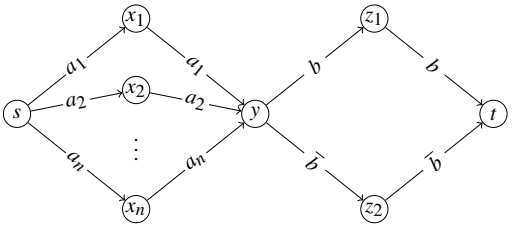


Figure 5.2 A reduction of the subset sum problem to the problem of splitting a flow in a DAG into the minimum number of weighted paths.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

It is immediately seen that the flow f can be decomposed into the n paths $P_1, \ldots, P_n$, each P_i having weight $w_i = a_i$.

For the backward implication, let $P_1, \ldots, P_n$, each P_i having weight w_i, be a decomposition of the flow f into n paths. Observe first that, since each of the n arcs outgoing from s must be contained in at least one of the n paths, each such arc is contained in exactly one path in $\{P_1, \ldots, P_n\}$. Thus, $\{w_1, \ldots, w_n\} = \{a_1, \ldots, a_n\}$. Since the flow on the arc (y, z_1) is completely decomposed into a subset of the n paths, the solution S for the input (A, b, k) subset sum is given by the weights of the paths using the arc (y, z_1). Namely,

$$S = \{a_i \mid \text{the path } P \in \{P_1, \ldots, P_n\} \text{ having weight } a_i \text{ uses } (y, z_1)\}. \qquad \square$$

5.2 Minimum-cost flows and circulations

In this section we introduce the notion of a flow network, by enriching arcs with capacities. We make one further generalization by allowing the arcs to have also *demands*, namely, a lower bound on the amount of flow they must be traversed by. It is in fact the interplay among capacities, demands, and costs that renders network flows such a powerful model.

DEFINITION 5.4 *A* flow network *is a tuple $N = (G, \ell, u, c, q)$, where*

- *G is a directed graph (with the unique source s and the unique sink t);*
- *ℓ and u are functions assigning a non-negative* demand *and* capacity*, respectively, to every arc;*
- *c is a function assigning a* cost per unit of flow *to every arc;*
- *q is the required* value *of the flow.*

A flow over a flow network is a function satisfying the flow conservation property, the demand and capacity constraints, and having a given value.

DEFINITION 5.5 *Given a flow network $N = (G, \ell, u, c, q)$, a function $f : E(G) \to \mathbb{Q}_+$ is a* flow *over the network N if the following conditions hold.*

- ***Flow conservation**: for every vertex $x \in V(G) \setminus \{s, t\}$, it holds that*

$$\sum_{y \in N^-(x)} f(y, x) = \sum_{y \in N^+(x)} f(x, y).$$

- ***Demand and capacity constraints**: for every arc $(x, y) \in E(G)$, it holds that*

$$\ell(x, y) \leq f(x, y) \leq u(x, y).$$

- ***Required flow value** q:*

$$\sum_{y \in N^+(s)} f(s, y) = \sum_{y \in N^-(t)} f(y, t) = q.$$

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

Even though this is not explicitly stated, some arcs can have unlimited capacity, in which case we say that their capacity is ∞, or that they do not have a capacity. Exercise 5.6 asks the reader to show that any such network can be reduced to one in which all arcs are assigned a finite capacity. Note also that some ill-constructed flow networks may not admit any flow defined as in Definition 5.5, because of ill-posed demand or capacity constraints, or because the required flow value is impossible to attain given the demands and capacities of the arcs. A flow satisfying the demand and capacity constraints will also be called *feasible*.

In the minimum-cost flow problem, given a flow network, over all its feasible flows, we have to find the one which minimizes the total cost of sending the required flow value q from s to t.

Problem 5.1 Minimum-cost flow

Given a flow network $N = (G, \ell, u, c, q)$, find a flow f over N that minimizes

$$\mathsf{cost}(f) = \sum_{(x,y)\in E(G)} c(x,y)f(x,y).$$

In order to illustrate how a minimum-cost flow problem can be solved, we must introduce an apparent generalization, one that allows a concise statement of its solution. A *circulation* over a directed graph G is a function $f : E(G) \to \mathbb{Q}$ satisfying the flow conservation property *for every vertex*, and possibly taking negative values. Obviously, circulations must be defined only over directed graphs *without* sources or sinks. As such, a *circulation network* is a tuple $N = (G, \ell, u, c)$, where G is now a directed graph without sources or sinks and ℓ, u, and c have the same meanings as they do for flow networks. The *minimum-cost circulation* over N is a circulation $f : E(G) \to \mathbb{Q}$, satisfying the demand and capacity constraints of N (which can now take negative values), and minimizing $\mathsf{cost}(f)$.

The minimum-cost flow problem over a flow network $N = (G, \ell, u, c, q)$ can be solved by a minimum-cost circulation problem by adding to G an arc from t to s with $\ell(t,s) = u(t,s) = q$ and $c(s,t) = 0$. In fact, these two problems are equivalent, since also a minimum-cost circulation problem can be solved in terms of a minimum-cost flow problem (Exercise 5.7).

Moreover, both the maximum flow problem and the shortest s-t path problem can be easily reduced to a minimum-cost circulation problem. In a *maximum flow problem*, one is given a flow network N with capacities only (that is, without demands and costs), and is required to find a flow of maximum value. This can be reduced to a minimum-cost circulation problem by adding, as above, the arc (t,s) with cost -1 and infinite capacity, and setting the cost of all other arcs to 0.

A shortest s-t path (recall Problem 4.1 and 4.3) can be obtained in a minimum-cost flow problem by setting the required flow value to 1, and keeping the same costs as in

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

the original graph. Note that, in a case with multiple shortest s-t paths, one minimum-cost flow on this network may consist of multiple paths with sub-unitary weights which sum up to 1. However, we will later see in Corollary 5.10 that, whenever the demands, capacities, and flow value are integer-valued, an integer-valued minimum-cost flow can always be found.

5.2.1 The residual graph

Given a circulation f over a network $N = (G, \ell, u, c)$, we first show a condition guaranteeing that f can be transformed into another circulation over N of strictly smaller cost. This will be referred to as the *residual graph* $\mathsf{R}(f)$ of the circulation f, namely a directed graph having the same vertices as G, and in which, for every arc (x, y) of G, we do the following.

- We add the "increase" arc (x, y) to $\mathsf{R}(f)$, if the value of the circulation on (x, y) can be *increased* while satisfying the capacity of (x, y). This increase *adds* $c(x, y)$ to the cost of the resulting circulation.
- We add the "decrease" arc (y, x) to $\mathsf{R}(f)$, if the value of the circulation on (x, y) can be *decreased* while satisfying the demand of (x, y). This decrease *subtracts* $c(x, y)$ from the cost of the resulting circulation.

A circulation network is shown in Figure 5.3(a) and its residual graph in Figure 5.3(b). Formally, we have the following definition:

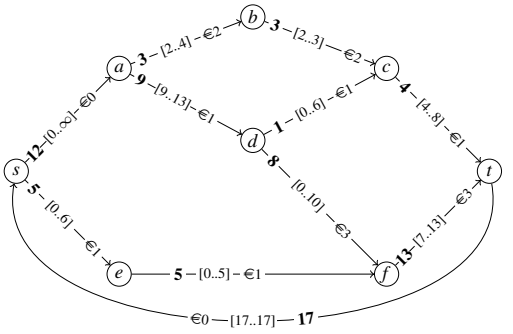
DEFINITION 5.6 *The* residual graph $\mathsf{R}(f)$ *of the circulation* f *is a directed graph with* $V(\mathsf{R}(f)) = V(G)$*, in which each arc is associated with the value by which the circulation can be increased or decreased, called its* residue*, and a cost. For every arc* $(x, y) \in E(G)$*:*

- *if* $f(x, y) < u(x, y)$*, the arc* (x, y) *is added to* $\mathsf{R}(f)$ *with residue* $r(x, y) = u(x, y) - f(x, y)$ *and cost* $c(x, y)$*; and*
- *if* $\ell(x, y) < f(x, y)$*, the reverse arc* (y, x) *is added to* $\mathsf{R}(f)$ *with residue* $r(y, x) = f(x, y) - \ell(x, y)$ *and cost* $-c(x, y)$*.*

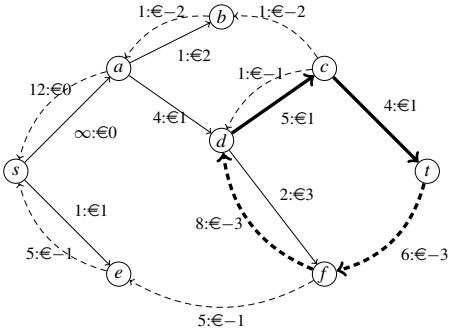
The residual graph of a circulation f is a compact representation of the ways in which f can be altered, since each of its arcs encodes a possible increase or decrease of the value of f, with the corresponding cost adjustments. However, only proper cycles in $\mathsf{R}(f)$ (that is, cycles without repeated vertices apart from their first and last elements, and of length at least 3) lead to adjustments of f that satisfy the flow conservation property. A cycle C in $\mathsf{R}(f)$ gives a way of transforming f into another circulation f_C, as follows:

- let $b(C) > 0$ be the minimum residue of the arcs of C (the bottleneck of C);
- if C uses an "increase" arc (x, y), where $(x, y) \in E(G)$, then $f_C(x, y) = f(x, y) + b(C)$; and
- if C uses a "decrease" arc (y, x), where $(x, y) \in E(G)$, then $f_C(x, y) = f(x, y) - b(C)$.

