# A Survey on Exact Algorithms for the Maximum Flow and Minimum-Cost Flow Problems

Oliverio Cruz-Mejía* Adam N. Letchford†

To appear in *Networks*

**Abstract**

Network flow problems form an important and much-studied family of combinatorial optimisation problems, with a huge array of practical applications. Two network flow problems in particular have received a great deal of attention: the maximum flow and minimum-cost flow problems. We review the progress that has been made on exact solution algorithms for these two problems, with an emphasis on worst-case running times.

**Keywords:** combinatorial optimisation; network flows

## 1 Introduction

Network flow problems have received a great deal of attention from the Operational Research and Optimisation communities, ever since Ford and Fulkerson published their influential textbook in 1962 [28]. In addition to their wealth of practical applications, network flow problems are also of great theoretical interest. For one thing, they can be viewed as "easy" combinatorial optimisation problems, in the sense that they can be solved in polynomial time. For another, they can be viewed as a rather simple kind of *linear program* (LP), in which the constraint matrix has a special structure. We refer the reader to the excellent textbooks [1, 79] for details.

Over the years, there have been many advances in exact algorithms for various network flow problems. In this paper, we review the progress that has been made on exact algorithms for two problems in particular: the *maximum flow* problem (MF for short) and the *minimum-cost flow* problem (MCF for short). Although MF is a special case of MCF, a large number of

*Department of Industrial Engineering, Universidad Nacional Autónoma de México, FES Aragón, México. E-mail: oliverio.cruz.mejia@comunidad.unam.mx

†Corresponding author. Department of Management Science, Lancaster University, Lancaster LA1 4YW, United Kingdom. E-mail: a.n.letchford@lancaster.ac.uk

exact algorithms have been devised specifically for MF. For this reason, we believe that it deserves special treatment.

The reader familiar with network flow problems will know that several other much-studied combinatorial optimisation problems can also be viewed as special cases of MCF. In particular, this includes the *shortest* $(s,t)$*-path*, *assignment*, *transportation* and *transshipment* problems. We will mention some of them in Subsection 3.2. The reader wanting more details is referred to [1, 13, 55, 70, 79].

The paper has a very simple structure. Section 2 surveys algorithm for MF, and Section 3 does the same for MCF.

We use the following (standard) notation throughout the paper. Let $G = (V, A)$ be a directed graph (or digraph). If an arc $a \in A$ goes from node $i$ to node $j$, we write $a = (i, j)$. We also write $n = |V|$ and $m = |A|$. The set of arcs leaving node $i$ is denoted by $\delta^+(i)$, and the set of arcs entering node $i$ is denoted by $\delta^-(i)$.

# 2 The Maximum Flow Problem

In this section, we survey exact algorithms for MF. Subsection 2.1 gives some key definitions and notation. Subsection 2.2 briefly reviews the "classical" algorithms, by which we mean all algorithms that came before the landmark paper of Goldberg & Rao [40] in 1998. Subsection 2.3 reviews the subsequent combinatorial algorithms. Finally, Subsection 2.4 considers some very different algorithms based on interior-point methods.

## 2.1 Definitions and notation

The maximum flow problem was introduced in Ford & Fulkerson [27]. We have a directed graph $G = (V, A)$. One node $s \in V$ is called the *source*, and another node $t \in V$ is called the *sink*. Each arc $a \in A$ has a *capacity* $u_a \in \mathbb{Q}_+$. The task is to send as much flow as possible from the source to the sink.

Without loss of generality, we assume that all capacities are integers. We also let $U = \max_{a \in A}\{u_a\}$ denote the maximum capacity. If the running time of an MF algorithm is bounded by a polynomial in $m$, $n$ and $\log U$, the algorithm is called *polynomial*. If the running time is bounded by a polynomial in $m$ and $n$ only, the algorithm is called *strongly polynomial*. We call a directed path from $s$ to $t$ in $G$ an *$(s,t)$-path*.

The material in the remainder of this subsection comes from [28]. For $a \in A$, let $x_a$ be a non-negative variable, representing the amount of flow sent through the arc $a$. MF can then be formulated as the following LP:

$$\begin{aligned} \max \quad & \textstyle\sum_{a \in \delta^+(s)} x_a && (1) \\ \text{s.t.} \quad & \textstyle\sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = 0 \quad (i \in V \setminus \{s,t\}) && (2) \end{aligned}$$

$$0 \leq x_a \leq u_a \qquad (a \in A). \tag{3}$$

The objective function (1) is to maximise the total amount of flow leaving the source node. The constraints (2) ensure that, for each vertex apart from the source and sink, the total amount of flow entering and leaving are equal. The constraints (3) ensure that flows are non-negative and arc capacities are not exceeded.

Note that the constraints (2) have a very special structure: each variable appears exactly once with a coefficient of $+1$, and exactly once with a coefficient of $-1$. For this reason, it is usually much more efficient to solve MF with a specialised algorithm, rather than using a generic LP algorithm (such as the simplex method).

A vector $\bar{x}$ satisfying (2) and (3) is called a *flow*. Given a flow, we can construct a digraph, called the *residual graph*, as follows. The vertex set is $V$. The arc set, which we call $\bar{A}$, is defined as follows. For each arc $(i, j) \in A$ such that $\bar{x}_a < u_a$, we place an arc $(i, j)$ in $\bar{A}$ with capacity $u_a - \bar{x}_a$. For each arc $(i, j) \in A$ such that $\bar{x}_a > 0$, we place an arc $(j, i)$ in $\bar{A}$ with capacity $\bar{x}_a$. The two types of arcs are called *forward* and *reverse* arcs, respectively.

