Discrete Optimization: Network Flows and Matchings

Naoyuki Kamiyama

Abstract In this paper, we give a brief introduction to network flow problems and matching problems that are representative problems in discrete optimization. Network flow problems are used for modeling, e.g., car traffic and evacuation. Matching problems are used when we allocate jobs to workers and assign students to laboratories, and so on. Especially, we focus on mathematical models that are used in these problems.

Keywords Discrete optimization · Matching · Network flow

1 Introduction

Optimization is a branch of mathematics that studies problems of finding the optimal object from a set of objects. Especially, discrete optimization is the study of optimization problems with certain discrete structures. In this paper, we give a brief introduction to network flow problems and matching problems that are representative problems in discrete optimization. Network flow problems are used for modeling, e.g., car traffic and evacuation. Matching problems are used when we allocate jobs to workers and assign students to laboratories, and so on. Especially, we focus on mathematical models that are used in these problems. See references given in each section for theory and algorithms.

In the rest of this paper is organized as follows. In Sect. 2, we explain graphs that play an important role in discrete optimization. In Sect. 3, we consider network flow problems. In Sect. 4, we consider matching problems.

N. Kamiyama (✉)
Institute of Mathematics for Industry, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka 819-0395, Japan
e-mail: kamiyama@imi.kyushu-u.ac.jp

Throughout this paper, we denote by $\mathbb{R}$, $\mathbb{R}_+$ and $\mathbb{Z}_+$ the sets of real numbers, nonnegative real numbers and nonnegative integers, respectively.

2 Graphs

In this section, we explain basic concepts related to graphs. Intuitively speaking, graphs model "links" connecting "objects" (e.g., people, countries, papers and words). In graph theory, "vertices" and "arcs" (or "edges") correspond to objects and links, respectively. See, e.g., [2, 17] for coverage of concepts related to graph theory. Here we define two kinds of graphs. The first one is a directed graph and the second one is an undirected graph.

A *directed graph* $D = (V, A)$ is a pair of a vertex set V and an arc set A, where every arc in A is an ordered pair of vertices in V. See Fig. 1a for an example of a directed graph. Notice that in a directed graph, we take the direction of each arc into consideration. For each arc $a = (v, w)$ in A, we call v and w the *tail* and *head* of a, respectively. For each vertex v in V, we denote by $\Delta^+(v)$ and $\Delta^-(v)$ the sets of arcs of A whose tails and heads are v, respectively. That is, $\Delta^+(v)$ represents the set of arcs "leaving" v, and $\Delta^-(v)$ represents the set of arcs "entering" v. For example, in Fig. 1a, $\Delta^+(v) = \{a_3\}$ and $\Delta^-(v) = \{a_1, a_2\}$.

An *undirected graph* $G = (V, E)$ is a pair of a vertex set V and an edge set E, where every edge e in E is a subset of V with $|e| = 2$. See Fig. 1b for an example of an undirected graph. Notice that in an undirected graph, we do not take the direction of each edge into consideration. For each subset F of E and each vertex v in V, we denote by $F(v)$ the set of edges e in F with $v \in e$. For example, in Fig. 1b, $E(v) = \{e_1, e_2, e_3\}$. An undirected graph $G = (V, E)$ is called a *bipartite graph*, if V is partitioned into two subsets P and Q, and every edge in E connects a vertex in P and a vertex in Q. See Fig. 6 for an example of a bipartite graph.

3 Network Flows

In this section, we consider network flow problems that are used for modeling, e.g., car traffic and evacuation. See [1] for applications of network flow problems. In the first half of this section, we consider an ordinary network flow model, called a static network flow. In the second half, we consider a dynamic network flow in which we take an important factor "time" into consideration. See, e.g., [1, 5, 17, 18] for coverage of concepts related to networks flows.

3.1 Static Network Flows

In this section, we explain an ordinary network flow model, called a static network flow model. Intuitively speaking, we do not take into account "time," i.e., objects "ceaselessly" flow.

