Journal of the Operations Research
Society of Japan
Vol. 43, No. 1, March 2000

RECENT DEVELOPMENTS IN MAXIMUM FLOW ALGORITHMS

Takao Asano
Chuo University

Yasuhito Asano
The University of Tokyo

(Received January 7, 1999; Revised September 1, 1999)

Abstract Goldberg and Rao recently proposed the blocking flow method based on a binary length function to obtain a better algorithm for the maximum flow problem. The previous algorithms based on the blocking flow method proposed by Dinic use the unit length function: every residual edge is of length 1. In this paper, we survey properties of the distance function defined by a length function and give an overview on the representative maximum flow algorithms proposed so far in a systematic way by utilizing these properties. Among them are included two new algorithms: the Goldberg-Rao algorithm which finds a maximum flow on an integral capacity network N of n vertices and m edges in $O(\min\{m^{1/2}, n^{2/3}\}m\log(n^2/m)\log U)$ time, where U is the maximum edge capacity of N, and the Karger-Levine algorithm which finds a maximum flow on an undirected network N with unit capacity and no parallel edges in $O(m + n\nu^{3/2})$ time, where ν is the value of a maximum flow of N.

1. Introduction

The maximum flow problem, finding a flow of maximum value on a network from a source to a sink, is one of the most fundamental problems with a wide variety of scientific and engineering applications and has been studied intensively. The problem was formulated by Dantzig [14] and solved by Ford-Fulkerson [19] based on the augmenting path method. Since then, a number of algorithms have been proposed and representative algorithms are listed in Table 1. Nice survey papers and books on this topic have also been published [46, 13, 25, 1, 2, 31, 29, 6]. The algorithm of Ford-Fulkerson [19] assumes that input networks have integral or rational capacities and sometimes fails to correctly find a maximum flow or to halt for a network with irrational capacities. Sunaga-Iri [45] proposed a method to find a maximum flow and terminate even for a network with irrational capacities. Dinic [15] and Edmonds-Karp [17] independently showed that the Ford-Fulkerson algorithm runs in polynomial time (even for networks with irrational capacities) if flows are augmented along shortest augmenting paths. More specifically, Dinic introduced a shortest augmenting path network, called a level network here, and proposed a blocking flow algorithm to find a maximum flow on a network with n vertices and m edges in $O(mn^2)$ time. Karzanov [37] improved this bound to $O(n^3)$ by introducing the concept of a preflow and obtaining a blocking flow in a level network in $O(n^2)$ time. Sleator-Tarjan [44] proposed a dynamic tree data structure, a new data structure suitable for manipulating flows in the level network, and obtained an algorithm for finding a blocking flow in the level network in $O(m\log n)$ time. This lead to an $O(mn\log n)$ time algorithm which is considered to be a most efficient algorithm based on the level network.

On the other hand, Goldberg-Tarjan [26] proposed a new algorithm which was not based on the level network. Their algorithm was called the push-relabel method and uses a preflow introduced by Karzanov and a distance label for each vertex that is a lower bound on the

Table 1: Representative maximum flow algorithms [23]. Years are based on the original publications while citations are based on the most complete publications.

year	authors	complexity
1955	Ford-Fulkerson [19]	$O(mnU)$
1970	Dinic [15]	$O(mn^2)$
1970	Edmonds-Karp [17]	$O(m^2n)$
1972	Dinic [15], Edmonds-Karp [17]	$O(m^2 \log U)$
1973	Dinic [16], Gabow [20]	$O(mn \log U)$
1974	Karzanov [37]	$O(n^3)$
1977	Cherkassky [11]	$O(n^2m^{1/2})$
1980	Galil-Naamad [21]	$O(mn(\log n)^2)$
1983	Sleator-Tarjan [44]	$O(mn \log n)$
1986	Goldberg-Tarjan [26]	$O(mn \log(n^2/m))$
1987	Ahuja-Orlin [3]	$O(mn + n^2 \log U)$
1987	Ahuja-Orlin-Tarjan [4]	$O(mn \log(2 + n\sqrt{\log U}/m))$
1990	Cheriyan-Hagerup-Mehlhorn [9]	$O(n^3/\log n)$
1990	Alon [5]	$O(mn + n^{8/3} \log n)$
1992	King-Rao-Tarjan [38]	$O(mn + n^{2+\epsilon})$
1993	Phillips-Westbrook [42]	$O(mn \log_{m/n} n + n^2(\log n)^{2+\epsilon})$
1994	King-Rao-Tarjan [39]	$O(mn \log_{m/(n \log n)} n)$
1997	Goldberg-Rao [23]	$O(\min\{m^{1/2}, n^{2/3}\}m \log(n^2/m) \log U)$

length of a shortest path to the sink. They obtained $O(n^3)$ and $O(mn \log(n^2/m))$ time algorithms based on queues and dynamic trees respectively. The push-relabel method had a number of flexibilities and many variants have been proposed. Cheriyan-Hagerup [8] proposed a randomized algorithm based on a combinatorial game. Their algorithm always finds a maximum flow in $O(mn+n^2(\log n)^2)$ time with probability at least $1-2^{-\sqrt{nm}}$ and in $O(mn \log n)$ time in the worst case. Alon [5] proposed a derandomization of the Cheriyan-Hagerup algorithm and obtained an $O(mn + n^{8/3} \log n)$ time algorithm. King-Rao-Tarjan [38] also considered a slightly different combinatorial game and obtained an $O(mn + n^{2+\epsilon})$ time algorithm for any positive constant ϵ. After that, Phillips-Westbrook [42] obtained an $O(mn \log_{m/n} n + n^2(\log n)^{2+\epsilon})$ time algorithm and King-Rao-Tarjan [39] obtained an $O(mn \log_{m/(n \log n)} n)$ time algorithm (this assumes $m > n \log n$ and if $m < n \log n$ then the time complexity of this algorithm becomes $O(mn \log(n^2/m))$). This slightly improved the Phillips-Westbrook algorithm (for $m = n \log n \log \log n$, the King-Rao-Tarjan algorithm runs a factor of $\Omega((\log n)^\epsilon)$ faster than the Phillips-Westbrook algorithm).

The shortest augmenting path method, the blocking flow method on the level network, and the push-relabel method described above used a concept of distance based on the unit length function: the length of an edge in a residual network was defined to be one. Goldberg-Rao [23] used a binary length function and obtained an efficient algorithm for networks with integral capacities. In practical applications, capacities of a network are often represented by approximate integers or rational numbers. Thus, integral capacity constraints make almost no restriction on practical problems. For the maximum flow problem on networks with integral capacities, we assume that the maximum edge capacity of a network is denoted by U and every edge capacity is an integer in the range $[0, U] = \{0, 1, ..., U\}$. To compare with polynomial and strongly polynomial time algorithms, the similarity assump-

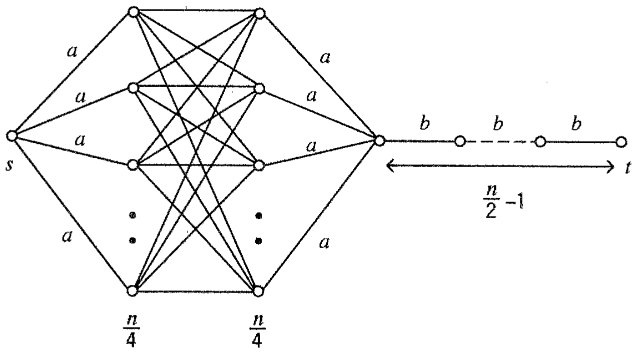

Figure 1: Example of a network N with total path length $\Omega(mn)$, where edges are directed from left to right, $cap(e)$ is shown near edge e ($cap(e)=1$ is omitted) and $a=n$ and $b=n^2$.