Any $(s, t)$-path in the residual graph is called an *augmenting* path (with respect to $\bar{x}$). Given any augmenting path $D \subseteq \bar{A}$, we can increase the amount of flow in $G$ as follows. Let $\delta$ be the smallest capacity among the arcs in the path. If $a \in D$ is a forward arc, then increase $\bar{x}_a$ by $\delta$. If it is a reverse arc, then decrease $\bar{x}_a$ by $\delta$. A flow $\bar{x}$ is maximum if and only if there exists no augmenting path.

Now consider the subgraph of the residual graph which contains only the forward arcs. If this subgraph does not contain any augmenting path, $\bar{x}$ is called a *blocking flow*. All maximum flows are blocking, but the reverse is not true.

## 2.2 Classical algorithms

As mentioned above, we call an MF algorithm "classical" if it appeared before the publication of [40]. (This is because the appearance of [40] was regarded as a major breakthrough in the field—see the next subsection.)

There exist several excellent surveys on the classical algorithms (see, e.g., [1, 45, 70]). For this reason, we mention only deterministic algorithms here, and we include only algorithms that included fundamental new ideas, and algorithms that had a faster running time than at least one earlier algorithm. These algorithms are summarised in Table 1.

The first exact algorithm for MF was introduced by Ford & Fulkerson [28]. Their algorithm was based on iteratively improving flows along augmenting paths. It performs $O(nU)$ augmentations and runs in $O(mnU)$ time. Note that this running time is pseudo-polynomial rather than polynomial. That is, it is polynomial only when $U$ is itself bounded by a polynomial in $m$ and $n$.

| Year | Bound | Reference |
|---|---|---|
| 1962 | $O(mnU)$ | Ford & Fulkerson [28] |
| 1969 | $O(m^2 n)$ | Edmonds & Karp [25, 26] |
| 1969 | $O(m^2 \log U)$ | Edmonds & Karp [25, 26] |
| 1970 | $O(mn^2)$ | Dinic [24] |
| 1970 | $O(mn \log U)$ | Dinic [24] |
| 1974 | $O(n^3)$ | Karzanov [50] |
| 1977 | $O(n^2\sqrt{m})$ | Cherkassky [20] |
| 1978 | $O(m^{2/3}n^{5/3})$ | Galil [34, 35] |
| 1978 | $O(mn \log^2 n)$ | Shiloach [72], Galil & Naamad [36] |
| 1980 | $O(mn \log n)$ | Sleator & Tarjan [73, 74] |
| 1986 | $O(mn \log (n^2/m))$ | Goldberg & Tarjan [41, 43] |
| 1989 | $O(mn + n^2 \log U)$ | Ahuja & Orlin [5] |
| 1989 | $O(mn + n^2\sqrt{\log U})$ | Ahuja *et al.* [6] |
| 1989 | $O\left(mn \log\left(\frac{n\sqrt{\log U}}{m}\right)\right)$ | Ahuja *et al.* [6] |
| 1990 | $O(n^3/\log n)$ | Cheriyan *et al.* [18, 19] |
| 1990 | $O(mn + n^{8/3} \log n)$ | Cheriyan & Hagerup [17], Alon [7] |
| 1992 | $O(mn + n^{2+\epsilon})$ | King *et al.* [52] |
| 1993 | $O\left(mn \frac{\log n}{\log(m/n)} + n^{2+\epsilon}\right)$ | Phillips & Westbrook [67, 68] |
| 1994 | $O\left(\frac{mn \log n}{\log\left(m/(n \log n)\right)}\right)$ | King *et al.* [53] |

Table 1: Some "classical" deterministic maximum flow algorithms.

Edmonds & Karp [25, 26] discovered two different ways to improve the augmenting-path approach:

1. *Shortest augmenting paths*: At any given stage in the Ford-Fulkerson algorithm, there may exist more than one augmenting path. Edmonds and Karp proved that, if one always uses an augmenting path having the least number of arcs, then only $O(mn)$ flow augmentations are needed. This reduces the running time to $O(m^2n)$, which is strongly polynomial.

2. *Capacity scaling:* We begin by selecting some positive constant $\Delta$. We then run the Ford-Fulkerson algorithm, but ignore any arc in the residual graph that has capacity less than $\Delta$. When the algorithm terminates, we halve the value of $\Delta$, and repeat. When $\Delta$ becomes smaller than 1, we can stop. With an appropriate choice of $\Delta$, this approach gives $O(m \log U)$ augmentations. The resulting running time is $O(m^2 \log U)$ time, which is (weakly) polynomial.

The next important development was due to Dinic [24]. He observed that one can construct a *blocking flow* directly, in $O(mn)$ time, without any considering of augmenting paths. This led him to propose a variant of the first Edmonds-Karp algorithm which begins by constructing a blocking flow, and then iteratively searches for blocking flows in the residual graph. He proved that this variant terminates after only $O(n)$ augmentations. Since each augmentation takes $O(mn)$ time, his algorithm runs in $O(mn^2)$ time. Applying a similar idea to the second Edmonds-Karp algorithm, Dinic also found an algorithm that performs $O(\log U)$ augmentations, leading to a running time of $O(mn \log U)$.

After the publication of Dinic [24], several faster algorithms were found for computing blocking flows. These included the $O(n^2)$ algorithm of Karzanov [50], the $O(n\sqrt{m})$ algorithm of Cherkasky [20], the $O((mn)^{2/3})$ algorithm of Galil [34, 35], the $O(m \log^2 n)$ algorithms of Shiloach [72] and Galil & Naamad [36], and the $O(m \log n)$ algorithm of Sleator & Tarjan [73, 74]. These improvements led to corresponding speed-ups in the Dinic algorithm. Thus, by the start of the 1980s, it was known that MF could be solved in either $O(n^3)$ or $O(mn \log n)$ time, whichever is the faster.