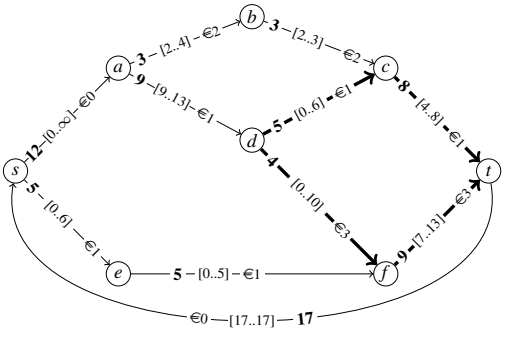
https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press



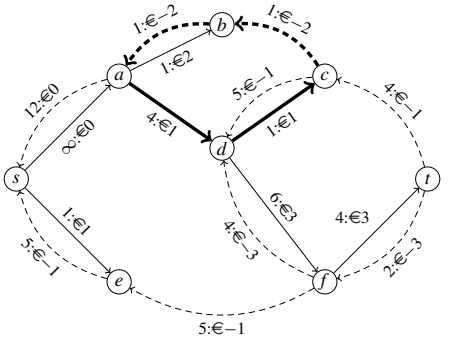
(a) A circulation network (G, ℓ, u, c) and a circulation f



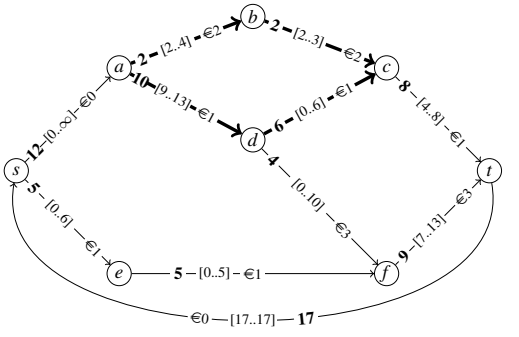
(b) The residual graph $\mathsf{R}(f)$ of f; cycle $C_1 = t, f, d, c, t$ has bottleneck $b(C_1) = 4$ and cost $c(C_1) = -4$



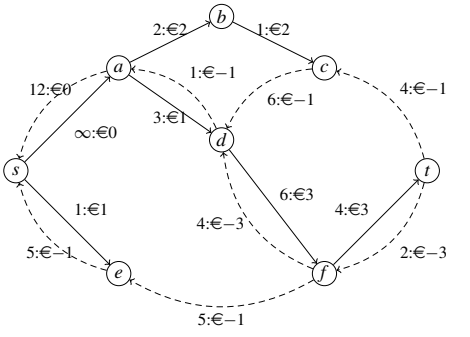
(c) The circulation $f_{C_1} = \mathsf{sw}(f, C_1)$; the arcs where it differs from f are drawn in bold



(d) The residual graph $\mathsf{R}(f_{C_1})$; cycle $C_2 = a, d, c, b, a$ has $b(C_2) = 1$, $c(C_2) = -2$



(e) The circulation $f_{C_2} = \mathsf{sw}(f_{C_1}, C_2)$; the arcs where it differs from f_{C_1} are drawn in bold



(f) The residual graph $\mathsf{R}(f_{C_2})$ has no negative-cost cycles

Figure 5.3 Circulations and their residual graphs. Arcs of circulation networks are labeled with f $[\ell..u]$ $€c$, and arcs of residual graphs with r:$€c$.

Onwards, we denote f_C also by $\mathsf{sw}(f, C)$, since it is obtained from f by "switching" the flow along C. Exercise 5.8 asks the reader to show that the function $\mathsf{sw}(f, C)$ is indeed a circulation over N. See Figure 5.3(c) for the circulation $\mathsf{sw}(f, C_1)$ obtained from f drawn in Figure 5.3(a), where C_1 is the cycle t, f, d, c, t of $\mathsf{R}(f)$.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

We define the *cost* of a cycle C in $\mathsf{R}(f)$, and denote it $c(C)$, as the sum of the costs of the arcs of C. The following lemma gives a necessary condition for a circulation to be one of minimum cost.

LEMMA 5.7 *Let $N = (G, \ell, u, c)$ be a circulation network and let f be a circulation over N. If there is a cycle C in $\mathsf{R}(f)$ of negative cost, then the circulation $\mathsf{sw}(f, C)$ satisfies $\mathsf{cost}(\mathsf{sw}(f, C)) < \mathsf{cost}(f)$.*

Proof Let C be a cycle in $\mathsf{R}(f)$ of cost $c(C) < 0$. Circulations f and $\mathsf{sw}(f, C)$ differ only on the arcs of the cycle C, where their values differ by precisely $b(C) > 0$. By the definition of the cost function extended to $\mathsf{R}(f)$, it holds that $\mathsf{cost}(\mathsf{sw}(f, C)) = \mathsf{cost}(f) + b(C)c(C)$. Since $c(C) < 0$, the claim follows. □

For example, the circulation f_{C_1} in Figure 5.3(c) is not of minimum cost, since the cycle $C_2 = a, d, c, b, a$ in $\mathsf{R}(f_{C_1})$ has cost -2; the circulation $\mathsf{sw}(f_{C_1}, C_2)$ is shown in Figure 5.3(e).

The following result, showing that the above necessary condition is also a sufficient condition, is a fundamental result on minimum-cost circulations, one that will also give rise in the next section to an algorithm for solving a minimum-cost circulation problem. In particular, it implies that the circulation in Figure 5.3(e) is of minimum cost.

THEOREM 5.8 *Let $N = (G, \ell, u, c)$ be a circulation network. A circulation f over N is of minimum cost among all circulations over N if and only if $\mathsf{R}(f)$ has no cycles of negative cost.*

Proof The forward implication follows from Lemma 5.7. For the backward implication, let f' be an arbitrary circulation over N, and consider the function f_Δ over N defined as

$$f_\Delta(x, y) = f'(x, y) - f(x, y),$$

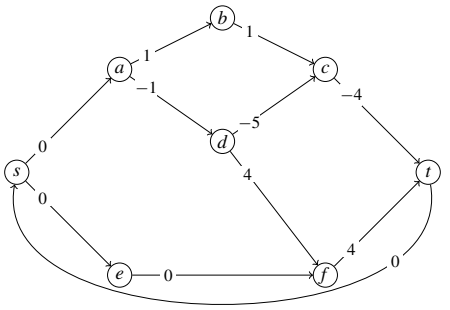
for all $(x, y) \in E(G)$. Clearly, f_Δ is a circulation over G, but it might not satisfy the arc constraints of N. Circulation f_Δ can instead be transformed into a circulation $\tilde{f}_\Delta$ over $\mathsf{R}(f)$ such that $\tilde{f}_\Delta(x, y) \geq 0$, for all $(x, y) \in E(\mathsf{R}(f))$, and $\mathsf{cost}(f_\Delta) = \mathsf{cost}(\tilde{f}_\Delta)$, as follows. For every $(x, y) \in E(G)$,

- if $f_\Delta(x, y) > 0$, then $f(x, y) < f'(x, y) \leq u(x, y)$ and thus the arc (x, y) is present in $\mathsf{R}(f)$; we set $\tilde{f}_\Delta(x, y) = f_\Delta(x, y)$; and
- if $f_\Delta(x, y) < 0$, then $\ell(x, y) \leq f'(x, y) < f(x, y)$ and thus the arc (y, x) is present in $\mathsf{R}(f)$; we set $\tilde{f}_\Delta(y, x) = -f_\Delta(x, y)$ (and $\tilde{f}_\Delta(x, y) = 0$ if the arc (x, y) is present in $\mathsf{R}(f)$).