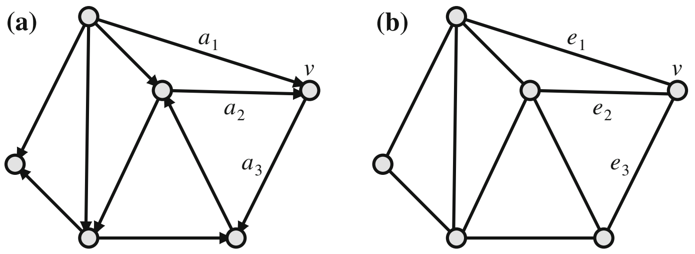

Fig. 1 **a** A directed graph. **b** An undirected graph

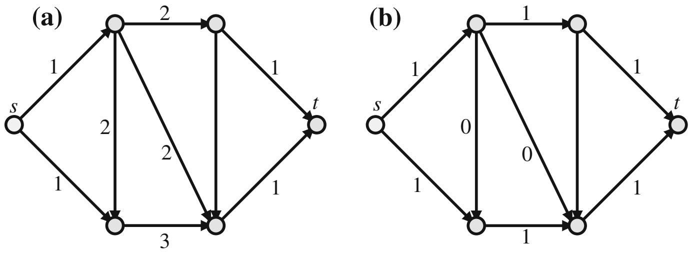

Fig. 2 **a** A static network. **b** A static flow

More formally, in a static network flow model, we are given a directed graph $D = (V, A)$ with specified vertices $s, t \in V$ and a capacity function $c\colon A \to \mathbb{R}_+$. We call the pair of D and c a *static network*. A function $\xi : A \to \mathbb{R}_+$ is called a *static flow*, if it satisfies the following two condition.

Capacity constraint. For every arc a in A,

$$\xi(a) \leq c(a).$$

Flow conservation. For every vertex v in V with $v \neq s, t$,

$$\sum_{a \in \Delta^+(v)} \xi(a) = \sum_{a \in \Delta^-(v)} \xi(a).$$

The *value* of a static flow ξ is defined as

$$\sum_{a \in \Delta^-(t)} \xi(a) - \sum_{a \in \Delta^+(t)} \xi(a).$$

See Fig. 2 for an example of a static flow model and a static flow. In Fig. 2a, the numbers attached to arcs represent their capacities. In Fig. 2a, the numbers attached to arcs represent a static flow.

Here we explain two representative problems in a static flow model. The first one is the *maximum-flow problem*. The goal of this problem is to find a static flow with maximum value. That is, this problem models the situation in which we want to send objects as much as much possible from s to t. For example, in the static network illustrated in Fig. 2a, the static flow in Fig. 2b is a solution of the maximum-flow problem. It is known that this problem can be efficiently solved. See, e.g., [9] for an efficient algorithm for the maximum-flow problem.

The second problem is the *minimum-cost flow problem*. In this problem, we are given a demand $d \in \mathbb{R}_+$ and a cost function $k\colon A \to \mathbb{R}_+$. The *cost* of a static flow $\xi\colon A \to \mathbb{R}_+$ is defined as

$$\sum_{a \in A} k(a) \cdot \xi(a).$$

The goal of the minimum-cost flow problem is to find a static flow whose cost is minimum among all static flows whose value is equal to d. That is, this problem models the situation in which a penalty incurs when we send objects on arcs. This problem can be efficiently solved. See, e.g., [15] for an efficient algorithm for the minimum-cost flow problem.

3.2 Dynamic Network Flows

In this section, we consider a dynamic network flow in which we take an important factor "time" into consideration. That is, in this model, the time required to transit an arc plays an important role.

More formally, in a dynamic network flow model, we are given a directed graph $D = (V, A)$ with a terminal subsets S of V partitioned into S^+ and S^-, a capacity function $c\colon A \to \mathbb{R}_+$, a transit time function $\tau\colon A \to \mathbb{Z}_+$ and a time horizon $T \in \mathbb{Z}_+$. The value $\tau(a)$ represent the time required to transit from the tail of a to the head of a. We call the triple D, c and τ a *dynamic network*. A function $f\colon A \times \mathbb{Z}_+ \to \mathbb{R}_+$ is called a *dynamic flow*, if it satisfies the following two conditions.