tion $\log U = O(\log n)$ introduced by Gabow [20] is often used. The ballpark bound denoted by O^* under the similarity assumption is also used ($O^*(f(n)) = O((\log n)^{O(1)} f(n))$). The bound $\Omega(mn)$ is a natural barrier for maximum flow algorithms and all algorithms described above are dominated by this bound. In a path decomposition of a flow, the total path length is $\Theta(mn)$ in the worst case (Figure 1). This implies an $\Omega(mn)$ lower bound on any algorithm which augments a flow along an augmenting path. This lower bound does not apply to algorithms that work preflows or use data structures like dynamic trees. In spite of numerous attempts, however, no algorithm described above achieves this lower bound in general. For dense graphs, Cheriyan-Hagerup-Mehlhorn [9] achieved the bound $O(n^3/\log n)$ beating the lower bound $\Omega(mn)$. On the other hand, Goldberg-Rao [23] proposed in 1997 an $O(\min\{m^{1/2}, n^{2/3}\} m \log(n^2/m) \log U)$ time algorithm beating drastically this lower bound under the assumption of similarity.

For networks with unit edge capacity, the total path decomposition length is $O(m)$ and $o(mn)$ bounds have been obtained by Karzanov [36] and Even-Tarjan [18], independently. Actually, Even-Tarjan have shown that the Dinic algorithm runs in $O(\min\{m^{1/2}, n^{2/3}\}m)$ time on networks with unit capacity and no parallel edges. The Goldberg-Rao algorithm [23] achieves this bound (the ballpark bound) for general networks. On the other hand, Goldberg-Rao [24] obtained an $O(\min\{m, n^{3/2}\} m^{1/2})$ time algorithm on undirected networks with unit capacities and no parallel edges. Karger-Levine [35] proposed an $O(m + n\nu^{3/2})$ time algorithm, where ν is the value of a maximum flow on an undirected network with unit capacity and no parallel edges. They also proposed an algorithm with $O(nm^{2/3}\nu^{1/6})$ time and a randomized algorithm with $O^*(m + n^{11/9}\nu)$ time. The latter algorithm suggests that the maximum flow problem of undirected networks with unit capacity seems easier than the maximum bipartite matching problem, since an $O(n^{2.5})$ time algorithm [28] has long been fastest for the bipartite matching problem in spite of many efforts for nearly thirty years.

This survey paper is organized as follows. We first give an overview of representative methods in maximum flow algorithms including the shortest augmenting path method, the blocking flow method on the level networks and the push-relabel method in Section 2. We also give fundamental properties of distance labelings defined by length functions and review the Dinic, Even-Tarjan, and Goldberg-Tarjan algorithms in a systematic way based on the distance labeling. Then in Section 3, we consider algorithms for integral capacity networks where scaling of edge capacities is widely used. We see how scaling techniques have been used in polynomial time algorithms. The central part of this survey is a description of the Goldberg-Rao algorithm where we try to give several comments on their original algorithm and present an illustrative example. In Section 4, we consider algorithms for undirected unit capacity networks. Sparsification of a network proposed by Nagamochi-Ibaraki [41] is a most powerful tool in the algorithms recently proposed by Goldberg-Rao [24] and Karger-Levine [35]. We give a brief overview of these two algorithms based on the sparsification and the properties of the distance labelings. In Section 5 we give concluding remarks.

2. Notation and Fundamental Algorithms

A directed graph $G = (V, E)$ having a nonnegative real-valued capacity $cap(e)$ on each edge $e \in E$ and two distinct distinguished vertices, a *source* s and a *sink* t, is called a *network* and denoted by $N = (G, cap, s, t)$. Throughout this paper, $n = |V|$ and $m = |E|$. We also use U to denote the maximum edge capacity if all edges have integral capacities. Let $\delta^+(v) = \{e = (v, w) \in E\}$ denote the set of edges in E out of v. Similarly, $\delta^-(v) = \{e = (u, v) \in E\}$ denotes the set of edges in E into v. A *flow* f on a network $N = (G, cap, s, t)$ is a real-valued function f on edge set E satisfying the following constraints:

$$0 \leq f(e) \leq cap(e) \text{ for all } e \in E \quad \text{(capacity constraint);} \tag{2.1}$$

$$\sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)} f(e) \text{ for all } v \in V - \{s, t\} \quad \text{(conservation constraint).} \tag{2.2}$$

The *value* of a flow f, denoted by $val(f)$, is the net flow out of source s:

$$val(f) = \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e).$$

By conservation constraint, $val(f)$ is equal to the net flow into sink t. A *maximum flow* is a flow of maximum value. The *maximum flow problem* is the problem of finding a maximum flow on a given network. We can assume that all edge capacities are finite, since if some edge capacities are infinite but no path consisting of infinite-capacity edges from s to t exists, then each infinite capacity can be replaced by the sum of the finite capacities without affecting the problem.

For a subset X of V with $s \in X$ and $t \in V - X$, let $\delta^+(X)$ be the set of edges out of X (to $V - X$). Similarly, $\delta^-(X)$ is the set of edges out of $V - X$ (to X). $\delta^+(X)$ is called an *s-t cut* and its *capacity* is defined by $cap(X) = \sum_{e \in \delta^+(X)} cap(e)$. Note that $\sum_{e \in \delta^+(X)} f(e) - \sum_{e \in \delta^-(X)} f(e) = val(f)$ and the following famous maximum flow minimum cut theorem holds.

Theorem 2.1 *For any flow f and any s-t cut $\delta^+(X)$ of a network $N = (G, cap, s, t)$, an inequality $val(f) \leq cap(X)$ holds. Furthermore, for a maximum flow f^* and an s-t cut $\delta^+(X^*)$ of minimum capacity, $val(f^*) = cap(X^*)$ holds.*

2.1. Residual networks and augmenting paths

The Ford-Fulkerson algorithm uses a residual network. For an edge $e = (v, w) \in E$, let $e^R = (w, v)$ be an edge reversing the direction of e. If $e = (v, w) \in E$ and $(w, v) \notin E$, then we add the edge $e^R = (w, v)$ with $cap(e^R) = 0$. If edges $e = (v, w)$ and $e' = (w, v)$ are both in E then we consider $e^R = e'$ and $e = e'^R$ ($cap(e^R) = cap(e')$ and $cap(e'^R) = cap(e)$). Thus, for simplicity, we assume throughout this paper that G is simple and symmetric (i.e., $(v, w) \in E$ if and only if $(w, v) \in E$). This implies $E = E \cup E^R$ ($E^R \equiv \{e^R \mid e \in E\}$). For a flow f on a network $N = (G, cap, s, t)$, define

$$cap_f(e) = cap(e) - f(e) + f(e^R) \quad (cap_f(e^R) = cap(e^R) - f(e^R) + f(e)). \tag{2.3}$$

Then $cap_f(e)$ is called a *residual capacity* of $e = (v, w)$, since we can increase $f(e)$ by $cap(e) - f(e)$ along $e = (v, w)$ and decrease $f(e^R)$ along $e^R = (w, v)$ (and thus we can increase flow from v to w by $cap_f(e)$ in total). The *residual network* $N(f) = (G(f) = (V, E(f)), cap_f, s, t)$ *with respect to* f is defined to be

$$E(f) = \{e \in E \mid cap_f(e) > 0\}. \tag{2.4}$$

Furthermore, if $f(e) > 0$ and $f(e^R) > 0$ for some $e \in E$, then we can easily modify f without changing the flow value $val(f)$ and the residual capacities as follows:

$$g(e) := \min\{f(e), f(e^R)\}; \quad f(e) := f(e) - g(e); \quad f(e^R) := f(e^R) - g(e). \tag{2.5}$$