The next significant advance was made in 1986, when Goldberg & Tarjan [41, 43] proposed the *preflow-push* approach. This approach uses the notion of *pre-flows*, first defined in [50]. Roughly speaking, a pre-flow is a relaxation of a flow, in which the total flow entering a node may exceed the total flow leaving that node. The difference between the entering and leaving flows is called the "excess". Goldberg & Tarjan's algorithm begins by constructing a "blocking pre-flow", and then repeatedly tries to "push" excess flow from a node to adjacent nodes that are estimated to be closer to the sink node.

Any excess flow that cannot be routed to the sink is eventually sent back to the source.

Goldberg and Tarjan showed that only $O(mn)$ "pushes" are needed. They then showed that, with the help of a data structure from [73, 74], their approach could be implemented so that each "push" takes only $O(\log(n^2/m))$ time. This particular implementation therefore runs in $O(mn \log(n^2/m))$ time.

The remaining eight algorithms listed in Table 1, dated 1989–1994, are all variants of the preflow-push algorithm. Some of them use remarkably sophisticated data structures. For brevity, we do not go into details. We remark however that the algorithm of Cheriyan *et al.* [18, 19] is rather unusual, in that it starts with a sparse subgraph and then adds the remaining edges as needed. Moreover, it was the first ever MF algorithm to run in $o(n^3)$ time on dense graphs.

To close this subsection, we remark that the worst-case running time is not the only criterion by which one may evaluate an algorithm. Some algorithms are easier to implement than others, and some perform better than their worst-case bound might suggest. For thorough empirical comparisons of various classical MF algorithms, see [4, 15, 21].

### 2.3 The "O(mn) barrier"

The observant reader will have noticed that, in the 1980s and early 1990s, authors were coming closer and closer to obtaining an algorithm running in $O(mn)$ time, but never quite achieving it. Actually, as pointed out in [40], $\Omega(mn)$ is a very natural "barrier" for MF algorithms. Indeed, every maximum flow can be expressed as a non-negative linear combination of $(s,t)$-paths, and each of those $(s,t)$-paths may contain up to $n-1$ arcs. Thus, any MF algorithm that uses augmenting paths explicitly must take $\Omega(mn)$ time.

The first algorithm to break through the barrier arose in 1998, with the publication of Goldberg & Rao [40]. The algorithm exploits an earlier result, proved in [43, 49], that the blocking-flow algorithm of Dinic [24] needs no more than $O(\min\{n^{2/3}, m^{1/2}\})$ augmentations in the *unit capacity* case (i.e., the case in which $u_a = 1$ for all $a \in A$ and there are no parallel arcs). Combined with capacity scaling, along with an some ingenious data structures, they obtain an MF algorithm which solves a sequence of $O(\log(n^2/m) \log U)$ unit-capacity MF subproblems. This yields an overall running time of

$$O\Big(m \min\{n^{2/3}, m^{1/2}\} \log(n^2/m) \log U\Big).$$

Provided that $U$ is not extremely large, this running time breaks the $O(mn)$ barrier. For an empirical study of the algorithm, see [46].

| Year | Bound | Reference |
|---|---|---|
| 1998 | $O\Big(m \min\{n^{2/3}, m^{1/2}\} \log\left(n^2/m\right) \log U\Big)$ | Goldberg & Rao [40] |
| 2013 | $O\big(mn + m^{31/16} \log^2 n\big)$ | Orlin [65] |
| 2021 | $O\left(\frac{mn \log n}{\log\log n + \log(m/n)}\right)$ | Orlin & Gong [66] |

Table 2: More recent deterministic maximum flow algorithms.

The Goldberg-Rao algorithm is a weakly polynomial-time algorithm. This left open the question of whether there exists a strongly polynomial-time algorithm running in $O(mn)$ time or less. This was answered in 2013 by Orlin [65]. He found a variant of the Goldberg-Rao algorithm which works on a series of "contracted" graphs and runs in $O(mn + m^{31/16} \log^2 n)$ time. Together with the result of King *et al.* [53] mentioned in the previous subsection, this implies that MF can be solved in $O(mn)$ time. (It suffices to run Orlin's algorithm for sparse graphs and the King *et al.* algorithm for dense graphs.)

Later on, Orlin & Gong [66] obtained a running time of $O\left(\frac{mn \log n}{\log\log n + \log(m/n)}\right)$, using an improved version of the algorithm of Ahuja *et al.* [6]. This running time is very slightly better than the one of King *et al.* [53].

The algorithms mentioned in this subsection are summarised in Table 2.

## 2.4 Interior-point methods

Recently, some rather different MF algorithms have been developed, based on interior-point methods (IPMs). These algorithms are randomised, but they find the optimal solution to any specified accuracy with high probability. Moreover, as we will see, some of them can be derandomised with only a small increase in running time. For a summary of these algorithms, see Table 3. In this table, 'polylog $n$' means $\log^k n$ for some constant $k \geq 1$, and '$o(1)$' means a constant that tends to zero as $n$ tends to infinity.

Before going into details on these algorithms, we give a bit of historical context. In 2004, Spielman & Teng [75] gave a fast randomised algorithm for finding approximate solutions to systems of linear equations that have a "symmetric and diagonally dominant" (sdd) constraint matrix. (Such linear systems, called "Laplacian" systems, have a wide range of applications.) Their algorithm runs in $O(m \text{ polylog} n)$ time, where $n$ is the number of variables and $m$ is the number of non-zeroes in the matrix. A series of improvements quickly followed. To our knowledge, the fastest known randomized algorithm for the problem is the one of [48], which runs in $O\Big(m(\log\log n)^k\Big)$ time for some positive constant $k$. There is also a deterministic version [22], which runs in $O(m^{1+o(1)})$ time.