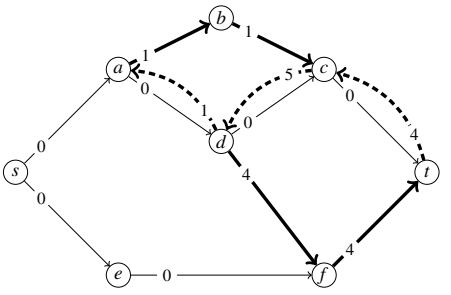
See Figure 5.4 for an example. As in Theorem 5.2 and Exercise 5.3, circulation $\tilde{f}_\Delta$ can be decomposed into $k \leq |E(\mathsf{R}(f))|$ cycles of $\mathsf{R}(f)$, $C_1, \ldots, C_k$, each C_i having a weight $w_i > 0$, such that, for every arc $(x, y) \in E(\mathsf{R}(f))$, it holds that

$$\tilde{f}_\Delta(x, y) = \sum_{C_i : (x,y) \in C_i} w_i.$$

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press



(a) The difference f_Δ between circulations f (Figure 5.3(a)) and f_{C_2} (Figure 5.3(e))



(b) The transformed circulation $\tilde{f}_\Delta$ over $\mathsf{R}(f_{C_2})$; we omitted costs and "decrease" arcs with null circulation. $\tilde{f}_\Delta$ can be decomposed into cycles a, b, c, d, a of weight 1 and cost 2, and t, c, d, f, t of weight 4 and cost 4

Figure 5.4 The difference between two circulations and its representation in the residual graph of the latter.

Therefore,

$$\mathsf{cost}(f') - \mathsf{cost}(f) = \mathsf{cost}(f_\Delta) = \mathsf{cost}(\tilde{f}_\Delta) = \sum_{i=1}^{k} w_i c(C_i).$$

Since no cycle in $\mathsf{R}(f)$ has negative cost, $\mathsf{cost}(f') - \mathsf{cost}(f) \geq 0$, implying that f is a minimum-cost circulation. □

5.2.2 A pseudo-polynomial algorithm

Theorem 5.8 shows, on the one hand, that the optimality of a given circulation f can be checked in polynomial time, by checking the existence of negative-cost cycles in $\mathsf{R}(f)$. This can be done, for example, in time $O(nm)$ with the Bellman–Ford method from Section 4.2.2. On the other hand, it leads to an algorithm for finding a minimum-cost circulation over a network N:

(1) start with any circulation f over N satisfying the demand and capacity constraints;
(2) as long as the residual graph $\mathsf{R}(f)$ of f has a cycle C of negative cost, replace f by $\mathsf{sw}(f, C)$.

COROLLARY 5.9 *If $N = (G, \ell, u, c)$ is a circulation network, with ℓ, u, and c integer-valued, then a minimum-cost circulation can be found in time $O(nm^2CU)$, where $n = |V(G)|$, $m = |E(G)|$, C is the maximum absolute value of the costs, and U is the maximum capacity.*

Proof The problem of finding an initial feasible circulation over N satisfying the demand and capacity constraints can be reduced to finding a minimum-cost circulation in a network N' without demand constraints, a problem in which the null circulation is feasible. Moreover, N' can be constructed such that it has $O(n)$ vertices and $O(m)$ arcs,

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

costs 0 or -1, and the maximum capacity is $O(U)$; thus a minimum-cost circulation on N can be found in time $O(nm^2U)$. Exercise 5.9 asks the reader to construct this reduction.

The cost of any feasible circulation is at most mCU, since, by definition, for each arc $(x, y) \in E(G)$, $-C \leq c(x, y) \leq C$ and $u(x, y) \leq U$. Analogously, the cost of the optimal circulation is at least $-mCU$. At each iteration of the above algorithm, the cost of f is changed by $b(C)c(C) < 0$. If ℓ, u, and c are integer-valued, then $b(C) \geq 1$ and $|c(C)| \geq 1$, and thus the algorithm terminates after $O(mCU)$ iterations.

If C is found using the Bellman–Ford algorithm, we can conclude that a minimum-cost circulation over N can be found in time $O(nm^2CU)$. □

Another corollary of Theorem 5.8 is the following.

COROLLARY 5.10 *If $N = (G, \ell, u, c)$ is a circulation network, with ℓ and u integer-valued, then the minimum-cost circulation problem always admits an integer-valued minimum-cost circulation.*

Proof In the proof of Corollary 5.9, we start either with a null circulation or with an optimal one in a circulation network with the same capacities, and integer costs and demands. At each iteration of the algorithm, each value of the circulation, if changed, is changed by $b(C)$. This is an integer, since ℓ and u are integer-valued. □

The complexity of the above algorithm depends on the actual values of the input variables and can take exponential time on certain inputs. However, for adequate choices of a cycle C of negative cost in $\mathsf{R}(f)$, the number of iterations can be bounded only by a polynomial in n and m, independently of the integrality of ℓ, u, or c. For example, choosing C to be a cycle of minimum-mean cost (see Exercise 4.18 on how this can be found in $O(nm)$ time), the number of iterations can be shown to be at most $4nm^2\lceil \log n \rceil$, leading to an algorithm of complexity $O(n^2m^3\lceil \log n \rceil)$. Using more advanced ideas, more efficient algorithms for solving minimum-cost circulations can be devised; see the various complexity bounds and references cited in the literature section of this chapter.

See Insight 5.2 for an extension of the minimum-cost flow problem to flow networks with convex costs.

5.3 Bipartite matching problems

In the following sections we show how minimum-cost flows can solve various optimization problems, which often seem unrelated to flows at first. To ensure a polynomial-time solution, the size of the reduction must be polynomial in the original input size.

Insight 5.2 Convex cost functions

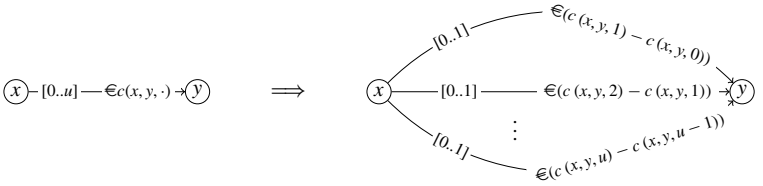
The minimum-cost flow problem discussed in this section has, for each arc, a cost function linear in the flow value. A more general problem is the one in which the cost function of each arc is convex in the value of the flow. We are given a flow network $N = (G, \ell, u, c, q)$, where $\ell, u : E(G) \to \mathbb{N}$, and $c : E(G) \times \mathbb{N} \to \mathbb{Q}$ is a convex

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

function in the last argument, that is, for each arc (x, y), the function $c(x, y, \cdot)$ has a "bathtub" shape. The task is to find a flow $f : E(G) \to \mathbb{N}$ over N that minimizes

$$\mathsf{cost}(f) = \sum_{(x,y)\in E(G)} c(x, y, f(x, y)).$$

Although efficient algorithms for this problem exist (see the references cited in the literature section), the following pseudo-polynomial reduction to a standard minimum-cost flow problem can turn out to be effective in practice. Construct the network N' from N, by replacing every arc $(x, y) \in E(G)$ by $u(x, y)$ parallel arcs between x and y, with demand 0 and capacity 1, such that the ith arc, $1 \le i \le u(x, y)$, has cost $c(x, y, i) - c(x, y, i - 1)$. Exercise 5.10 asks the reader to show that this reduction is correct, and that the cost of the optimal flow in N equals the cost of the optimal flow in N' plus $\sum_{(x,y)\in E(G)} c(x, y, 0)$. If an approximate solution is sufficient, then one can reduce the size of this reduction by introducing only $u(x, y)/t$ parallel arcs of capacity $t \ge 1$ such that the ith arc has cost $c(x, y, it) - c(x, y, (i - 1)t)$.



On the other hand, if the cost functions are concave, then this problem is generally NP-hard.