Thus, for simplicity, we assume that, whenever $f(e)$ is modified, we always perform (2.5) for $e, e^R \in E$ immediately after that and, thus, a flow f satisfies the following constraint throughout the paper:

$$f(e) = 0 \quad \text{or} \quad f(e^R) = 0 \quad \text{for all } e \in E. \tag{2.6}$$

Note that $E(f) = E - \{e \in E \mid cap_f(e) = 0\}$ and if $cap(e) = f(e) > 0$ then $cap_f(e) = 0$ since $f(e^R) = 0$ by (2.6). A path $P = P(s, t)$ in the residual network $N(f)$ from s to t is called an *augmenting path with respect to* f. Actually, define the *residual path capacity* of $P = P(s, t)$, denoted by $\Delta(P)$, to be the minimum value among the residual capacities of edges in P (thus $\Delta(P) > 0$) and set

$$f' := f + \Delta(P). \tag{2.7}$$

Here, by $f' := f + \Delta(P)$, we mean that we first set

$$f'(e) := \begin{cases} f(e) + \Delta(P) & (e \in P) \\ f(e) & (e \notin P) \end{cases}$$

and then perform (2.5) for each $e \in P$ (by substituting f' for f). Thus, the obtained f' satisfies flow constraints and (2.6) and f' is a flow on N with $val(f') = val(f) + \Delta(P) > val(f)$. Note that, before performing (2.5), some $f'(e)$ may be larger than $cap(e)$, and that, after performing (2.5), some $f'(e')$ with $e' \in P$ may be zero. We have, however, $cap_{f'}(e) = cap_f(e) - \Delta(P)$ and $cap_{f'}(e^R) = cap_f(e^R) + \Delta(P)$ for each $e \in P$ unless P contains both e and e^R, since the residual capacity of edge $e \in P$ is not changed by performing (2.5). (If P contains both e and e^R, then $cap_{f'}(e) = cap_f(e)$ and $cap_{f'}(e^R) = cap_f(e^R)$.) Thus, we can increase the value of a flow by sending a flow along an augmenting path. This implies that, if f is a maximum flow on N then there is no augmenting path with respect to f. The converse is also true and the following theorem holds.

Theorem 2.2 *For any flow f on $N = (G, cap, s, t)$, f is a maximum flow if and only if there is no augmenting path with respect to f.*

Proof. Since the necessity is already described above, we consider the sufficiency. Suppose that there is no path from s to t in the residual network $N(f)$. Let X be the vertex set reachable from s in $N(f)$. Then $s \in X$ and $t \in V - X$ and $\delta^+(X)$ is an s-t cut of N. By the definitions of X and $N(f)$, each edge $e \in \delta^+(X)$ of N satisfies $f(e) = cap(e)$, $f(e^R) = 0$ $(cap_f(e) = 0)$ and each edge $e \in \delta^-(X)$ of N satisfies $f(e) = 0$, $f(e^R) = cap(e^R)$ $(cap_f(e^R) = 0)$. Thus,

$$val(f) = \sum_{e \in \delta^+(X)} f(e) - \sum_{e \in \delta^-(X)} f(e) = \sum_{e \in \delta^+(X)} cap(e) = cap(X).$$

Since, for any flow f' and any s-t cut $\delta^+(X')$ of N, an inequality $val(f') \leq cap(X')$ holds, we have $val(f) = cap(X) \geq val(f')$ and $cap(X) = val(f) \leq cap(X')$. Thus, f is a maximum flow and $\delta^+(X)$ is an s-t cut of minimum capacity in N. □

Based on the above observation, Ford-Fulkerson obtained the following algorithm.

Ford-Fulkerson Algorithm

1. Set $f := 0$ (i.e., $f(e) := 0$ for each edge $e \in E$).
2. Repeat finding an augmenting path P_f with respect to f and augmenting the flow by $f := f + \Delta(P_f)$ until there is no augmenting path P_f with respect to f.

If all edge capacities are integers, then one can easily obtain by induction that, at any time of the iterations, the edge capacities of $N(f)$ and $\Delta(P_f)$ are integers and a flow f is integral (i.e., $f(e)$ is an integer for each $e \in E$). Thus, we have the integrality theorem as follows.

Theorem 2.3 *If all edge capacities are integers, then there is a maximum flow f such that $f(e)$ is an integer for every $e \in E$.*

Since any flow f'' on the residual network $N(f)$ can be decomposed into a set of paths $\mathcal{P}$ with $f''(e) = \sum_{P \in \mathcal{P}: P \ni e} \Delta(P)$, we can use the notation

$$f' := f + f'' \tag{2.8}$$

extending (2.7) to augment f to f' using f''. Furthermore, if f'' is a maximum flow on $N(f)$, then $f' := f + f''$ is a maximum flow of N. Such a maximum flow f'' is called a *maximum residual flow* of f.

2.2. Blocking flows and level networks

As mentioned in Introduction, the Ford-Fulkerson algorithm may have many augmentations if the network has large integral capacities and it sometimes fails to correctly find a maximum flow or to halt if the network has irrational capacities. Thus, we have to select augmenting paths carefully so that their method becomes efficient. Dinic and Edmonds-Karp independently showed that the Ford-Fulkerson algorithm runs in polynomial time (even for networks with irrational capacities) if augmentations are done along shortest augmenting paths. In this section, we give an overview of the Dinic algorithm.

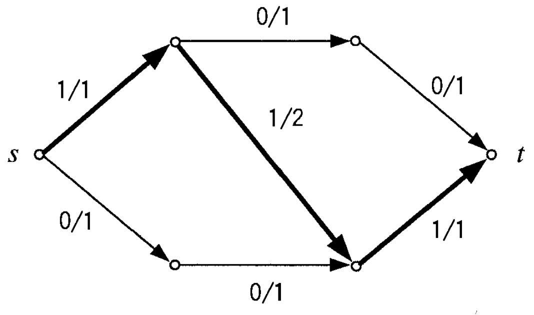

Figure 2: A blocking flow f'' of value 1 on a level network $N_L(f)$ $(f''(e)/cap_f(e))$

Dinic used a blocking flow and a level network (Figure 2). A flow f'' is a *blocking flow* of the residual network $N(f) = (G(f) = (V, E(f)), cap_f, s, t)$ with respect to a flow f on $N = (G, cap, s, t)$ if any path from s to t has an edge e with $f''(e) = cap_f(e) > 0$, that is, there is no path from s to t in the network obtained from $N(f)$ by deleting all edges e with

$f''(e) = cap_f(e) > 0$. A *level network* $N_L(f) = (G_L(f) = (V, E_L(f)), cap_f, s, t)$ of $N(f)$ is the network obtained from $N(f)$ by choosing all the edges in shortest paths in $N(f)$ from s to t (a shortest path from s to t here is a path from s to t containing the minimum number of edges) and

$$E_L(f) \subseteq \{(u,v) \in E(f) \mid level[u] = level[v] + 1\}$$

where $level[v]$ is the length (i.e., the number of edges) of a shortest path in $N(f)$ from a vertex v to t. The levels $level[v]$ of all vertices v can be computed in $O(m)$ time by the breadth-first search for $N(f)$. Now the Dinic algorithm can be written as follows.

Dinic Algorithm

1. Set $f := 0$ (i.e., $f(e) := 0$ for each edge $e \in E$).
2. Repeat finding a blocking flow f'' on the level network $N_L(f)$ of the residual network $N(f)$ and augmenting $f := f + f''$ until there is no path from s to t in $N(f)$.