Armed with this background, we can now make some remarks about the

| Year | Bound | Reference |
|---|---|---|
| 2008 | $O\big(m^{3/2}\,\text{polylog}\,n\,\log U\big)$ | Daitch & Spielman [23] |
| 2014 | $O\big(m\sqrt{n}\,\text{polylog}\,n\,\log^2 U\big)$ | Lee & Sidford [57] |
| 2016 | $O\big(m^{10/7}\,\text{polylog}\,n\,U^{1/7}\big)$ | Madry [59] |
| 2020 | $O\big(m^{11/8}\,\text{polylog}\,n\,U^{1/4}\big)$ | Liu & Sidford [58] |
| 2020 | $O\big(m^{4/3+o(1)}\,U^{1/3}\big)$ | Kathuria *et al.* [51] |
| 2021 | $O\Big(\big(m+n^{1.5}\big)\,\text{polylog}\,n\,\log U\Big)$ | van den Brand *et al.* [12] |
| 2021 | $O\big(m^{3/2-1/328}\,\text{polylog}\,n\,\log U\big)$ | Gao *et al.* [39] |
| 2022 | $O\big(m^{3/2-1/58}\,\text{polylog}\,n\,\log U\big)$ | van den Brand *et al.* [11] |
| 2022 | $O\big(m^{1+o(1)}\log U\big)$ | Chen *et al.* [16] |

Table 3: Overview of recent randomised maximum flow algorithms.

new MF algorithms:

- Daitch & Spielman [23] devised a specialised IPM for LPs with sdd constraint matrices, which uses the algorithm of Spielman & Teng [75] as a subroutine. They then show that the max-flow LP (1)-(3) can be converted into an LP of the desired form, using a certain linear transformation. Note that the running time of the Daitch-Spielman algorithm is comparable to that of the one by Goldberg & Rao [40] mentioned above.

- Lee & Sidford [57] presented an improved IPM for LPs with sdd constraint matrices, which uses random sampling to sparsify the constraint matrix. The IPM requires only $O(\sqrt{r}L)$ major iterations, where $r$ is the rank of the constraint matrix and $L$ is its bit complexity. When specialised to MF, their algorithm obtains the running time stated in the table.

- The approach of Lee and Sidford was improved by Van den Brand *et al.* [12, 11] and Gao *et al.* [39].

- Madry [59] presented a rather different approach, based on three main observations: (i) the max-flow problem is similar to the problem of sending electrical current through a network of resistors while consuming the minimum amount of energy (although the latter has a quadratic objective function instead of a linear one); (ii) this electrical flow problem can be solved quickly via a reduction to a series of Laplacian system solves; and (iii) one can construct a maximum flow via a series of electrical flow problems, rather than by a series of blocking flow problems as in the classical algorithms. We remark that Madry's algorithm runs in pseudo-polynomial time.

- Variants of Madry's algorithm can be found in Liu & Sidford [58] and Kathuria *et al.* [51]. These algorithms are pseudo-polynomial as well.

- Finally, and most recently, Chen *et al.* [16] found a randomised MF algorithm that runs in only $O(m^{1+o(1)} \log U)$ time. The algorithm is however extremely complex, using concepts and data structures from traditional MF algorithms, Laplacian systems and IPMs simultaneously.

To close this section, we remark that the Daitch-Spielman and Madry algorithms can be de-randomised, using the deterministic Laplacian system solver mentioned above. This comes at the cost of an additional factor of $n^{o(1)}$ in the running time. An interesting open question is whether any of the other IPM algorithms can be de-randomised, without incurring a significant increase in running time.

# 3 The Minimum-Cost Flow Problem

Next, we look at exact algorithms for MCF. Subsection 3.1 gives some key definitions and notation. Subsection 3.2 mentions some key problems that often arise as subproblems when solving MCF. Subsection 3.3 recalls some of the earliest MCF algorithms, which run in pseudo-polynomial time. Finally, Subsection 3.4 covers polynomial-time algorithms.

## 3.1 Definitions and notation

In MCF, we have a digraph $G = (V, A)$. Associated with each node $i \in V$ is an integer $b_i$. If $b_i > 0$, $i$ is called a supply node. If $b_i < 0$, $i$ is called a demand node. It is assumed that $\sum_{i \in V} b_i = 0$. Each arc $a \in A$ has a cost $c_{ij} \in \mathbb{Z}$ and a capacity $u_a \in \mathbb{Z}_+$. (Note that arc costs are usually permitted to be negative.) The task is to send flow through the network in such a way that the flow leaving each supply node $i$ is equal to $b_i$, the flow arriving at each demand node $i$ is equal to $b_i$, and the amount of flow through any given arc $a$ does not exceed the capacity.

In the standard LP formulation of MCF, one uses a non-negative variable $x_a$ for each arc $a \in A$, just as in the case of MF. The LP is then:

$$\begin{aligned} \min \quad & \textstyle\sum_{a \in A} c_a x_a & & (4) \\ \text{s.t.} \quad & \textstyle\sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = b_i & (i \in V) & (5) \\ & 0 \leq x_a \leq u_a & (a \in A). & (6) \end{aligned}$$

The objective function (4) is to minimise the total cost. The constraints (5) ensure that the supplies and demands are met. The constraints (6) bound the flow through each arc.

As in the previous section, we let $U$ denote the maximum arc capacity. We will also find it helpful to let $C$ denote $\max_{a \in A} \{|c_a|\}$.