5.3.1 Perfect matching

In a bipartite matching problem one must match the objects of one set A with the elements on another set B, while satisfying some constraints and minimizing or maximizing a certain cost. Let us look at one of the simplest instances of such a problem, namely the minimum-cost perfect matching problem. Consider the problem in Example 5.2 below. We can model it as a *bipartite* graph, namely as an undirected graph $G = (V, E)$, where V is partitioned into two sets, A and B, such that there are no edges between vertices in A and there are no edges between vertices in B. When the two sets A and B are known in advance, we will write $G = (A \cup B, E)$.

Example 5.2 In a factory, n jobs need to be executed using one of the n available machines. We assume that each job takes one day to complete, and that each job can be performed by only a particular set of machines. Depending on the machine, each job can be performed with a given operating cost. Each machine cannot perform more than one job at the same time, and all jobs need to be finished on the same day. The task is to assign each job to a machine such that the total operating cost is minimized.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

Set A to be the set of jobs, and set B to be the set of machines. Between every job x and every machine y that can solve job x we add the edge (x, y) to E. An assignment of the n jobs to the n machines is just a set M of edges of G with the property that no two edges of M share a vertex, and all vertices are touched by some edge of M.

More formally, given a set M of edges of G, and given $x \in V$, we denote by $d_M(x)$ the number of edges of M incident to x, namely

$$d_M(x) = |\{y \in V \mid (x, y) \in E\}|. \tag{5.1}$$

The set M is called a *matching* if, for all $x \in A \cup B$, $d_M(x) \leq 1$ holds. Solving the problem in Example 5.2 amounts to solving Problem 5.3 below (see Figure 5.5(a) for an example).

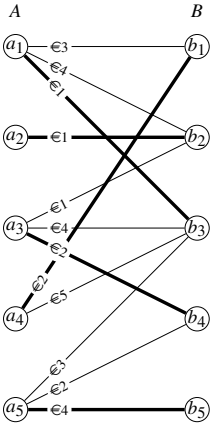
Problem 5.3 Minimum-cost perfect matching

Given a bipartite graph $G = (A \cup B, E)$, and a cost function $c : E \to \mathbb{Q}_+$, find a *perfect* matching M, in the sense that

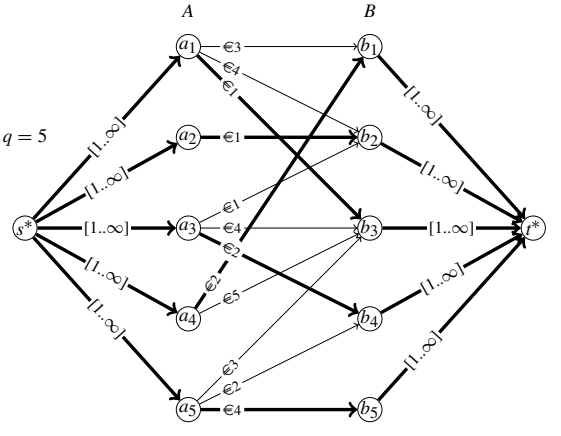
- for every $x \in A \cup B$, $d_M(x) = 1$,

which minimizes $\sum_{(x,y)\in M} c(x, y)$, or report that no perfect matching exists.

Let us now see how the handles provided by a minimum-cost flow problem can be used to solve the minimum-cost perfect matching problem. We construct, as illustrated in Figure 5.5(b), the flow network $N = (G^*, \ell, u, c, q)$, where G^* is obtained from the input bipartite graph G, by orienting all edges from A toward B; they are assigned



(a) A bipartite graph G whose edges are labeled with costs; the minimum-cost perfect matching is highlighted.



(b) The flow network N constructed from G; arcs incident to s^* or t^* have null cost; arcs between sets A and B have demand 0 and capacity 1; arcs with non-null value in the minimum-cost flow over N are highlighted.

Figure 5.5 Reducing minimum-cost perfect matching to minimum-cost flow.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

demand 0 and cost as in the input to the minimum-cost perfect matching problem. A global source s^* is added, together with arcs from s^* to every vertex of A. Likewise, a global sink t^* is added, together with arcs from every vertex of B to t^*.

We set the flow value to be $q = |A|$, and set the demand of all arcs incident to s^* and t^* to 1, and their cost to 0. Observe that we have not assigned capacities to the arcs of N, since, if $q = |A|$ and all arcs incident to s^* have demand 1, then their flow values must be at most 1. Moreover, if a perfect matching exists, then $|A| = |B|$, and hence also all arcs incident to t^* have flow value at most 1.

By Corollary 5.10, network N admits an integer-valued minimum-cost flow, which *induces* a perfect matching M of G consisting of those edges between the sets A and B with flow value 1. Vice versa, every matching in G analogously *induces* a flow in N. A minimum-cost circulation over a network without capacity constraints, and whose demands are bounded by U, can be solved in time $O(n \log U(m + n \log n))$, by the algorithm of Gabow and Tarjan (see the literature section of this chapter). Therefore, since the flow network N constructed from G as described above has only demands, and they are all at most 1, we can further refine this reduction to obtain an algorithm of improved complexity.

THEOREM 5.11 *The minimum-cost perfect matching problem on a bipartite graph G with n vertices, m edges, and non-negative costs is solvable in time $O(nm + n^2 \log n)$.*

Proof We transform the flow network N into a circulation network N' with demands only by adding the arc (t^*, s^*) with demand 0 and cost $c(t^*, s^*) = 1 + \sum_{(x,y)\in E(G)} c(x, y)$. Let f be an integer-valued minimum-cost circulation over N'.

Suppose that G admits a perfect matching M, thus also that $|A| = |B|$. Assume also for a contradiction that $f(t^*, s^*) > |A|$. The perfect matching M induces another circulation f', which satisfies $f'(t^*, s^*) = |A|$. By the choice of $c(t^*, s^*)$, we get $\mathsf{cost}(f') < \mathsf{cost}(f)$, which contradicts the minimality of f. Therefore, we have that indeed $f(t^*, s^*) = |A|$, and thus f' takes the value 1 on all arcs incident to s^* or to t^*. Thus, it induces a minimum-cost perfect matching in G.

If G does not admit a perfect matching, then some arc incident to s^* or t^* has a value of f' strictly greater than 1. □

5.3.2 Matching with capacity constraints

In the next problem we are interested in possibly matching more elements of A to the same element of B (a *many-to-one matching*), but with two particular constraints on their number. Consider the following example.

Example 5.4 In another factory, n jobs need to be executed using one of the j available machines. In this problem, we assume that a machine can execute multiple jobs, but that there is a limit on the total number of jobs it can execute. As before, a job can be executed only by a particular set of machines, with a given operating cost. There is one additional constraint, namely that jobs are of two types, easy and hard, and that each machine also has a limit on the number of hard jobs it can execute.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

The task is to assign each job to a machine such that the total operating cost is minimized, and that no machine executes more jobs than its total limit, and no more hard jobs than its limit for hard jobs.

The problem in Example 5.4 can be modeled as before, through a bipartite graph $G = (A \cup B, E)$ in which the set A of jobs is further partitioned into A_e, the set of easy jobs, and A_h, the set of hard jobs. Given a vertex $y \in B$ and a matching M, we denote

$$d_{M,A_h} = |\{x \in A_h \mid (x, y) \in M\}|. \tag{5.2}$$

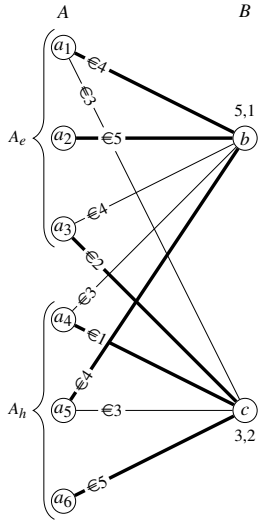
Solving the problem in Example 5.4 amounts to solving the following one (see Figure 5.6(a) for an example).