Capacity constraint. For each arc a in A and each nonnegative integer θ,

$$f(a, \theta) \le c(a).$$

Flow conservation. For each vertex v in $V \setminus S$ and each nonnegative integer θ,

$$\mathsf{ex}_f(v, \theta) \begin{cases} \ge 0 & \text{if } \theta = 0, 1, \dots, T-1 \\ = 0 & \text{if } \theta \ge T, \end{cases}$$

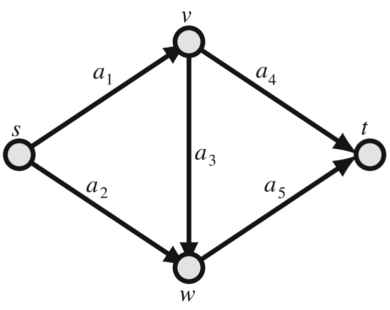

Fig. 3 A dynamic network. In this figure, $S^+ := \{s\}$ and $S^- := \{t\}$. Furthermore, we set $c(a) := 1$ for every arc a in A, and $\tau(a_1) = \tau(a_5) = 0$, $\tau(a_3) = 1$, $\tau(a_2) = \tau(a_4) = 3$ and $T = 5$

where define

$$\mathsf{ex}_f(v,\theta) := \sum_{a\in\Delta^-(v)} \sum_{t=0}^{\theta-\tau(a)} f(a,t) - \sum_{a\in\Delta^+(v)} \sum_{t=0}^{\theta} f(a,t).$$

Intuitively speaking, $f(a,\theta)$ represents the value of flow entering the tail of a at the time θ. The value $\mathsf{ex}_f(v,\theta)$ represents the excess of supplies on the vertex v until the time θ. Notice that in the flow conservation constraint, we allow supplies to stay at vertices. See Fig. 3 for an example of a dynamic network.

Here we explain two representative problems in a dynamic flow model. The first one is the *maximum dynamic flow problem*. Intuitively speaking, this problem is a dynamic version of the maximum-flow problem in a static flow model. In this problem, S^+ and S^- consists of single vertices s^+ and s^-, respectively. The goal of the maximum dynamic flow problem is to find a dynamic flow maximizing $\mathsf{ex}_f(s^-, T)$, i.e., we want to send objects from s^+ to s^- as much as possible within the time limit T. This problem can be efficiently solved. See, e.g., [6] for an efficient algorithm for the maximum dynamic flow problem.

The second problem is the *dynamic transshipment problem*. There exists no corresponding problem in a static flow model. In this problem, we are given a demand function $d\colon S \to \mathbb{R}$ such that

$$d(s) \begin{cases} \le 0 & s \in S^+ \\ \ge 0 & s \in S^-. \end{cases}$$

The dynamic transshipment problem asks for discerning where there exists a dynamic flow f such that

$$\forall s \in S\colon\ \mathsf{ex}_f(s,T) = d(s),$$

and find it, if one exists. That is, we want to send objects from S^+ to S^- so that all supplies and demands are satisfied. This problem can be efficiently solved. See, e.g., [11] for an efficient algorithm for the dynamic transshipment problem.

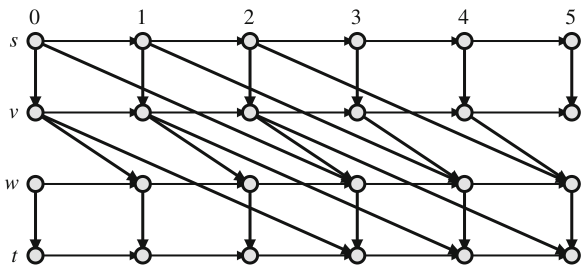

Fig. 4 The time-expanded network of the dynamic network in Fig. 3

From now on, we show that problems in a dynamic flow model can be reduced to those in a static flow model. For this, we define the *time-expanded graph* $\mathscr{T}$ of a dynamic network D with c and τ, which is a static network. See Fig. 4 for an example of a time-expanded network. The vertex set of $\mathscr{T}$ consists of a new vertex v_θ for each vertex v in V and each $\theta = 0, 1, 2, \dots, T$. The arc set of $\mathscr{T}$ consists of the following two parts. The first part consists of an arc $a_\theta = (v_\theta, w_{\theta+\tau(a)})$ for each arc $a = (v, w)$ in A and each $\theta = 0, 1, \dots, T - \tau(a)$. Furthermore, the capacity of a_θ is equal to $c(a)$. The second part consists of an arc $(v(\theta), v(\theta + 1))$ for each vertex v in V and each $\theta = 0, 1, \dots, T - 1$. Furthermore, the capacity of $(v(\theta), v(\theta + 1))$ is infinite.