### 3.2 Key subproblems

It turns out that many of the MCF algorithms in the literature call on subroutines for other simpler problems, such as (a) the maximum flow problem, (b) the shortest $(s,t)$-path problem, (c) the transshipment problem, and (d) the problem of detecting negative-cost cycles in digraphs. To make this paper self-contained, we briefly recall some results on the last three of the problems mentioned.

In the *shortest $(s,t)$-path* problem, we have a digraph $G$, two specified nodes $s$ and $t$, and a cost vector $c \in \mathbb{Q}^m$. The task is find a path from $s$ to $t$ of minimum total cost. This problem has a long history, and a wide variety of algorithms have been proposed for it. Algorithms have also been found for various special cases. A good survey is given in [70]. For brevity, we mention only three algorithms. Dijkstra's algorithm, which is designed for the case in which $c \in \mathbb{Q}_+^m$, can be implemented to run in $O(m+n\log n)$ time [29]. The algorithm of Thorup [77], which makes the stricter assumption that $c \in \mathbb{Z}_+^m$, runs in $O(m + n\log\log C)$ time. Finally, the algorithm in [9], which assumes that $c \in \mathbb{Z}^m$, runs in $O(m \operatorname{polylog} n \log W)$ time, where $W$ is the most-negative arc cost.

The *transshipment* problem, mentioned in the introduction, is the variant of MCF in which no arc capacities are present (or, equivalently, all arc capacities are very large) [61]. A folklore result states that MCF can be reduced to transshipment by subdividing each arc into two, and adjusting the supplies, demands and costs appropriately (see [55]). Note that this transformation increases the number of nodes from $n$ to $n+m$, which can have an effect on the worst-case running time of various solution algorithms.

Finally, in the *negative-cycle detection* problem, we have a digraph $G$ and a cost vector $c \in \mathbb{Z}^m$. The task is to check whether there exists a directed cycle in $G$ whose total cost is negative. The problem can be solved in $O(mn)$ time using the Bellman-Ford algorithm (see [1]).

### 3.3 Pseudo-polynomial MCF algorithms

We now recall the "classical" MCF algorithms, all of which run in pseudo-polynomial time. These algorithms are summarised in Table 4.

In this table, "MF", "SP" and "NC" stand for "maximum flow", "shortest-path" and "negative-cost cycle", respectively. So, for example, the first MCF algorithm in the table uses a shortest-path algorithm as a subroutine, and it calls that subroutine $O(mn^2C)$ times. As in the previous section, we only include algorithms that include fundamental new ideas, and algorithms that had a faster running time than at least one earlier algorithm.

We now make some remarks about each of these algorithms:

- Iri [47] and Busacker & Gowen [14] developed what is now known as the "successive shortest path" approach. It is a "primal-dual" ap-

| Year | Bound | Reference |
|---|---|---|
| 1960 | $O\big(mn^2C\big)$ SP | Iri [47], Busacker & Gowen [14] |
| 1960 | $O(mU)$ SP | Minty [60], Fulkerson [30] |
| 1962 | $O\big(n \min\{U, C\}\big)$ SP & MF | Ford & Fulkerson [28] |
| 1967 | $O(mC)$ NC | Klein [54] |
| 1969 | $O(nU)$ SP | Edmonds & Karp [25] |

Table 4: Classical pseudo-polynomial algorithms for MCF.

proach, in the sense that it maintains dual optimality from the start and then strives to attain primal feasibility. It involves the solution of $O(mn^2C)$ shortest-path problems in which negative edge-lengths may appear. Later on, Tomizawa [78] showed that one can restrict oneself to shortest-path problems with non-negative edge lengths.

- Minty [60] and Fulkerson [30] independently developed the so-called "out-of-kilter" method. It is an unusual method as it does not need primal or dual solutions to be feasible in its early stages. It involves the solution of $O(mU)$ shortest-path problems.

- Ford & Fulkerson [28] developed a primal-dual algorithm that has $O(n \min\{U, C\})$ major iterations. In each such iteration, a shortest-path problem and a maximum flow problem is solved.

- Klein [54] presented a primal algorithm, based on iteratively cancelling negative-cost cycles. Although he did not give a formal analysis of the running time, the number of major iterations must be $O(mC)$, since the cost of the flow decreases by at least one in each iteration.

- Edmonds & Karp [25] found an improved out-of-kilter algorithm that does not need to solve max-flow problems. It requires the solution of $O(nU)$ shortest-path problems.

We remark that Klein's algorithm appeared implicitly in the 1940s, in the works of Leonid Kantorovich (see [71]).

### 3.4 Polynomial MCF algorithms

In the 1970s and 1980s, several polynomial-time algorithms were developed for MCF. The main developments are summarised in Table 5. We now make some remarks about these algorithms:

- Edmonds & Karp [26] used capacity scaling (see Subsection 2.2) to obtain a faster out-of-kilter algorithm that involves only $O(m \log U)$ shortest-path computations.

| Year | Bound | Reference |
|---|---|---|
| 1972 | $O(m \log U)$ SP | Edmonds & Karp [26] |
| 1980 | $O(m \log C)$ MF | Röck [69] |
| 1984 | $O\big(m^2 \log n\big)$ SP | Orlin [62], Fujishige [31] |
| 1985 | $O\big(m^2 \log n\big)$ MF | Tardos [76] |
| 1986 | $O\big(n^2 \log n\big)$ SP | Galil & Tardos [37, 38] |
| 1987 | $O\big(mn \log n \log U \log(nC)\big)$ | Gabow & Tarjan [32, 33] |
| 1987 | $O\big(mn \log (n^2/m) \log (nC)\big)$ | Goldberg & Tarjan [42, 44] |
| 1988 | $O(m \log n)$ SP | Orlin [63, 64] |
| 1988 | $O\big(mn \log \log U \log(nC)\big)$ | Ahuja *et al.* [2, 3] |

Table 5: Some polynomial-time algorithms for MCF.