Problem 5.5 Minimum-cost many-to-one matching with constrained load factors

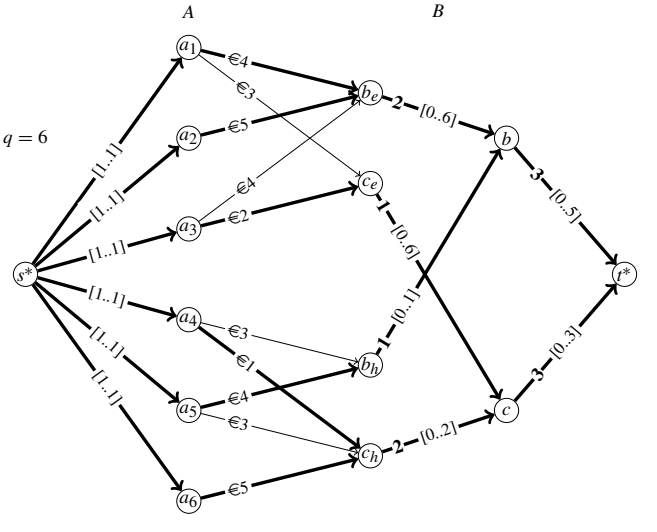
Given a bipartite graph $G = (A \cup B, E)$, with $A = A_e \cup A_h$, a cost function $c : E \to \mathbb{Q}_+$, and functions $\mathtt{ut}, \mathtt{uh} : B \to \mathbb{N}$, the capacities for the total number of jobs, and the total number of hard jobs, respectively, find a many-to-one matching M satisfying

- for every $x \in A$, $d_M(x) = 1$, and
- for every $y \in B$, $d_M(y) \leq \mathtt{ut}(y)$, and $d_{M,A_h}(y) \leq \mathtt{uh}(y)$,

and minimizing $\sum_{(x,y)\in M} c(x, y)$, or report that no such matching exists.



(a) A bipartite graph G and a minimum-cost many-to-one matching satisfying the load constraints of the vertices b and c, drawn as $\mathtt{ut}, \mathtt{uh}$



(b) The flow network N constructed from G; arcs incident to s^*, t^*, b, and c have null cost; arcs between sets A and B have demand 0 and capacity 1; arcs with non-null value in the minimum-cost flow over N are highlighted

Figure 5.6 Reducing minimum-cost many-to-one matching with constrained load factors to minimum-cost flow.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

If the constraints on the hard jobs were missing, we could easily model this problem as done in the previous section, but removing the demand constraints of the arcs from each machine y to the global sink, and setting their capacity to the total limit of jobs each machine y can execute.

Therefore, for this particular many-to-one matching problem we construct the flow network $N = (G^*, \ell, u, c, q)$ as illustrated in Figure 5.6(b). For each vertex y of B we introduce three copies of it, y, y_e, and y_h. We add an arc from y_h to y with null demand and cost, and capacity $\texttt{uh}(y)$, and an arc from y_e to y with null demand and cost, and infinite capacity. Moreover, we add an arc from y to the global sink t^* with null demand and cost, and capacity $\texttt{ut}(y)$. For every job $x \in A_e$ that can be executed on machine y, we add the arc (x, y_e), and for every job $x \in A_h$ that can be executed on machine y, we add the arc (x, y_h). These arcs have demand 0, capacity 1, and cost as in the many-to-one matching problem instance. We also add arcs from the global source s^* to each vertex in A. The arcs incident to s^* have null cost, and demand and capacity 1. Finally, the flow is required to have value $q = |A|$.

An integer-valued flow in N induces a many-to-one matching M in G. Matching M satisfies the upper bound constraints for the total number of hard jobs of any machine $y \in B$, because of the capacity constraint of the arc (y_h, y). It also satisfies the upper bound constraint on the total number of jobs of machine y because of the capacity constraint of the arc (y, t^*). Conversely, a many-to-one matching M in G satisfying all load constraints induces a flow in N. Thus, a minimum-cost many-to-one matching with constrained load factors can be found from an integer-valued minimum-cost flow in N.

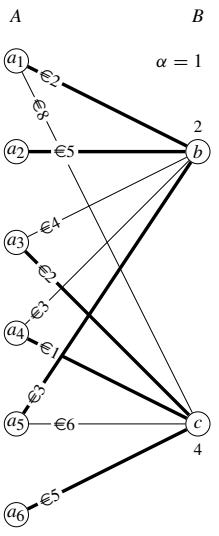
Graph G^* has $O(|V|)$ vertices and $O(|V| + |E|)$ arcs. The algorithms mentioned in the literature section of this chapter lead to various polynomial complexity bounds for this problem.

5.3.3 Matching with residual constraints

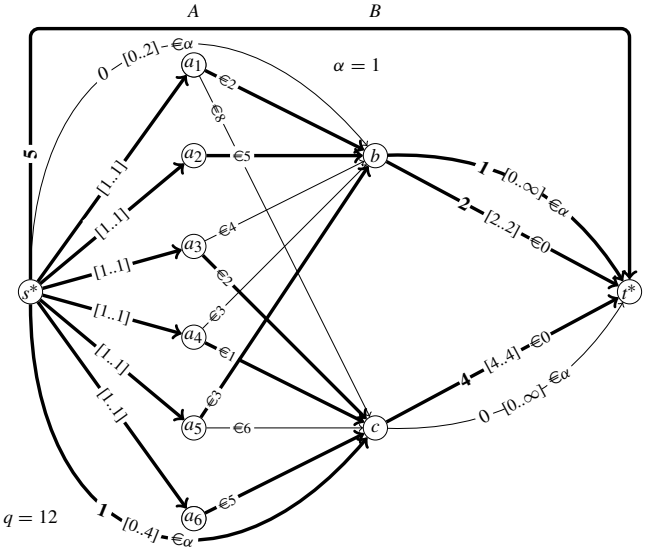
In the next many-to-one matching problem, we no longer have a rigid upper bound on the number of elements of A that can be matched with an element of B, but we are given ideal load factors for each element of B. Accordingly, we want to find a matching which better approximates these load factors. See Figure 5.7(a) for an example.

Example 5.6 In yet another factory, n jobs need to be executed using one of the j available machines. As before, a job can be executed only by a particular set of machines, with a given operating cost. For technical reasons, each machine must ideally execute a given number of jobs. If this number is not attained, the firm incurs an operating cost consisting of the absolute difference between the actual number of jobs the machine is assigned and its ideal load, multiplied by a fixed cost. The task is to assign each job to a machine such that the total operating cost is minimized.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press



(a) A bipartite graph G and a many-to-one matching with optimal residual load factors; vertices b and c are labeled with their ideal load factors



(b) The flow network N constructed from G; arcs from s^* to a vertex of A or to t^* have null cost; arcs between sets A and B have demand 0 and capacity 1; arcs with non-null value in the minimum-cost flow over N are highlighted

Figure 5.7 Reducing minimum-cost many-to-one matching with optimal residual load factors to minimum-cost flow.

Problem 5.7 Minimum-cost many-to-one matching with optimal residual load factors

Given a bipartite graph $G = (A \cup B, E)$, a cost function $c : E \to \mathbb{Q}_+$, constant $\alpha \in \mathbb{Q}_+$, and $\mathtt{id} : B \to \mathbb{N}$, the optimal load factors for the vertices in B, find a many-to-one matching M satisfying

- for every $x \in A$, $d_M(x) = 1$,

and minimizing

$$\sum_{(x,y)\in M} c(x,y) + \alpha \sum_{y\in B} |\mathtt{id}(y) - d_M(y)| \,.$$

We construct the flow network $N = (G^*, \ell, u, c, q)$ as in Figure 5.7(b). We start from G and orient all edges from A toward B, and set their demand to 0, their capacity to 1, and their cost as in the input many-to-one matching instance. We add a global source s^* with arcs toward all vertices in A, with demand and capacity 1, and null cost. Since we

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

pay the fixed penalty α for every job below or above the ideal load factor of a machine y, we add the following arcs:

- an arc from y to a global sink t^* with $\ell(y, t^*) = u(y, t^*) = \texttt{id}(y)$, and $c(y, t^*) = 0$;
- an arc from y to t^* with $\ell(y, t^*) = 0$, $u(y, t^*) = \infty$, and $c(y, t^*) = \alpha$; and
- an arc from s^* to y with $\ell(s^*, y) = 0$, $u(s^*, y) = \texttt{id}(y)$, and $c(s^*, y) = \alpha$.

Finally, we set $q = |A| + \sum_{y \in B} \texttt{id}(y)$, and also add the arc (s^*, t^*) with null demand and cost, and infinite capacity. The optimal matching for the many-to-one matching with optimal residual load factors consists of the edges of G whose corresponding arcs in G^* have non-null flow value in the integer-valued minimum-cost flow over N.

The correctness of this reduction can be seen as follows.