Let $level_k[s]$ denote $level[s]$ in the k-th iteration of finding a blocking flow. Then it can be shown that $level_{k+1}[s] > level_k[s]$ holds due to blocking flows (see Lemma 2.7 later). Thus there are at most $n-1$ iterations. In each iteration, a blocking flow can be computed in $O(mn)$ time by the depth-first search for the level network $N_L(f)$. Thus, the following theorem is obtained.

Theorem 2.4 *The Dinic algorithm finds a maximum flow f on N in $O(mn^2)$ time.*

2.3. Length functions and distance labelings

Let $N(f) = (G(f), cap_f, s, t)$ be the residual network with respect to f. A *length function* ℓ is a function on $E(f)$ to nonnegative numbers. For a labeling function d on V with $d(t) = 0$ and a length function ℓ on $E(f)$, a *reduced length* of edge $e = (u,v) \in E(f)$ is defined by $\ell_d(e) = \ell(e) + d(v) - d(u)$. The labeling function d is called a *distance labeling* if the reduced lengths of all edges are nonnegative. Such a distance labeling always exists here since there is no negative cycle in $N(f)$ with respect to the length function ℓ. Let $d_\ell(v)$ be the length of a shortest path from v to t in $N(f)$ with the length function ℓ. Then d_ℓ is a distance labeling and the following inequality holds for any distance labeling d of $N(f)$:

$$d \leq d_\ell \quad \text{(i.e., } d(v) \leq d_\ell(v) \text{ for any } v \in V\text{)}.$$

Thus, d_ℓ can be called the *maximum distance labeling.* Note that, by the linear programming duality, the shortest path problem can be formulated as the dual problem for the problem of maximizing d with the constraint that d is a distance labeling.

For a flow f and a length function ℓ of $N(f)$, an edge $e = (u,v) \in E(f)$ satisfying $d_\ell(u) = d_\ell(v) + \ell(e)$ is called *admissible* in $N(f)$ and the *shortest residual network* $N_\ell(f)$ is the network obtained from $N(f)$ by deleting the edges not contained in any shortest path from s to t. That is, the edge set $E_\ell(f)$ of $N_\ell(f)$ is a subset of admissible edges of $N(f)$ and defined as follows.

$$\begin{aligned} E_\ell(f) &= \{e \in E(f) \mid e \text{ is contained in a shortest path from } s \text{ to } t \text{ in } N(f)\} \\ &\subseteq \{e = (u,v) \in E(f) \mid d_\ell(u) = d_\ell(v) + \ell(e)\}. \end{aligned} \tag{2.9}$$

Of course, the capacity of $e \in E_\ell(f) = E_\ell(f) \cap E(f)$ is kept the same.

Let P be a path in $N_\ell(f)$ from s to t and $0 < \Delta' \leq \Delta(P)$ ($\Delta(P)$ is the residual path capacity of P). We try to augment the current flow f by pushing a flow of value Δ' along P and obtain a flow $f' := f + \Delta'$ (by extending (2.7), $f' := f + \Delta'$ means that we first set

$$f'(e) := \begin{cases} f(e) + \Delta' & (e \in P) \\ f(e) & (e \notin P)) \end{cases}$$

and then perform (2.5) for each $e \in P$ (by substituting f' for f)). We also update the length function ℓ to ℓ' using a nonnegative function ℓ'' as follows:

$$\ell'(a) := \begin{cases} \ell(a) + \ell''(a) & (cap_{f'}(a) < cap_f(a)) \\ \ell''(a) & (cap_{f'}(a) > cap_f(a)) \\ \ell(a) & (cap_{f'}(a) = cap_f(a)) \end{cases} \tag{2.10}$$

Note that it is possible that $cap_{f'}(a) \neq cap_f(a)$ in spite of $f'(a) = f(a)$ since $cap_{f'}(a) = cap(a) - f'(a) + f'(a^R)$ and $cap_f(a) = cap(a) - f(a) + f(a^R)$. If $cap_{f'}(a) = 0 < cap_f(a)$, then a is not in $N(f')$ and we assume that $\ell''(a) = \infty$ holds implicitly. Then we have the following lemma.

Lemma 2.5 *d_ℓ is a distance labeling of $N(f')$ with respect to ℓ' defined by (2.10).*

Proof. The lengths are changed according to the residual capacities and the residual capacities are changed only for the edges a and a^R with a in P. Therefore, it suffices to show that d_ℓ satisfies the requirement of a distance labeling with respect to ℓ' only for such edges.

If $cap_{f'}(a) < cap_f(a)$ for an edge $a = (u, v)$, then $a \in P \cap E_\ell(f)$ and we have $d_\ell(v) + \ell'(a) \geq d_\ell(v) + \ell(a) = d_\ell(u)$ by $\ell'(a) - \ell(a) = \ell''(a) \geq 0$. If $cap_{f'}(a) > cap_f(a)$ for an edge $a = (u, v)$, then $a^R = (v, u) \in P \cap E_\ell(f)$ and $d_\ell(v) = \ell(a^R) + d_\ell(u) \geq d_\ell(u)$, and thus $d_\ell(v) + \ell'(a) \geq d_\ell(v) \geq d_\ell(u)$. □

Note that P can contain a cycle of length 0 although it does not go through the same edge twice or more. Thus, P may not be simple. By Lemma 2.5, the maximum distance labeling $d_{\ell'}$ with respect ℓ' satisfies $d_{\ell'} \geq d_\ell$. We can generalize Lemma 2.5 as follows. If f'' is a flow on the residual shortest network $N_\ell(f)$ of $N(f)$, then f'' can be decomposed into a set of flows along shortest paths in $N_\ell(f)$. Considering each such a flow in the decomposition independently and simultaneously, we have the following corollary.

Corollary 2.6 *If f'' is a flow on the residual shortest network $N_\ell(f)$ of $N(f)$, then, for $f' := f + f''$, d_ℓ is a distance labeling of $N(f')$ with respect to ℓ' defined in (2.10) and thus $d_{\ell'} \geq d_\ell$.*

Note that, in the Dinic algorithm, each edge in $N(f)$ has unit length and the level network exactly corresponds to the residual shortest network. The $level_k[s]$ in the k-th iteration can be shown to satisfy $level_{k+1}[s] > level_k[s]$ as a special case of the following lemma.

Lemma 2.7 *If f'' is a blocking flow on the residual shortest network $N_\ell(f)$ of $N(f)$ and all edges lengths in ℓ of $N(f)$ and in ℓ' of $N(f')$ with $f' := f + f''$ are positive, then $d_{\ell'}(s) > d_\ell(s)$ (ℓ' is defined in (2.10)).*

Proof. Let $N_\ell^+(f)$ be the network obtained from $N_\ell(f)$ by adding all edges e^R with length 0 for edges e in $N_\ell(f)$. Note that $d_\ell(u) = d_\ell(v) + \ell(e) > d_\ell(v)$ for an edge $e = (u, v)$ of $N_\ell(f)$ and thus $e^R = (v, u)$ is a kind of backward edge directed from a vertex of less distance to a vertex of larger distance in $N_\ell(f)$ and thus e^R is not in $N_\ell(f)$. Furthermore, if $e^R = (v, u)$ is of length more than $d_\ell(v) - d_\ell(u)$ ($d_\ell(v) - d_\ell(u) < 0$), then its addition to $N_\ell(f)$ makes no effect on the shortest paths in $N_\ell(f)$ from s to t.