- Röck [69] used "cost scaling" to obtain a primal-dual algorithm that involves $O(m \log C)$ max-flow computations. (Cost scaling is similar to capacity scaling, except that one iteratively increases the precision of the costs, rather than the capacities.)

- Orlin [62] and Fujishige [31] showed how to reduce the transshipment problem to the solution of $O(n^2 \log n)$ shortest-path problems. Using the transformation mentioned above, this shows that MCF can be reduced to $O(m^2 \log n)$ shortest-path problems in a digraph with $O(m + n)$ nodes and arcs.

- Around the same time, Tardos [76] found a different strongly polynomial-time algorithm which involves the solution of $O(m^2 \log n)$ max-flow problems.

- Galil & Tardos [37, 38] showed that, in fact, only $O(n^2 \log n)$ shortest-path problems are needed even when capacities are present.

- Gabow & Tarjan [32, 33] devised an algorithm based on a combination of capacity scaling, cost scaling, and a reduction to a series of transportation problems. The running time is $O\Big(mn \log n \log U \log(nC)\Big)$.

- Goldberg & Tarjan [42, 44] developed a new primal-dual approach that uses cost scaling. Using an appropriate data structure, they obtain a running time of $O(mn \log (n^2/m) \log (nC))$.

- Orlin [63, 64] presented a new approach based on a relaxed notion of flow called a *pseudo-flow*. His approach enables one to solve the transshipment problem with only $O(n \log n)$ shortest-path calls. Using the transformation mentioned above, this shows that MCF can be reduced to $O(m \log n)$ shortest-path problems in a digraph with $O(m+n)$ nodes and arcs.

| Year | Bound | Reference |
|---|---|---|
| 2008 | $O\big(m^{3/2}\,\text{polylog}\,n\,\log U\big)$ | Daitch & Spielman [23] |
| 2014 | $O\big(m\sqrt{n}\,\text{polylog}\,n\,\log^2 U\big)$ | Lee & Sidford [57] |
| 2021 | $O\Big(\big(m\log(UC)+n^{1.5}\log^2(UC)\big)\,\text{polylog}\,n\Big)$ | van den Brand *et al.* [12] |
| 2021 | $O\big(m^{3/2-1/762}\,\text{polylog}\,n\,\log(U+C)\big)$ | Axiotis *et al.* [8] |
| 2022 | $O\big(m^{3/2-1/58}\,\text{polylog}\,n\,\log^2 U\big)$ | van den Brand *et al.* [11] |
| 2022 | $O\big(m^{1+o(1)}\log U\log C\big)$ | Chen *et al.* [16] |

Table 6: Overview of recent randomised algorithms for MCF.

- Ahuja *et al.* [2, 3] combined the best features of the capacity-scaling and cost-scaling approaches, together with sophisticated data structures, to yield an algorithm running in $O(mn\log\log U\log(nC))$ time.

For empirical comparisons of various MCF algorithms, see [10, 56].

Finally, we mention that some of the interior-point MF algorithms, mentioned at the end of the previous section, can be used for MCF as well. This does however sometimes come at the cost of a slightly increased running time. For a summary, see Table 6. We remark that the Daitch-Spielman algorithm can be derandomised, as mentioned in the previous section.

# References


[1] R.K. Ahuja, T.L. Magnanti & J.B. Orlin (1993) *Network Flows*. Englewood Cliffs, NJ: Prentice-Hall.

[2] R.K. Ahuja, A.V. Goldberg, J.B. Orlin & R.E. Tarjan (1988) Finding minimum-cost flows by double scaling. *Technical Report 164-88*, Department of Computer Science, Princeton University.

[3] R.K. Ahuja, A.V. Goldberg, J.B. Orlin & R.E. Tarjan (1992) Finding minimum-cost flows by double scaling. *Math. Program.*, 53, 243–266.

[4] R.K. Ahuja, M. Kodialam, A.K. Mishra & J.B. Orlin (1997) Computational investigations of maximum flow algorithms. *Eur. J. Oper. Res.*, 97, 509–542.

[5] R.K. Ahuja & J.B. Orlin (1989) A fast and simple algorithm for the maximum flow problem. *Oper. Res.*, 37, 748–759.

[6] R.K. Ahuja, J.B. Orlin & R.E. Tarjan (1989) Improved time bounds for the maximum flow problem. *SIAM J. Comput.*, 18, 939–954.

[7] N. Alon (1990) Generating pseudo-random permutations and maximum flow algorithms. *Inf. Proc. Lett.*, 35, 201–204.

[8] K. Axiotis, A. Madry & A. Vladu (2021) Faster sparse minimum cost flow by electrical flow localization. In *Proc. FOCS '21*, pp. 528–539. Piscataway, NJ: IEEE.

[9] A. Bernstein, D. Nanongkai & C. Wulff-Nilsen (2022) Negative-weight single-source shortest paths in near-linear time. In *Proc. FOCS '22*, pp. 600–611. Piscataway, NJ: IEEE.

[10] R.G. Bland, J. Cheriyan, D. Jensen & L. Ladányi (1993) An empirical study of min cost flow algorithms. In D.S. Johnson & C.C. McGeoch (eds) *Network Flows and Matching: First DIMACS Implementation Challenge*, pp. 119–156. Providence, RI: AMS.