- As long as a machine y receives exactly $\texttt{id}(y)$ jobs, no operating cost is incurred; this is represented by the arc (y, t^*) with demand and capacity $\texttt{id}(y)$, and null cost.
- If y receives more jobs than $\texttt{id}(y)$, then these additional jobs will flow along the parallel arc (y, t^*) of cost α.
- If y receives fewer jobs than $\texttt{id}(y)$, then some compensatory flow comes from s^* through the arc (s^*, y), where these fictitious jobs again incur cost α. We set the capacity of (s^*, y) to $\texttt{id}(s^*, y)$, since at most $\texttt{id}(y)$ compensatory jobs are required in order to account for the lack of jobs for machine y.

Requiring flow value $q = |A| + \sum_{y \in B} \texttt{id}(y)$ ensures that there is enough compensatory flow to satisfy the demands of the arcs of type (y, t^*). Having added the arc (s^*, t^*) with null cost ensures that there is a way for the compensatory flow which is not needed to satisfy any demand constraint to go to t^*, without incurring an additional cost.

Graph G^* has $O(|V|)$ vertices and $O(|V| + |E|)$ arcs. The algorithms mentioned in the literature section of this chapter give various polynomial-complexity bounds for this problem.

5.4 Covering problems

Generally speaking, in a covering problem one is given an arbitrary graph G and is required to cover the vertices of G with some subgraphs of G, under some constraints. For example, in a perfect matching problem one has to cover all the vertices of the graph with single edges that do not intersect. In this section we discuss two other covering problems, where the covering subgraphs are more complex.

5.4.1 Disjoint cycle cover

In the following problem, the covering subgraphs are disjoint cycles.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

Example 5.8 A set of n players will take part in a competition. We know in advance which pairs of players are able to compete with one another, how much the spectators will enjoy a game between two players, and which of the two players is more likely to win. A set of games must be selected, such that

- each player plays exactly two games with other players;
- each player must be more likely to win in one of his games, and less likely to win in his other game; and
- the total enjoyment of the selected games be maximum.

We can model this problem as a directed graph $G = (V, E)$, where V is the set of players. A game in which player x is more likely to win than player y is represented by the arc (x, y). Observe that the set of games selected corresponds to a set of vertex-disjoint cycles of G covering all the vertices of G, since every vertex has precisely one in-neighbor and one out-neighbor in the set of games selected. Such a collection of cycles is called a *disjoint cycle cover*. Solving this problem amounts to solving the following one (see Figure 5.8(a) for an example).

Problem 5.9 Minimum-cost disjoint cycle cover

Given a directed graph $G = (V, E)$, and cost function $c : E \to \mathbb{Q}_+$, find a disjoint cycle cover $C_1, \ldots, C_k$ of G which minimizes

$$\sum_{i=1}^{k} \sum_{(x,y)\in C_i} c(x, y),$$

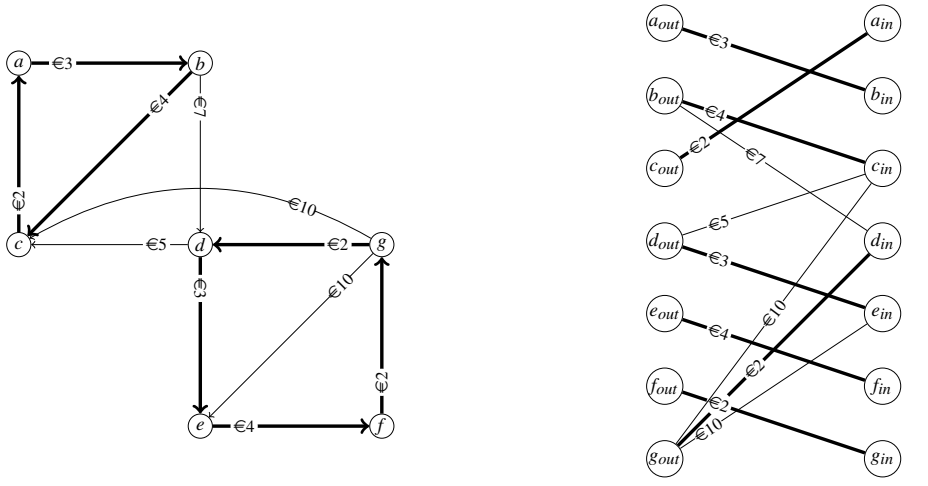
or report that no such cover exists.

Even though the problem in Example 5.8 is a maximization problem, and Problem 5.9 stated above is a minimization one, the positive costs accepted by Problem 5.9 can be suitably initialized as the sum of all possible spectators' enjoyment minus the benefit that the spectators receive if that game is played.

The minimum-cost disjoint cycle cover can be solved by a minimum-cost perfect matching problem in a bipartite graph, and thus by a minimum-cost flow problem. The idea of this reduction is that every vertex must choose its successor in some cycle of the optimal disjoint cycle cover. As such, we can construct a bipartite graph G^* to which we introduce, for every vertex $x \in V(G)$, two copies of it, x_{out} and x_{in}. For every arc (x, y) of G, we add an edge between x_{out} and y_{in} having cost $c(x, y)$. See Figure 5.8(b) for an example.

A disjoint cycle cover $C_1, \ldots, C_k$ in G naturally induces a matching M in G^* by adding to M the edge (x_{out}, y_{in}), for all arcs (x, y) in the cover. This is also a perfect matching since, in a disjoint cycle cover, every vertex is the start point of exactly one

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press



(a) A minimum-cost disjoint cycle cover in a directed graph G.

(b) A minimum-cost perfect matching in G^*.

Figure 5.8 Reducing minimum-cost disjoint cycle cover to minimum-cost perfect matching.

arc of the cover and the endpoint of exactly one arc of the cover. Conversely, every perfect matching M in G^* induces a disjoint cycle cover in the same way.

Observe that graph G^* has $O(|V|)$ vertices and $O(|E|)$ edges. From Theorem 5.11, we obtain the following corollary.

COROLLARY 5.12 *The minimum-cost disjoint cycle cover problem on a directed graph G with n vertices and m arcs and non-negative costs is solvable in time $O(nm+n^2 \log n)$.*

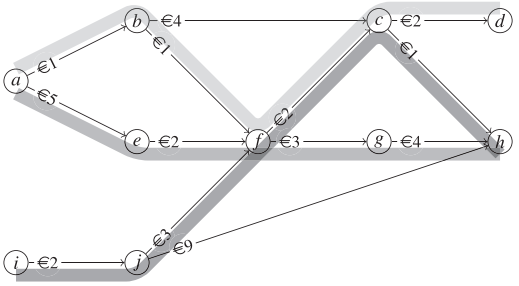
5.4.2 Minimum path cover in a DAG

In the next problem, the covering subgraphs are paths that may share vertices.

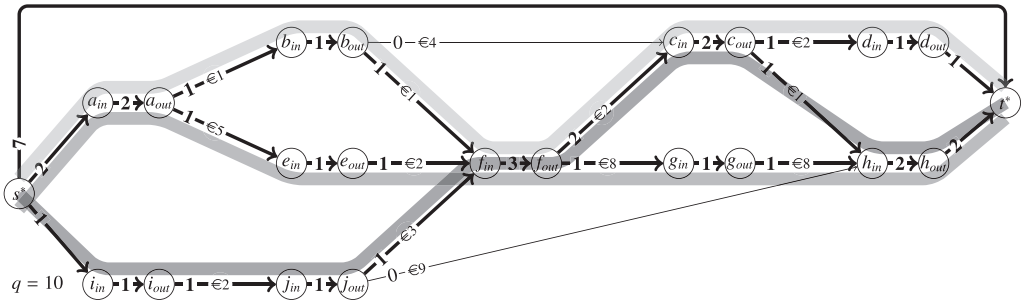
Example 5.10 A newspaper's office wants to cover a set of n events $e_1, \ldots, e_n$ taking place in a territory. Each event e_i starts at a known time $t(i)$ and lasts for a known duration $d(i)$. The newspaper is also aware of the time $t(i,j)$ and the cost $c(i,j)$ needed to travel from each event i to each other event j. The management would like to cover these events using the minimum number of reporters, and, among all such solutions, would like one that minimizes the total cost of traveling. Moreover, it is assumed that the reporters start and end their journey at the newspaper's premises.