Let ξ be a static flow in the time-expanded network $\mathscr{T}$. By defining $f(a, \theta) := \xi(a_\theta)$ for each arc a in A and each $\theta = 0, 1, 2 \dots, T - \tau(a)$, we can construct a dynamic flow f in D with c and τ. Conversely, we can construct a static flow from a dynamic flow in the similar way. This observation implies that by defining $s = s_0$ and $t = t_T$, we can reduce the maximum dynamic flow problem to the maximum-flow problem. It should be noted that the size of the time-expanded network is exponentially larger than that of the input dynamic network.

3.3 Other Problems

In this section, we give other problems in a dynamic flow model.

Similarly to a static flow model, it is natural to consider the problem of finding a dynamic flow with minimum cost. More precisely, we are given a cost function $k \colon A \to \mathbb{R}_+$ and a demand $d \in \mathbb{R}_+$. Furthermore, we assume that S^+ and S^- consists of single vertices s^+ and s^-, respectively. For each dynamic flow f, we define its cost as

$$\sum_{a \in A} \sum_{\theta=0}^{T} k(a) \cdot f(a, \theta).$$

The goal of this problem is to find a dynamic flow f whose cost is minimum among all dynamic flows f' such that $\mathsf{ex}_{f'}(s^-, T) = d$. Unlike the minimum-cost flow problem in a static flow model, this problem is very hard. See [12] for details.

Furthermore, it is practically important to consider the case where there exist many kinds of objects. For example, there exist several kinds of people in evacuation situations. We can model this problem by using "multi-commodity" flow. There are many papers considering multicommodity flow problems in a static flow model. Thus, it is natural to consider a dynamic version of a multicommodity flow problem in a dynamic flow model. See [10] for details.

In the above problems, we implicitly assume that we can control movement of objects. However, if objects are people, then it is natural to consider that objects selfishly move. That is, it is natural to consider problems from the game theoretical viewpoint. There are many papers considering network flow problems in a static flow model from the game theoretical viewpoint. In a dynamic flow model, e,g, the paper [13] considers a dynamic flow problem from the game theoretical viewpoint.

4 Matchings

In this section, we consider matching problems that are used when we allocate jobs to workers and assign students to laboratories, and so on. See [16] for applications of matching problems. In the first half of this section, we consider an ordinary matching problem, called the maximum-size matching problem. In the second half of this section, we consider the stable matching problem in which each agent has a preference list edges, i.e., a matching problem in a strategic situation. See, e.g., [14, 16, 17] for coverage of concepts related to matching problems.

4.1 Maximum-Size Matchings

In this section, we consider the *maximum-size matching problem*. Intuitively speaking, in this problem, we try to find "pairs" as many as possible.

More formally, in the maximum-size matching problem, we are give an undirected graph $G = (V, E)$. A subset M of E is called a *matching*, if

$$\forall e, f \in M \text{ s.t. } e \neq f\colon\ e \cap f = \emptyset.$$

The maximum-size matching problem asks for finding a matching with maximum cardinality. See Fig. 5 for an example of a maximum-size matching. The matching in Fig. 5b is a maximum-size matching in the undirected graph illustrated in Fig. 5a. This problem can be efficiently solved. See, e.g., [3] for a efficient algorithm for the maximum-size matching problem.