Let $N_\ell^-(f)$ ($N_\ell^\pm(f)$, resp.) be the network obtained from $N_\ell(f)$ ($N_\ell^+(f)$, resp.) by deleting all edges a with $cap_f(a) = f''(a)$. Then there is no path in $N_\ell(f)$ (and in $N_\ell^+(f)$ of length at most $d_\ell(s)$) from s to t, since f'' is a blocking flow of $N_\ell(f)$. Let $N^\pm(f)$ be the network obtained from $N_\ell^\pm(f)$ by adding all edges e in $N(f)$ but not in $N_\ell^\pm(f)$. Then, any path from s to t in the network $N^\pm(f)$ is of length greater than $d_\ell(s)$, since all edges a with $cap_f(a) = f''(a)$ are deleted.

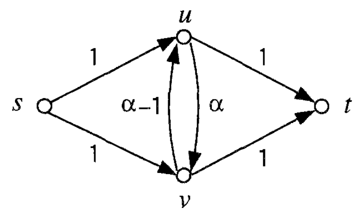

Figure 3: For threshold α, only edge (u,v) is of length 0 and all other edges are of length 1. Thus, $d_\ell(s)=2$. If we augment the current flow by using a blocking flow along a shortest path s,u,v,t of length 2 and increase the flow value by 1, then, in the residual network in the next time, the edge (v,u) satisfies $\ell'(v,u)=0$ and thus $d_{\ell'}(s)=2$.

Note that the edge set of $N(f')$ is a subset of the edge set of $N^{\pm}(f)$ and the length ℓ' is at least the length of $N^{\pm}(f)$ and thus, any path from s to t in the network $N(f')$ is of length greater than $d_\ell(s)$. □

By this lemma, we can estimate the number of iterations of finding a blocking flow in a maximum flow algorithm. If we use the unit length function ℓ of $N(f)$ as in the Dinic algorithm, then the level network $N_L(f)$ of $N(f)$ exactly corresponds to the residual shortest network $N_\ell(f)$ of $N(f)$ and the number of iterations is $O(n)$. Goldberg-Rao introduced a concept of volume which can also be used to estimate the number of iterations. Consider an edge e in $N(f)$ as a pipe and the residual capacity $cap_f(e)$ of e as an area of the cross section of pipe e. Then $cap_f(e)\ell(e)$ becomes the *volume* of pipe e and the total volume $Vol_{f,\ell}$ of the residual network $N(f)$ is

$$Vol_{f,\ell}=\sum_{e\in E(f)} cap_f(e)\ell(e).$$

The difference of the value of a maximum flow f^* and that of a current flow f, denoted by $val(f^*)-val(f)$, can be estimated as follows by using this volume, since if we augment f by a flow of value 1 along an augmenting path in $N(f)$ from s to t then the volume decreases by at least $d_\ell(s)$ (note that $Vol_{f,\ell}$ is nonnegative by the definition).

Lemma 2.8 *[23] For a maximum flow f^* and a current flow f on N and a length function ℓ of $N(f)$,*

$$val(f^*)-val(f)\le\frac{Vol_{f,\ell}}{d_\ell(s)}.$$

By Lemmas 2.7 and 2.8, to decrease the number of iterations, the number of different values which $d_\ell(s)$ can take on and the value $Vol_{f,\ell}$ should be made small. This leads us to the following strategy: edges with large residual capacity should be of shorter length. More specifically, all edges with large residual capacities should be of length 0. However, if we allow a general length function, then analysis becomes harder. From these kinds of viewpoints, a *binary length function* ℓ with $\ell(a)=0$ or $\ell(a)=1$ for each a may be of help. Goldberg-Rao [23] considered a binary length function ℓ such that $\ell(a)=0$ if $cap_f(a)\ge\alpha$ and $\ell(a)=1$ if $0<cap_f(a)<\alpha$ for some threshold α. However, the existence of edges of length 0 makes Lemma 2.7 violated in some cases (Figure 3). In spite of these weak points, a binary length function ℓ has nice aspects described below.

Using a maximum distance labeling d_ℓ, let $S_k=\{v\in V \mid d_\ell(v)\ge k\}$ $(k=1,2,...,d_\ell(s))$ and define $d_\ell(s)$ *s-t* cuts $\delta^+(S_k)$ of $N(f)$ to be *canonical cuts* of $N(f)$. Canonical cuts can be rewritten by $\delta^+(S_k) = E(V_k, V_{k-1})$ if we use $V_k \equiv \{v \in V \mid d_\ell(v) = k\}$ and $E(V_k, V_{k-1}) \equiv \{a = (u,v) \in E(f) \mid d_\ell(u) = k,\ d_\ell(v) = k-1\}$. Let S be a set in $\mathcal{S} = \{S_1, S_2, ..., S_{d_\ell(s)}\}$ such that $cap_f(S) = \min_{S_k \in \mathcal{S}}\{cap_f(S_k)\}$. That is, $\delta^+(S)$ is a canonical cut of minimum capacity. Then the value $val(f'') = val(f^*) - val(f)$ of a maximum residual flow f'' on $N(f)$ $(f^* := f + f'')$ can be bounded by $cap_f(S)$ and the following lemma is obtained.

Lemma 2.9 *[23] Let ℓ be any binary length function of $N(f)$ and $\delta^+(S)$ be a canonical cut of $N(f)$ of minimum capacity. Then*

$$val(f'') = val(f^*) - val(f) \le cap_f(S) \le \beta\left(\frac{n}{d_\ell(s)}\right)^2,$$

where f'' is a maximum residual flow on $N(f)$ $(f^* := f + f'')$ and β is the maximum residual capacity of edges of $N(f)$ of length 1.

Proof. Consider $V_{2k+1} \cup V_{2k}$ $(k = 0, 1, ..., \lceil \frac{d_\ell(s)}{2} \rceil - 1)$. If all such $\lceil \frac{d_\ell(s)}{2} \rceil$ sets had more than $\frac{2n}{d_\ell(s)}$ vertices, then the network N would have more than n vertices and a contradiction is obtained. Thus, there is a set $V_{2k+1} \cup V_{2k}$ with at most $\frac{2n}{d_\ell(s)}$ vertices. Then we have $cap_f(S) \le \beta |E(V_{2k+1}, V_{2k})| \le \beta \left(\frac{n}{d_\ell(s)}\right)^2$ since N is simple and $|E(V_{2k+1}, V_{2k})| \le |V_{2k+1}||V_{2k}| \le \left(\frac{n}{d_\ell(s)}\right)^2$. □

By Lemmas 2.7, 2.8 and 2.9, we have the following lemma and theorem for unit capacity networks.