In this example, the input graph $G = (V, E)$ is a DAG whose vertices are the events $e_1, \ldots, e_n$, and whose arcs are the possible connections between them: $(e_i, e_j) \in E$ if $t(i) + d(i) + t(i,j) \leq t(j)$. Each arc (e_i, e_j) has cost $c(i,j)$, the cost of traveling from event e_i to event e_j. We add two dummy events, e_0 and e_{n+1}, to model the requirement that the reporters start and end their journey at the newspaper's offices. For every $i \in \{1, \ldots, n\}$,

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press



(a) A DAG G and three paths forming a minimum-cost minimum path cover. Path 1: a, b, f, c, d; path 2: a, e, f, g, h; path 3: i, j, f, c, h.



(b) The flow network N constructed from G; arcs of type (x_{in}, x_{out}) have demand 1 and cost 0, arcs of type (x_{out}, y_{in}) have demand 0 and cost as in G, arcs incident to s^* or t^* have demand 0; arcs incident to s^* have demand 0 and their cost equals the sum of all costs plus 1, arcs incident to t^* have demand 0 and cost 0; arcs with non-null value in the minimum-cost flow over N are highlighted, and a decomposition of the flow into three paths is shown

Figure 5.9 Reducing minimum-cost minimum path cover to minimum-cost flow.

we add the arcs (e_0, e_i) and (e_i, e_{n+1}) labeled with the cost of traveling between the newspaper's offices and event e_i. A path in G from the unique source e_0 to the unique sink e_{n+1} is one possible route of a reporter. Each vertex has to be covered by at least one path, since each event must be covered by a reporter. Such a collection of paths covering all the vertices of a directed graph is called a *path cover*. Therefore, solving the problem in Example 5.10 amounts to solving the following problem (see Figure 5.9(a) for an example).

Problem 5.11 Minimum-cost minimum path cover

Given a DAG $G = (V, E)$, and cost function $c : E \to \mathbb{Q}_+$, find the minimum number k of paths $P_1, \ldots, P_k$ such that

- every $v \in V$ belongs to some P_i (that is, $P_1, \ldots, P_k$ form a path cover of G),
- every P_i starts in a source of G and ends in a sink of G, and

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

among all path covers with k paths satisfying the above conditions, the function

$$\sum_{i=1}^{k} \sum_{(x,y)\in P_i} c(x,y)$$

is minimized.

We construct the flow network $N = (G^*, \ell, u, c, q)$ as shown in Figure 5.9(b). Namely,

- for every vertex x of G, we introduce two vertices x_{in} and x_{out} in G^*, and add the arc (x_{in}, x_{out}) to G^*, with null cost and demand 1;
- for every arc (y, x) incoming to x of G, we add the arc (y_{out}, x_{in}) to G^*, with cost $c(y, x)$ and demand 0; and
- for every arc (x, y) outgoing from x of G, we add the arc (x_{out}, y_{in}) to G^*, with cost $c(x, y)$ and demand 0.

We add a global source s^* with arcs to every vertex x_{in}, if x is a source in G, with demand 0. Likewise, we add a global sink t^* and arcs (x_{out}, t^*), for every sink x in G, with demand 0.

As opposed to the previous problems, here we also need to find a solution of minimum size. Accordingly, since the cost of any path from a source to a sink is at most the sum of all costs of G, we set the cost of each arc (s^*, x_{in}) as

$$c(s^*, x_{in}) = 1 + \sum_{(x,y)\in E(G)} c(x,y). \tag{5.3}$$

We can then set the cost of the arcs incident to t^* to 0. We set the value of the flow to $q = |V|$, since at most $|V|$ paths are needed in any path cover. To account for the supplementary flow, we also add the arc (s^*, t^*) with cost 0 and demand 0.

Having an integer-valued minimum-cost flow f in the network N, we show how f can be converted into a minimum path cover in G whose paths start in a source of G and end in a sink of G. Then we show that this path cover also has minimum cost among all such minimum path covers.

Even though decomposing an arbitrary flow into the minimum number of paths is an NP-hard problem (by Theorem 5.3), the following lemma shows that in our case this problem is simpler.

LEMMA 5.13 *If f is a minimum-cost flow over the acyclic flow network N constructed as above, then any decomposition of f, minus the arc (s^*, t^*), into integer-weighted paths has the same number of paths.*

Proof Let $P_1, \ldots, P_k$ be a decomposition of f, minus the arc (s^*, t^*), into weighted paths, having integer weights $w_1, \ldots, w_k$, respectively. To prove our claim it suffices to show that $w_1 = \cdots = w_k = 1$.

Suppose for a contradiction that some path P_j has integer weight $w_j > 1$. Then, we can subtract 1 from the flow value of each of its arcs, and obtain another flow f', still

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

feasible over N, of strictly smaller cost than f, which contradicts the minimality of f. This is true because at least the cost of the first arc of P_j outgoing from s^* is positive. □

Hence f can induce a path cover in G in the following way. Since G^* is acyclic, we can consider a decomposition of f minus the arc (s^*, t^*) into the k paths $P_1, \ldots, P_k$ from s^* to t^* (all having weight 1, by the proof of Lemma 5.13). Removing the vertices s^* and t^* from each P_i, and contracting the arcs of the form (x_{in}, x_{out}) back into x, we obtain a collection $Q_1, \ldots, Q_k$ of paths in G.

We next show that k is also the minimum size of a path cover whose paths start in a source and end in a sink of G.

LEMMA 5.14 *The minimum number of paths starting in a source of G and ending in a sink of G and covering all its vertices is k, namely the number of paths that f is decomposable into.*

Proof We argue by contradiction, and assume that there exists such a collection of paths $Q'_1, \ldots, Q'_j$ with $j < k$ paths. These paths in G induce a feasible flow f' over the network N. Since flow f has value k, and flow f' has value j, it holds that

$$\sum_{(s^*,x_{in})\in E(G^*)} c(s^*, x_{in}) f(s^*, x_{in}) - \sum_{(s^*,x_{in})\in E(G^*)} c(s^*, x_{in}) f'(s^*, x_{in}) \geq (k-j)\left(1 + \sum_{(x,y)\in E(G)} c(x,y)\right),$$

according to (5.3). As noted above, the cost of any path from s^* to t^* in N is at most $\sum_{(x,y)\in E(G)} c(x,y)$. Therefore, f' has strictly smaller cost than f, contradicting the minimality of f. □

Therefore, $Q_1, \ldots, Q_k$ is a path cover of G with the minimum number of paths that start in a source of G and end in a sink of G. The fact that this path cover also has the minimum cost among all such path covers of size k follows from the optimality of f.

Graph G^* has $O(|V|)$ vertices and $O(|V| + |E|)$ arcs. Decomposing f into the k paths and constructing $Q_1, \ldots, Q_k$ can be done in time $O(|V| + |E|)$, by Theorem 5.2. Therefore, we obtain the following theorem.

THEOREM 5.15 *The minimum-cost minimum path cover problem on a DAG G with n vertices, m arcs, and non-negative costs is solvable in time $O(nm + n^2 \log n)$.*

Proof A minimum-cost circulation over a network without capacity constraints and demands bounded by U can be solved in time $O(n \log U(m + n \log n))$ by the algorithm of Gabow and Tarjan (see the literature section of this chapter). Since the flow network N constructed from G as described above has only demands, and they are all at most 1, we can transform N into an equivalent circulation network N' with demands only, by adding the arc (t^*, s^*) with demand 0. □

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

5.5 Literature

The seminal article on network flows is that by Ford & Fulkerson (1956). The first polynomial-time algorithm for the minimum-cost flow problem was given by Edmonds & Karp (1972), and the first strongly polynomial-time algorithm by Tardos (1985). Our presentation in this chapter is inspired by two well-known books on combinatorial optimization, *Network Flows* (Ahuja *et al.* 1993) and *Combinatorial Optimization* (Schrijver 2003).

The first proof of the fact that decomposing a flow into the minimum number of components is an NP-hard problem appeared in Vatinlen *et al.* (2008). The heuristics mentioned in Insight 5.1 are from Vatinlen *et al.* (2008) and Hartman *et al.* (2012).

Table 5.1, compiled from Schrijver (2003), lists some algorithms for minimum-cost circulation problems with integer-valued demands, capacities, and costs; the optimal solution returned is integer-valued.

One of the first algorithms for solving minimum-cost flows with convex cost functions was published by Hu (1966). See Weintraub (1974) and Minoux (1986) for polynomial-time algorithms for this problem, and Guisewite & Pardalos (1990) for minimum-cost flows with concave cost functions.