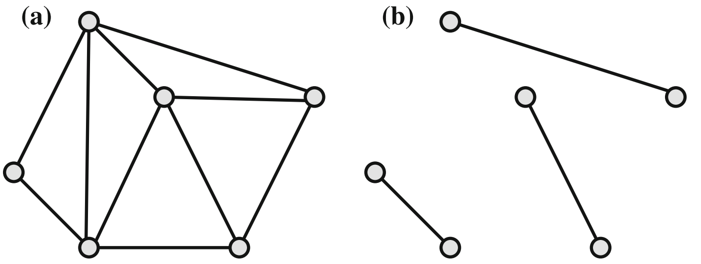

Fig. 5 **a** An undirected graph. **b** A maximum-size matching

In the maximum-size matching, the goal is to find a matching with maximum cardinality. If a "profit" is given for each edge, then it is natural to maximize the profit of a matching. This problem is called the *maximum-weight matching problem*. More formally, in the maximum-weight matching problem, we are given an undirected graph $G = (V, E)$ and a weight function $\sigma \colon E \to \mathbb{R}_+$. The *weight* of a matching M is defined as

$$\sum_{e \in M} \sigma(e).$$

The goal of the maximum-weight matching problem is to find a matching with maximum weight. This problem can be efficiently solved. See, e.g., [17] for details.

4.2 Stable Matchings

In this section, we explain the *stable matching problem*. Intuitively speaking, in this model, there exist two groups of agents and each agent has a preference ranking over members of the other group. The goal is to find a matching between these two groups with some specified properties.

More formally, the stable matching problem is defined as follows. We are given a bipartite graph $G = (V, E)$. We assume that V is partitioned into P and Q. For each vertex v in V, we are given a strict linear order $>_v$ that represents the preference of v. If $e >_v f$ for some edges e, f in $E(v)$, then v prefers e to f. See Fig. 6a for an example of the stable matching problem. In this example, we assume that

$$\begin{gathered}\{u, y\} >_u \{u, x\} \\ \{v, x\} >_v \{v, y\} >_v \{v, z\} \\ \{v, x\} >_x \{u, x\} \\ \{v, y\} >_y \{w, y\} >_y \{u, y\}.\end{gathered}$$

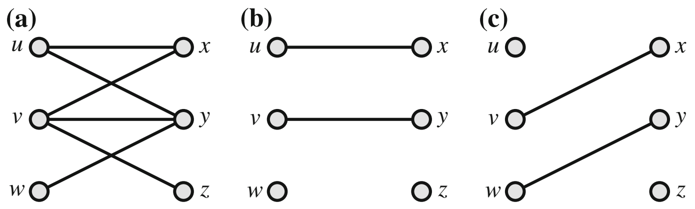

Fig. 6 **a** A bipartite graph. **b** An unstable matching. **c** A stable matching

Let M be a matching (see Sect. 4.1 for the definition of a matching). An edge e in $E \backslash M$ is said to be *free* on its end-vertex $v \in e$, if

- $M(v)$ is empty, or
- $e >_v f$, where $M(v) = \{f\}$.

We say that an edge e in $E \backslash M$ *blocks* M, if e is free on both vertices in e. A matching M is said to be *stable*, if there exists no edge in $E \backslash M$ blocking M. That is, if a given matching is not stable, then there is incentive for some pair to break a current matching.

It is not clear that there exists a stable matching in every instance of the stable matching problem. Gale and Shapley [7] proved that there always exists a stable matching and we can efficiently find it. For example, the matching in Fig. 6b is not stable since $\{v, x\}$ is a blocking pair. On the other hand, the matching in Fig. 6c is stable.

4.3 Other Problems

Here we explain other problems related to the stable matching problem.

In the stable matching problem, we try to find one-to-one matching. However, when we consider the problem of allocating jobs to workers or assigning residents to hospitals, it is natural to consider that one agent can be matched to more than one partners. That is, we consider a many-to-one or many-to-many matching. This problem is called that *hospital/residents problem*. More formally, in this problem, we are given the same input as the stable matching problem. Furthermore, we are given a capacity function $c\colon Q \to \mathbb{Z}_+$. A subset F of E is called an *assignment*, if $|F(p)| \leq 1$ for every vertex p in P and $|F(q)| \leq c(q)$ for every vertex q in Q. Let F be an assignment. An edge $e = \{p, q\}$ in $E \backslash F$ is said to be *free* on $q \in Q$, if

- $|M(q)| < c(q)$, or
- there exists an edge f in $M(q)$ with $e >_q f$.