Lemma 2.10 *For a flow f in a unit capacity network N, let E_f be the set of edges a with $f(a) = 1$. Then there is a flow f of value r such that $|E_f| \le 2n\sqrt{r}$.*

Proof. Using the Ford-Fulkerson algorithm, we initially set $f' := 0$ and repeat augmenting f' along a shortest path in $N(f')$ from s to t (with the unit length function ℓ) and increase the flow value by 1 until the value of flow f' becomes r. Let f be the flow f' of value r obtained in this way. We rewrite N to be the unit capacity network with edge set E_f. Then N contains no directed cycle since we used shortest augmenting paths. Now try again to find the flow f of value r in N by the same method as above. Then, for a flow f' of value r' in $(r'+1)$-st iteration, $f - f'$ is a maximum residual flow in the residual network $N(f')$ and $d_\ell(s)$ satisfies $d_\ell(s) \le \frac{n}{\sqrt{r-r'}}$ by Lemma 2.9 and $\beta = 1$. Thus, in $(r'+1)$-st iteration, the flow value is increased from r' to $r'+1$ and at most $\frac{n}{\sqrt{r-r'}}$ edges are newly used to augment flow f'. This implies

$$|E_f| \le \sum_{r'=0}^{r-1} \frac{n}{\sqrt{r-r'}} = \sum_{k=1}^{r} \frac{n}{\sqrt{k}} \le n\left(1 + \int_1^r \frac{1}{\sqrt{x}}dx\right) \le 2n\sqrt{r}.$$

□

Theorem 2.11 *[36, 18] The Dinic algorithm finds a maximum flow on a unit capacity network N with no parallel edge in $O(\min\{m^{1/2}, n^{2/3}\}m)$ time.*

Proof. Since the residual capacities are 1 or 2 in the residual network and each blocking flow can be obtained in $O(m)$ time, we can assume that there are at least $\min\{m^{1/2}, n^{2/3}\}$ iterations of finding a blocking flow (otherwise the theorem trivially holds). Thus, the maximum flow f^* is of value at least $\min\{m^{1/2}, n^{2/3}\}$ and $val(f^*) \le n \le m$ since N has no parallel edge. We will show below that there are at most $3\min\{m^{1/2}, n^{2/3}\}$ iterations of finding a blocking flow. Let ℓ be the unit length function.

We first consider the case when $m^{1/2} \leq n^{2/3}$. After the $m^{1/2}$-th blocking flow iteration, the flow f is of value at least $m^{1/2}$ and $d_\ell(s) \geq m^{1/2}$ by Lemma 2.7. By Lemma 2.8, we have

$$val(f^*) - val(f) \leq \frac{Vol_{f,\ell}}{d_\ell(s)} \leq \frac{2m}{m^{1/2}} = 2m^{1/2}$$

and there are at most $2m^{1/2}$ blocking flow iterations after the $m^{1/2}$-th blocking flow iteration.

Next we consider the case when $m^{1/2} > n^{2/3}$. After the $n^{2/3}$-th blocking flow iteration, the flow f is of value at least $n^{2/3}$ and $d_\ell(s) \geq n^{2/3}$ by Lemma 2.7. By Lemma 2.9, we have

$$val(f^*) - val(f) \leq 2\left(\frac{n}{d_\ell(s)}\right)^2 \leq 2\left(\frac{n}{n^{2/3}}\right)^2 = 2n^{2/3}$$

and there are at most $2n^{2/3}$ blocking flow iterations after the $n^{2/3}$-th blocking flow iteration.

□

2.4. Preflows and push-relabel method

A *preflow* f on a network $N = (G = (V, E), cap, s, t)$ is a real-valued function f on edge set E satisfying the following constraints [37]:

$$0 \leq f(e) \leq cap(e) \text{ for all } e \in E \quad \text{(capacity constraint);} \tag{2.11}$$

$$\sum_{e\in\delta^-(v)} f(e) - \sum_{e\in\delta^+(v)} f(e) \geq 0 \text{ for all } v \in V - \{s\} \quad \text{(preflow constraint).} \tag{2.12}$$

The value $ex(v) \equiv \sum_{e\in\delta^-(v)} f(e) - \sum_{e\in\delta^+(v)} f(e)$ is called an *excess* of v and a vertex $v \in V - \{s,t\}$ is called *active* if $ex(v) > 0$. Clearly $ex(s) \leq 0$ by the preflow constraint. Note that the only difference between flows and preflows is $ex(v) = 0$ in a flow but $ex(v) \geq 0$ in a preflow for each vertex $v \in V - \{s,t\}$. Thus, we use the same terms, notation and requirements (except preflow constraint) for preflows as for flow introduced before in this paper. An edge $e \in E$ is *saturated* if $cap_f(e) = 0$ and *unsaturated* otherwise. (The capacity constraint implies that any unsaturated edge e has $cap_f(e) > 0$ and the residual network $N(f)$ of preflow f consists of the unsaturated edges.)

We describe the Goldberg-Tarjan push-relabel method based on preflows [26] by borrowing the summary given by Ahuja-Orlin-Tarjan [4]. The preflow algorithm maintains a preflow f and moves flow from active vertices through edges in $N(f)$ toward the sink t, along estimated shortest paths. Excess flow that cannot be moved to the sink t is returned to the source s, also along estimated shortest paths. Eventually the preflow becomes a maximum flow.

As an estimate of path lengths, the algorithm uses the unit length function ℓ and a distance labeling d of $N(f)$ such that $d(s) = n$, $d(t) = 0$ and $d(v) \leq d(w) + \ell(e)$ for every edge $e = (v, w)$ of $N(f)$ ($\ell(e) = 1$). (Recall that, for a labeling function d on V with $d(t) = 0$ and a nonnegative length function ℓ on $E(f)$, if the reduced length $\ell_d(e) = \ell(e) + d(w) - d(v)$ of each edge $e = (v, w) \in E(f)$ is nonnegative, then d is called a distance labeling.) Let $d_\ell(v)$ be the length of a shortest path from v to t in $N(f)$ with length function ℓ as before. Then d_ℓ is the maximum distance labeling and the requirement for d to be a distance labeling implies there is no path in $N(f)$ from s to t since $d \leq d_\ell$ holds (i.e., $d(v) \leq d_\ell(v)$ for any $v \in V$). Furthermore, a proof by induction shows that, for any distance labeling d in the algorithm, $d(v) \leq \min\{d_\ell(v, s) + n, d_\ell(v, t)\}$, where $d_\ell(v, w)$ is the length of a shortest path from v to w in $N(f)$. An edge $e = (v, w)$ of $N(f)$ is called *eligible* if $d(v) = d(w) + 1$. Note that $d_\ell(v) = d_\ell(v,t)$ and that the term *eligible* is defined by using a distance labeling d while the term *admissible* was defined by using the maximum distance labeling d_ℓ.

More specifically, the algorithm initially sets

$$f(e) := \begin{cases} cap(e) & \text{if } e \in \delta^+(s), \\ 0 & \text{if } e \in E - \delta^+(s), \end{cases}$$

$$d(v) := \min\{d_\ell(v,s) + n, d_\ell(v,t)\}.$$

Thus, initially, f is a preflow and d is a distance labeling. Note that $d(s) = n$ and $d(t) = 0$ since there is no edge in $N(f)$ out of s and $d_\ell(s,t) = \infty$. The algorithm consists of repeating the following two steps which maintain a preflow f and distance labeling d, in any order, until no vertex is active:

Push(v,w).
Applicability: Vertex v is active and edge $e = (v,w)$ is eligible.
Action: Increase $f(e)$ by $\min\{ex(v), cap_f(e)\}$
(we mean that we first set
$f(e) := f(e) + \min\{ex(v), cap_f(e)\}$
and then
$f(e) := f(e) - \min\{f(e), f(e^R)\}; \quad f(e^R) := f(e^R) - \min\{f(e), f(e^R)\}$
as before). The push is *saturating* if $e = (v,w)$ is saturated after the push and *nonsaturating* otherwise.

Relabel(v).
Applicability: Vertex v is active and no edge $e \in E(f)$ out of v is eligible.
Action: Replace $d(v)$ by $\min\{d(w) + 1 \mid e = (v,w) \in E(f) \text{ out of } v \text{ is unsaturated}\}$.

When the algorithm terminates, f is a maximum flow. Goldberg-Tarjan derived the following bounds on the number of steps required by the algorithm.