Algorithms for the maximum-cost bipartite matching problem include those published by Edmonds & Karp (1972), of complexity $O(n(m + n \log n))$, and by Gabow & Tarjan (1989), of complexity $O(\sqrt{n}m \log(nC))$, where C is the maximum absolute cost value. The minimum-cost perfect matching problem in an arbitrary graph can be solved in time $O(n(m + n \log n))$, by methods not based on network flows (Gabow 1990) (see Exercise 5.15 for using this algorithm in finding a minimum-cost maximum matching in an arbitrary graph).

The many-to-one matching with constrained load factors problem presented in Section 5.3.2 is adapted from Ahuja *et al.* (1993, Application 9.3). The many-to-one matching with optimal residual load factors problem presented in Section 5.3.3 is a simplified version of a problem proposed by Lo *et al.* (2013). The reduction of the disjoint cycle cover problem to a perfect matching problem used in Section 5.4.1 was first given by Tutte (1954).

Table 5.1 Some algorithms for minimum-cost circulations over integer-valued circulation networks. We denote by U the maximum value of a capacity (or demand), by U^+ the sum of all capacities, and by C the maximum absolute value of an arc cost.

$O(m \log U(m + n \log n))$	Capacity scaling (Edmonds & Karp 1972)
$O(n^{5/3}m^{2/3} \log(nC))$	Generalized cost scaling (Goldberg & Tarjan 1987)
$O(nm \log n \log(nC))$	Minimum-mean cost cycle canceling (Goldberg & Tarjan 1988, 1989)
$O(m \log n(m + n \log n))$	(Orlin 1988, 1993)
$O(nm \log(n^2/m) \log(nC))$	Generalized cost scaling (Goldberg & Tarjan 1990)
$O(nm \log \log U \log(nC))$	Double scaling (Ahuja *et al.* 1992)
$O((nm + U^+ \log U^+) \log(nC))$	(Gabow & Tarjan 1989)
$O(n \log U(m + n \log n))$	(Gabow & Tarjan 1989) – *for networks without capacities*

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

Our formulation of the minimum path cover problem from Section 5.4.2 included the constraint that the solution paths must start/end in a source/sink, respectively. This is usually absent from standard formulations of the problem; however, our minimum-cost flow reduction can accommodate any sets of starting and ending vertices of the solution paths.

The first result on path covers in DAGs is Dilworth's theorem (Dilworth 1950), which equates the minimum number of paths in a path cover to the maximum cardinality of an anti-chain. A constructive proof of this theorem (Fulkerson 1956) showed that the minimum path cover problem can be reduced to a maximum matching problem in a bipartite graph having $t(G)$ edges, where $m \leq t(G) \leq \binom{n}{2}$ is the number of arcs of the transitive closure of the input DAG G. Using the maximum matching algorithm of Hopcroft & Karp (1973), a minimum path cover can be computed in time $O(\sqrt{n}t(G))$. A minimum-cost maximum matching in the same graph, computable for example in time $O(n^2 \log n + nt(G))$ with the algorithm in Edmonds & Karp (1972), gives a minimum-cost minimum path cover. The connection between the minimum path cover problem and a network flow problem (namely, a minimum flow problem, see Exercise 5.12) appears for example in Ntafos & Hakimi (1979), Pijls & Potharst (2013), and Bang-Jensen & Gutin (2008). Note that the minimum path cover problem in arbitrary directed graphs is NP-complete, since a path cover with only one path is a Hamiltonian path. Example 5.10 is adapted from Bang-Jensen & Gutin (2008).

Exercises

5.1 Given an arbitrary graph G with a unique source s and a unique sink t, and given a function $f : E(G) \to \mathbb{Q}$ satisfying the flow conservation property on G, show that

$$\sum_{y \in N^+(s)} f(s, y) = \sum_{y \in N^-(t)} f(y, t).$$

5.2 Let G be a DAG with a unique source s and a unique sink t, and let $f : E(G) \to \mathbb{N}$ satisfy the flow conservation property on G, such that the flow exiting s equals q. Show that if there exists an integer x such that, for every $f(e)$, $e \in E(G)$ is a multiple of x, then f can be decomposed into q/x paths, each of weight x.

5.3 Show that a flow f on an arbitrary graph G with a unique source s and a unique sink t can be decomposed into at most $|E(G)|$ weighted paths or cycles. How fast can this be done?

5.4 In the 3-partition problem, we are given a set $A = \{a_1, \ldots, a_{3q}\}$ of $3q$ positive integers, such that

- $\sum_{i=1}^{3q} a_i = qB$, where B is an integer, and
- for all $i \in \{1, \ldots, 3q\}$ it holds that $B/4 < a_i < B/2$.

We are asked whether there exists a partition of A into q disjoint sets, such that the sum of the integers in each of these sets is B. This problem is known to be NP-hard,

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

even when the values of the $3q$ integers in the set A are bounded by a certain value not depending on q (it is called *strongly* NP-hard). Use a similar reduction to that in the proof of Theorem 5.3 to show that the 3-partition problem can be reduced to the problem of decomposing a flow in a DAG into the minimum number of weighted paths.

5.5 Given a weighted DAG G, show that we can find an s-t path whose bottleneck is maximum among all s-t paths of G, in time $O(|E(G)|)$.

5.6 Suppose that in a flow network N with a unique source and a unique sink some arcs have infinite capacity. Show that there exists a flow f over N of minimum cost, with the additional property that the value of f on each arc without capacity is at most the sum of all arc capacities.

5.7 Show that the minimum-cost circulation problem can be reduced to the minimum-cost flow problem.

5.8 Given a circulation f over a circulation network N, show that, if C is a cycle in the residual graph $\mathsf{R}(f)$ of f, then the circulation f_C defined in Section 5.2 is a circulation over N.

5.9 Given a circulation network $N = (G, \ell, u, c)$, consider the problem of finding a feasible circulation over N, that is, a circulation satisfying the demand and capacity constraints of N. Construct the circulation network N' as follows:

- N' has the same vertices as N;
- for every arc (x, y) of N, add to N' two parallel arcs with demand 0: one with cost -1 and capacity $\ell(x, y)$, and one with cost 0 and capacity $u(x, y) - \ell(x, y)$.

Show that N admits a feasible circulation if and only if the minimum-cost circulation over N' has cost less than or equal to minus the sum of all demands of N. How can you obtain a feasible circulation f over N from a minimum-cost circulation f' over N'?

5.10 Show that the reduction of a minimum-cost flow problem with convex costs to a standard minimum-cost flow problem from Insight 5.2 is correct.

5.11 Given a DAG G with a unique source s and a unique sink t, show that we can find the maximum number of s-t paths without common vertices (apart from s and t) by a reduction to a maximum flow problem.

5.12 In the *minimum flow problem*, we are given a flow network N with arc demands only, and are asked to find a flow of minimum value over N satisfying all demand constraints. Show that the minimum flow problem can be solved by two applications of a maximum flow problem.

5.13 Show that finding a *maximum-cardinality matching* in a bipartite graph can be reduced to a minimum-cost flow problem.

5.14 Show that, among all maximum-cardinality matchings of a bipartite graph G, the problem of finding one of minimum cost can be reduced to a minimum-cost network flow problem.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press

5.15 Suppose that you have an algorithm for solving the minimum-cost perfect matching problem in an arbitrary graph (not necessarily bipartite). Show that, given any graph G with costs associated with its edges, you can use this algorithm, by appropriately transforming G, to find a *minimum-cost maximum matching* (that is, a matching of maximum cardinality in G, with the additional property that, among all matchings of maximum cardinality, it has minimum cost).

5.16 Consider Problem 5.7, in which one is required to minimize instead

$$\sum_{(x,y)\in M} c(x,y) + \alpha \sum_{y\in B} |\mathtt{id}(y) - d_M(y)|^2 .$$

How do you need to modify the reduction given for Problem 5.7 to solve this new problem? Does the resulting flow network still have size polynomial in the size of the input? *Hint.* Use the idea from Insight 5.2.

5.17 Show that the problem of finding a minimum-cost disjoint cycle cover in an undirected graph can be reduced to the problem of finding a minimum-cost disjoint cycle cover in a directed graph.

5.18 An *edge cover* of an undirected graph G is a set of edges covering all the vertices of G.

- Show that a minimum-cost edge cover in a bipartite graph can be solved by a minimum-cost flow problem.
- Show that a minimum-cost edge cover in an arbitrary graph G can be reduced to the problem of finding a maximum-weight matching in G.

https://doi.org/10.1017/CBO9781139940023.007 Published online by Cambridge University Press