We say that an edge e in $E \backslash F$ *blocks* F, if e is free on both vertices in e. An assignment F is said to be *stable*, if there exists no edge in $E \backslash F$ blocking F. It is known that there always exists a stable matching and we can efficiently find it. See [7] for details.

Next we consider a problem related to preference lists. In the stable matching problem, we are given a "strict" linear order as a preference list of each agent. That is, some agent v strictly prefer some edge in $E(v)$ to other edge. However, in many practical situations, it is natural to think that agents has preference lists with "ties." That is, in this case, it is possible that some agent v is "indifferent" between some edge in $E(v)$ and other edge. It is known that in this case, there always exists a stable matching, but a new issue arises. This is "Pareto efficiency" of a matching. This concept means that there exists no other matching improving some agent without hurting everyone else. See [4] for this topic.

Finally, we consider the popular matching problem introduced by Gärdenfors [8]. Recall that the concept of stability of a matching is "locally" defined. Thus, it is natural to consider a "global" fairness. The concept of popular matching is one of such concepts of global fairness. In the popular matching problem, we decide the order over matchings by "voting." More precisely, when we are given two matchings, we conclude that a matching for which much people vote is preferable. It is not clear that there always exists a popular matching, but Gärdenfors [8] proved that there always exists a popular matching. In fact, a stable matching is also a popular matching. Thus, the existence of a popular matching follows from that of a stable matching.

References

1. R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications* (Prentice Hall, Englewood Cliffs, 1993)
2. J. Bang-Jensen, G.Z. Gutin, *Digraphs: Theory, Algorithms and Applications* (Springer, New York, 2009)
3. J. Edmonds, Paths, trees, and flowers. Can. J. Math. **17**, 449–467 (1965)
4. A. Erdil, H. Ergin, What's the matter with tie-breaking? Improving efficiency in school choice. Am. Econ. Rev. **98**(3), 669–689 (2008)
5. L.R. Ford, D.R. Fulkerson, *Flows in Networks* (Princeton University Press, New Jersey, 1962)
6. L.R. Ford, D.R. Fulkerson, Constructing maximal dynamic flows from static flows. Oper. Res. **6**(3), 419–433 (1958)
7. D. Gale, L.S. Shapley, College admissions and the stability of marriage. Am. Math. Monthly **69**(1), 9–15 (1962)
8. P. Gärdenfors, Match making: assignments based on bilateral preferences. Behav. Sci. **20**(3), 166–173 (1975)
9. A.V. Goldberg, R.E. Tarjan, A new approach to the maximum-flow problem. J. ACM **35**(4), 921–940 (1988)
10. A. Hall, S. Hippler, M. Skutella, Multicommodity flows over time: efficient algorithms and complexity. Theoret. Comput. Sci. **379**(3), 387–404 (2007)
11. B. Hoppe, É Tardos, The quickest transshipment problem. Math. Oper. Res. **25**(1), 36–62 (2000)
12. B. Klinz, G.J. Woeginger, Minimum-cost dynamic flows: the series-parallel case. Networks **43**(3), 153–162 (2004)
13. R. Koch, M. Skutella, Nash equilibria and the price of anarchy for flows over time, in *Proceedings of tje 2nd International Symposium on Algorithmic Game Theory*, vol. 5814, Lecture Notes in Computer Science, pp. 323–334 (2009)
14. D. Manlove, *Algorithmics of matching under preferences* (World Scientific Publishing, Singapore, 2013)

15. J.B. Orlin, A faster strongly polynomial minimum cost flow algorithm. Oper. Res. **41**(2), 338–350 (1993)
16. A.E. Roth, A.O. Sotomayor, *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis* (Cambridge University Press, Cambridge, 1992)
17. A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency* (Springer, Berlin, 2003)
18. M. Skutella, An introduction to network flows over time, in *Research Trends in Combinatorial Optimization* (Springer, Berlin, 2009), pp. 451–482