Lemma 2.12 *[26] Relabeling a vertex v strictly increases $d(v)$. No vertex label $d(v)$ exceeds $2n - 1$, and the total number of relabelings is $O(n^2)$.*

Lemma 2.13 *[26] There are at most $O(mn)$ saturating pushes and at most $O(n^2m)$ nonsaturating pushes.*

Efficient implementations of the above algorithm require a mechanism for selecting pushing and relabeling steps to perform. Goldberg-Tarjan proposed the following method: For each vertex, construct a (fixed) list $A(v)$ of the edges out of v. Designate one of these edges, initially the first on the list, as the current edge out of v. To execute the algorithm, repeat the following step until there are no active vertices:

Push/Relabel(v).
Applicability: Vertex v is active.
Action: If the current edge (v,w) of v is eligible, perform push(v,w). Otherwise, if (v,w) is not the last edge on $A(v)$, make the next edge after (v,w) the current one. Otherwise, perform relabel(v) and make the first edge on $A(v)$ the current one.

With this implementation, the algorithm runs in $O(nm)$ time plus $O(1)$ time per nonsaturating push. This gives an $O(n^2m)$ time bound for any order of selecting vertices for push/relabel steps. Making the algorithm faster requires reducing the time spent on nonsaturating pushes. The number of such pushes can be reduced by selecting vertices for push/relabel steps carefully. Goldberg-Tarjan [26] showed that first-in, first-out selection (first active, first selected) reduces the number of nonsaturating pushes to $O(n^3)$. Cheriyan-Maheshwari [10] showed that highest label selection (always pushing flow from a vertex with highest label) reduces the number of nonsaturating pushes to $O(n^2m^{1/2})$. (The latter rule was first proposed by Goldberg [22], who gave an $O(n^3)$ bound.)

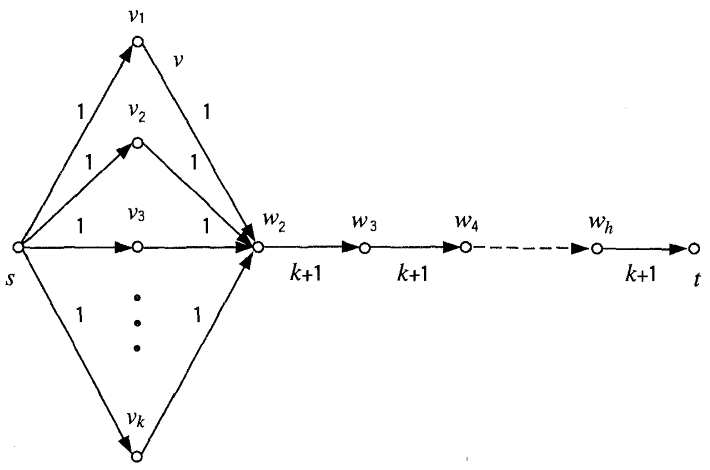

Figure 4: Example of a level network $N_L(f)$ whose edges on the path from w_2 to t are searched many times by the Dinic algorithm.

2.5. Dynamic trees implementations

The Dinic algorithm finds a blocking flow on a level network $N_L(f)$ by doing the depth-first search from s in $N_L(f)$ and saturating one edge at a time. But this wastes much time since edges are searched many times as shown in Figure 4. To reduce the time per edge saturation, we should keep track of the flow by using appropriate data structures. Galil-Namaad [21] and Shiloach [43] discovered a method of this kind that runs in $O(m(\log n)^2)$ time. Sleator-Tarjan [44] improved the bound to $O(m \log n)$ inventing the data structure of dynamic trees. Goldberg-Tarjan [26] obtained a $O(mn \log(n^2/m))$ time algorithm for finding a maximum flow on a network N based on the push-relabel method implemented by using dynamic trees.

3. Maximum Flow Algorithms for Integral Capacity Networks

As described in Theorem 2.11, Karzanov [36] and Even-Tarjan [18] have shown independently that the Dinic algorithm runs in $O(\min\{m^{1/2}, n^{2/3}\}m)$ time on networks with unit capacities and no parallel edges. For general integral capacity networks, Edmonds-Karp [17] obtained an $O(m^2 \log U)$ time algorithm and Dinic [16] and Gabow [20] improved this bound to $O(mn \log U)$ (note that U is the maximum edge capacity of a network). They used a scaling method. Ahuja-Orlin [3] combined this scaling method with the push-relabel method of Goldberg-Tarjan based on preflows [26] and obtained an $O(mn+n^2 \log U)$ time algorithm. Ahuja-Orlin-Tarjan [4] explored possible improvements to Ahuja-Orlin algorithm and obtained an $O(mn + n^2(\log U)^{1/2})$ time algorithm and an $O(mn \log(2 + (n/m)(\log U)^{1/2}))$ time algorithm. Goldberg-Rao [23] further improved these bounds and obtained an $O(\min\{m^{1/2}, n^{2/3}\}m \log(n^2/m) \log U)$ time algorithm beating drastically the lower bound $\Omega(mn)$ on the path decomposition method under the similarity assumption $\log U = O(\log n)$ introduced by Gabow.

In this section, we review the scaling method of Gabow [20] and the Ahuja-Orlin algorithm briefly and describe the Goldberg-Rao algorithm extensively.

3.1. Scaling method

Let $N = (G, cap, s, t)$ be a given network with integral capacities. Set $N^0 := N$ and recursively define $N^{i+1} = (G^{i+1}, cap^{i+1}, s, t)$ to be the network obtained from $N^i = (G^i, cap^i, s, t)$ by setting $cap^{i+1}(e) := \lfloor cap^i(e)/2 \rfloor$ for each edge e in N^i ($i = 0, 1, ..., \lfloor \log U \rfloor - 1$). Since every edge is of capacity 0 or 1 in $N^{\lfloor \log U \rfloor}$, a maximum flow $f^{\lfloor \log U \rfloor}$ can be obtained in $O(mn)$ time. Assume that a maximum flow f^{i+1} on N^{i+1} is already obtained and consider the residual network $N^i(2f^{i+1})$ of N^i with respect to flow $2f^{i+1}$. Note that any path P in $N^i(2f^{i+1})$ from s to t has the residual path capacity $\Delta(P) \leq 1$, since otherwise P is a path in $N^{i+1}(f^{i+1})$ from s to t with residual path capacity $\Delta(P) \geq 1$ and contradicts that f^{i+1} is a maximum flow on N^{i+1}. Thus, for a maximum flow g^i on $N^i(2f^{i+1})$, $f^i := g^i + 2f^{i+1}$ is a maximum flow on N^i and $val(g^i) = val(f^i) - val(2f^{i+1}) \leq m$ since $g^i = f^i - 2f^{i+1}$ can be decomposed into a set of paths in $N^i(2f^{i+1})$ each with the residual path capacity 1. This implies that we can find a maximum flow f^i on N^i by at most m augmentations. Each augmentation can be done by finding a path from s to t in $O(m)$ time and we can obtain f^i on N^i in $O(m^2)$ time. Thus, a maximum flow in N can be obtained in $O(m^2 \log U)$ time. Gabow [20] improved this bound to $O(mn \log U)$.

3.2. Ahuja-Orlin algorithm

In this section we describe the Ahuja-Orlin scaling algorithm [3] by borrowing the summary given by Ahuja-Orlin-Tarjan [4]. This enables us to understand the Goldberg-Rao algorithm in the next subsection easily. The intuitive idea behind the Ahuja-Orlin algorithm is to move large amounts of flow when possible. One way to apply this idea to the preflow algorithm is to always push flow from a vertex of large excess to a vertex of small excess, or to the sink. The effect of this is to reduce the maximum excess at a rapid rate.