[11] J. van den Brand, Y. Gao, A. Jambulapati, Y.T. Lee, Y.P. Liu, R. Peng & A. Sidford (2022) Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Proc. STOC '22*, pp. 543–556. New York: ACM.

[12] J. van den Brand, Y.T. Lee, Y.P. Liu, T. Saranurak, A. Sidford, Z. Song & D. Wang (2021) Minimum cost flows, MDPs, and $\ell_1$-regression in nearly linear time for dense instances. In *Proc. STOC '21*, pp. 859–869. New York: ACM.

[13] R. Burkard, M. Dell'Amico & S. Martello (2012) *Assignment Problems*. Philadelphia, PA: SIAM.

[14] R.G. Busacker & P.J. Gowen (1960) A procedure for determining a family of minimum cost network flow patterns. *Technical paper OCO-TP-15*, Operations Research Office, John Hopkins University.

[15] B.G. Chandran & D.S. Hochbaum (2009) A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Oper. Res.*, 57, 358–376.

[16] L. Chen, R. Kyng, Y.P. Liu, R. Peng, M.P. Gutenberg & S. Sachdeva (2022) Maximum flow and minimum-cost flow in almost-linear time. *arXiv preprint 2203.00671*.

[17] J. Cheriyan & T. Hagerup (1989) A randomized maximum-flow algorithm. In *Proc. FOCS '89*, pp. 118–123. Piscataway, NJ: IEEE.

[18] J. Cheriyan, T. Hagerup & K. Mehlhorn (1990) Can a maximum flow be computed in $o(nm)$ time? In M.S. Paterson (ed.) *Proc. ICALP '90*, pp. 35–248. Berlin: Springer.

[19] J. Cheriyan, T. Hagerup & K. Mehlhorn (1996) An $o(n^3)$-time maximum-flow algorithm. *SIAM J. Comput.*, 25, 1144–1170.

[20] B.V. Cherkassky (1977) Algorithm for construction of maximal flows in networks with complexity of $O(V^2\sqrt{E})$ operations (in Russian). *Mathematical Methods of Solution of Economical Problems*, 7, 112–125.

[21] B.V. Cherkassky & A.V. Goldberg (1997) On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 19, 390–410.

[22] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng & T. Saranurak (2020) A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Proc. FOCS '20*, pp. 1158–1167. Piscataway, NJ: IEEE.

[23] S.I. Daitch & D.A. Spielman (2008) Faster approximate lossy generalized flow via interior point algorithms. In *Proc. STOC '08*, pp. 451–460. New York: ACM.

[24] E.A. Dinic (1970) Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Doklady*, 11, 1277–1280.

[25] J. Edmonds & R.M. Karp (1969) Theoretical improvements in algorithmic efficiency for network flow problems. Presented at *Calgary International Conference on Combinatorial Structures and Their Applications*, Calgary, AB.

[26] J. Edmonds & R.M. Karp (1972) Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19, 248–264.

[27] L.R. Ford & D.R. Fulkerson (1956) Maximal flow through a network. *Canadian J. Math.*, 8, 399–404.

[28] L.R. Ford & D.R. Fulkerson (1962) *Flows in Networks*. Princeton: Princeton University Press.

[29] M.L. Fredman & R.E. Tarjan (1984) Fibonacci heaps and their uses in improved network optimization algorithms. In *Proc. FOCS '84*, pp. 338–346). IEEE Computer Society.

[30] D.R. Fulkerson (1961) An out-of-kilter method for minimal-cost flow problems. *J. SIAM*, 9, 18–27.

[31] S. Fujishige (1986) An $O(m^3 \log n)$ capacity-rounding algorithm for the minimum cost circulation problem: a dual framework of Tardos' algorithm. *Math. Program.*, 35, 298–309.

[32] H.N. Gabow & R.E. Tarjan (1987) Faster scaling algorithms for network problems. *Technical Report 111-87*, Department of Computer Science, Princeton University.

[33] H.N. Gabow & R.E. Tarjan (1989) Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18, 1013–1036.

[34] Z. Galil (1978) A new algorithm for the maximal flow problem. *Proc. FOCS '78*, pp. 231–245. Piscataway, NJ: IEEE.

[35] Z. Galil (1980) An $O(|V|^{5/3}|E|^{2/3})$ algorithm for the maximal flow problem. *Acta Informatica*, 14, 221–242.

[36] Z. Galil & A. Naamad (1980) An $O(EV \log^2 V)$ algorithm for the maximal flow problem. *J. Comput. Sys. Sci.*, 21, 203–217.

[37] Z. Galil & E. Tardos (1986) An $O(n^2(m + n \log n) \log n)$ min-cost flow algorithm. In *Proc. FOCS '86*, pp. 1–9. Piscataway, NJ: IEEE.

[38] Z. Galil & E. Tardos (1988) An $O(n^2(m + n \log n) \log n)$ min-cost flow algorithm. *J. ACM*, 35, 374–386.

[39] Y. Gao, Y.P. Liu & R. Peng (2021) Fully dynamic electrical flows: sparse maxflow faster than Goldberg-Rao. In *Proc. FOCS '21*, pp. 516–527. Piscataway, NJ: IEEE.

[40] A.V. Goldberg & S. Rao (1998) Beyond the flow decomposition barrier. *J. ACM*, 45, 783–797.

[41] A.V. Goldberg & R.E. Tarjan (1986) A new approach to the maximum-flow problem. *Proc. STOC '86*, 136–146. New York: ACM.

[42] A.V. Goldberg & R.E. Tarjan (1987) Solving minimum-cost flow problems by successive approximation. In *Proc. STOC '87*, pp. 7–18. New York: ACM.

[43] A.V. Goldberg & R.E. Tarjan (1988) A new approach to the maximum-flow problem. *J. ACM*, 35, 921–940.

[44] A.V. Goldberg & R.E. Tarjan (1990) Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15, 430–466.

[45] A.V. Goldberg & R.E. Tarjan (2014) Efficient maximum flow algorithms. *Commun. ACM*, 57, 82–89.

[46] T. Hagerup, P. Sanders & J.L. Träff (1998) An implementation of the binary blocking flow algorithm. In K. Melhorn (ed.) *Proc. 2nd Workshop on Algorithm Engineering*, pp. 143–154. Max-Planck-Institut für Informatik, Saarbrücken.

[47] M. Iri (1960) A new method for solving transportation network problems. *J. Oper. Res. Soc. Jap.*, 3, 27–87.

[48] A. Jambulapati & A. Sidford (2021) Ultrasparse ultrasparsifiers and faster Laplacian system solvers. In *Proc. SODA '21*, pp. 540–559. Philadelphia, PA: SIAM.

[49] A.V. Karzanov (1973) On finding maximum flows in networks with special structure and some applications (in Russian). *Matematicheskie Voprosy Upravleniya Proizvodstvom*, 5, 81–94.

[50] A.V. Karzanov (1974) Determining the maximal flow in a network by the method of preflows. *Soviet Math. Doklady*, 15, 434–437.

[51] T. Kathuria, Y.P. Liu & A. Sidford (2020) Unit capacity maxflow in almost $O(m^{4/3})$ time. In *FOCS '20*, pp. 119–130. Piscataway, NJ: IEEE.

[52] V. King, S. Rao & R.E. Tarjan (1992) A faster deterministic maximum flow algorithm. In *Proc. SODA '92*, pp. 157–164. Philadelphia, PA: SIAM.

[53] V. King, S. Rao & R.E. Tarjan (1994) A faster deterministic maximum flow algorithm. *J. Algorithms*, 23, 447–474.

[54] M. Klein (1967) A primal method for minimal cost flows with applications to the assignment and transportation problems. *Manag. Sci.*, 14, 205–220.

[55] B.H. Korte & J. Vygen (2018) *Combinatorial Optimization* (6th edn). Heidelberg: Springer.

[56] P. Kovács (2015) Minimum-cost flow algorithms: an experimental evaluation. *Optim. Meth. Softw.*, 30, 94–127.

[57] Y.T. Lee & A. Sidford (2014) Path finding methods for linear programming: solving linear programs in $\tilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *Proc. FOCS '14*, pp. 424–433. Piscataway, NJ: IEEE.

[58] Y.P. Liu & A. Sidford (2020) Faster energy maximization for faster maximum flow. In *Proc. STOC '20*, pp. 803–814. New York: ACM.

[59] A. Madry (2016) Computing maximum flow with augmenting electrical flows. *Proc. FOCS '16*, pp. 593–602. Piscataway, NJ: IEEE.

[60] G.J. Minty (1960) Monotone networks. *Proc. Roy. Soc. Lon. A*, 257, 194–212.

[61] A. Orden (1956) The transhipment problem. *Manag. Sci.*, 2, 276–285.

[62] J.B. Orlin (1984) Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. *Working paper 1615-84*, Sloan School of Management, MIT.

[63] J.B. Orlin (1988) A faster strongly polynomial minimum cost flow algorithm. In *Proc. STOC '88*, pp. 377–387. New York: ACM.

[64] J.B. Orlin (1993) A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.*, 41, 338–350.

[65] J.B. Orlin (2013) Max flows in $O(nm)$ time, or better. In *Proc. STOC '13*, pp. 765–774. New York: ACM.

[66] J.B. Orlin & X.Y. Gong (2021) A fast maximum flow algorithm. Networks, 77, 287–321.

[67] S. Phillips & J. Westbrook (1993) Online load balancing and network flow. In *Proc. STOC '93*, pp. 402–411. New York: ACM.

[68] S. Phillips & J. Westbrook (1998) Online load balancing and network flow. *Algorithmica*, 21, 245–261.

[69] H. Röck (1980) Scaling techniques for minimal cost network flows. In U. Pape (ed.) *Discrete Structures and Algorithms: Proceedings of WG '79*, pp. 181–191. Munich: Hanser.

[70] A. Schrijver (2003) *Combinatorial Optimization: Polyhedra and Efficiency*. Berlin: Springer.

[71] A. Schrijver (2005) On the history of combinatorial optimization (till 1960). In K. Aardal, G.L. Nemhauser & R. Weismantel (eds) *Discrete Optimization*, pp. 1–68. Amsterdam: North Holland.

[72] Y. Shiloach (1978) *An $O(nI \log^2 I)$ maximum-flow algorithm.* Technical report, Computer Science Department, Stanford University.

[73] D.D. Sleator (1980) An $O(nm \log n)$ algorithm for maximum network flow. *Technical Report STAN-CS-80-831*, Department of Computer Science, Stanford University.

[74] D.D. Sleator & R.E. Tarjan (1983) A data structure for dynamic trees. *J. Comput. Sys. Sci.*, 26, 362–391.

[75] D.A. Spielman & S.H. Teng (2004) Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. STOC '04*, pp. 81–90. New York: ACM.

[76] É. Tardos (1985) A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5, 247–255.

[77] M. Thorup (2004) Integer priority queues with decrease key in constant time and the single source shortest paths problem. *J. Comput. Syst. Sci.*, 69, 330–353.

[78] N. Tomizawa (1971) On some techniques useful for solution of transportation network problems. *Networks*, 1, 173–194.

[79] D.P. Williamson (2019) *Network Flow Algorithms*, Cambridge, UK: Cambridge University Press.