Making this method precise requires specifying when an excess is large and when it is small. For this purpose the Ahuja-Orlin algorithm uses an excess bound Δ and an integer scaling factor $k \geq 2$. A vertex v is said to have *large excess* if its excess exceeds Δ/k and *small excess* otherwise. As the algorithm proceeds, k remains fixed, but Δ periodically decreases. Initially, Δ is the smallest power of k such that $\Delta \geq U$. The algorithm maintains the invariant that $ex(v) \leq \Delta$ for every vertex v. This requires changing the pushing step described in Section 2.4 to the following:

Push(v, w).

Applicability: Vertex v is active and edge $e = (v, w)$ is eligible.

Action: If $w \neq t$, increase $f(e)$ by $\min\{ex(v), cap_f(e), \Delta - ex(w)\}$. Otherwise ($w = t$), increase $f(e)$ by $\min\{ex(v), cap_f(e)\}$.

The algorithm consists of a number of scaling phases, during each of which Δ remains constant. A phase consists of repeating push/relabel steps, using the following selection rule, until no active vertex has large excess, and then replacing Δ by Δ/k. The algorithm terminates when there are no active vertices.

Large excess, smallest label selection: Apply a push/relabel step to an active vertex v of large excess; among such vertices, choose one of smallest label.

Since the edge capacities are integers, the algorithm terminates after at most $\lfloor \log_k U \rfloor + 1$ phases. After $\lfloor \log_k U \rfloor + 1$ phases, $\Delta < 1$, which implies that f is a flow, since the algorithm maintains integrality of vertex excesses. Ahuja-Orlin derived a bound of $O(kn^2 \log_k U)$ on the total number of nonsaturating pushes. Choosing k to be a constant independent of n gives a total time bound of $O(nm + n^2 \log U)$ for this algorithm, given an efficient implementation of vertex selection rule. One way to implement the rule is to maintain an array of sets indexed by vertex label, each set containing all large excess vertices with corresponding label, and to maintain a pointer to the nonempty set of smallest index. The total time needed to maintain this structure is $O(nm + n^2 \log U)$.

Ahuja-Orlin-Tarjan [4] improved the Ahuja-Orlin algorithm and obtained an $O(mn + n^2(\log U)^{1/2})$ time algorithm and an $O(mn \log(2 + (n/m)(\log U)^{1/2}))$ time algorithm.

3.3. Goldberg-Rao algorithm

In this section, we give an overview of the Goldberg-Rao algorithm which drastically cleared the barrier $\Omega(mn)$ of path decomposition methods under similarity assumption. Its ballpark complexity is $O^*(\min\{m^{3/2}, mn^{2/3}\})$.

Let f^* be a maximum flow and let f be a current flow on N. They tried to estimate the difference between the maximum flow value and the current flow value $val(f^*) - val(f)$ using some upper bound F. Since U is the maximum edge capacity and $val(f^*) \leq \sum_{(s,w)\in\delta^+(s)} cap(s,w) \leq nU$, they first set $F := nU$.

In each phase of the Goldberg-Rao algorithm, the updates are repeated until $val(f^*) - val(f)$ is assured to be no more than $F/2$. Then a next phase starts setting $F := F/2$. If $F < 1$, then a maximum flow is obtained since each capacity is integer and we terminate. Thus the number of phases is at most $1 + \log(nU)$.

At the beginning of each phase, we define Δ to be

$$\Delta = \left\lceil \frac{F}{\min\{m^{1/2}, n^{2/3}\}} \right\rceil$$

and find a flow f'' of value Δ or a blocking flow f'' of value at most Δ and augment $f := f + f''$. In each phase, augmentations using a flow of value Δ occur at most $\min\{m^{1/2}, n^{2/3}\}$ times and augmentations using a blocking flow can also be shown to occur at most $O(\min\{m^{1/2}, n^{2/3}\})$ times. Furthermore, such an augmentation using a flow f'' of value Δ or a blocking flow f'' of value at most Δ can be obtained in $O(m \log(n^2/m))$ time based on the special structure of the treated network. Thus the time complexity of the Goldberg-Rao algorithm becomes $O(\min\{m^{1/2}, n^{2/3}\} m \log(n^2/m) \log(nU))$. We can further improve $\log(nU)$ in the bound to $\log U$.

The key point in the Goldberg-Rao algorithm is a binary length function ℓ such that $\ell(a) = 0$ if $cap_f(a) \geq \alpha$ and $\ell(a) = 1$ if $0 < cap_f(a) < \alpha$ for some threshold α. However, the existence of edges of length 0 makes Lemma 2.7 violated in some cases (Figure 3). To cope with these situations, they introduced a notion of *special edges*.

Goldberg-Rao algorithm

1. Initialize $f := 0$, $F := nU$, and $\Delta := \lceil \frac{F}{\min\{m^{1/2}, n^{2/3}\}} \rceil$.
2. (Iterations of phase step) (we choose $\alpha = 5\Delta - 4$ and $\alpha_L = 2\Delta - 1$)
   (a) If $F < 1$ then terminate (f is a maximum flow). Otherwise, go to the following update step.
   (b) (Iterations of update step)
      i. Compute the residual network $N(f)$ and define a length function $\ell(a)$ on $a \in E(f)$ as follows (Figure 5(a)):

$$\ell(a) = \begin{cases} 0 & (cap_f(a) \geq \alpha) \\ 1 & (0 < cap_f(a) < \alpha). \end{cases}$$

      ii. Compute the maximum distance label d_ℓ of $N(f)$ for ℓ. If $d_\ell(s) = 0$, then find a flow g of value α_L along a shortest path of $N(f)$ from s to t, and go to x (2(b)x). Otherwise, proceed to the following.

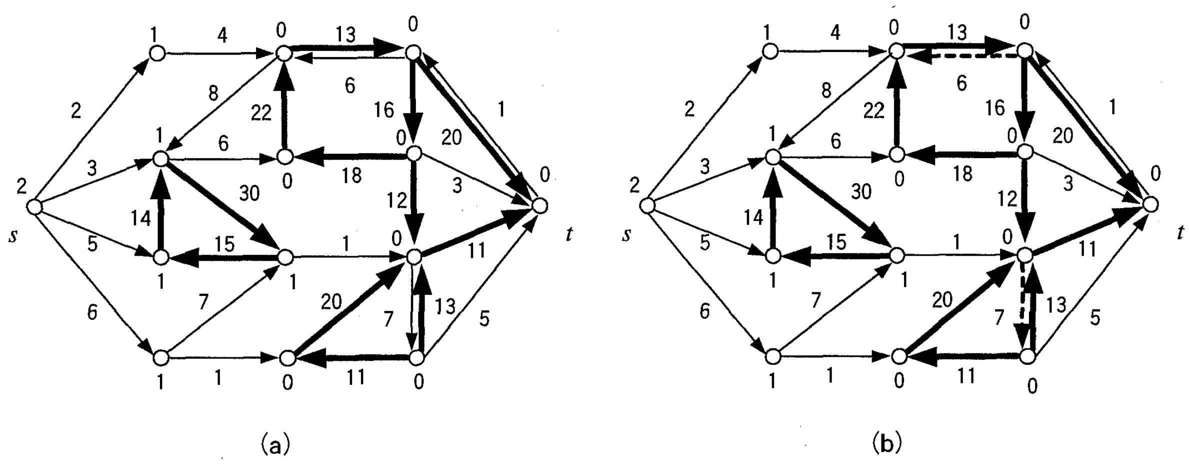

Figure 5: A residual network $N(f)$ with $\Delta = 3$, $\alpha = 11$ and $\alpha_L = 5$. (a) Thick edges are of length 0 and the others are of length 1 in ℓ. (b) Thick broken edges are special edges. Thick edges are of length 0 and the others are of length 1 in $\bar{\ell}$.

iii. Find a canonical cut $\delta^+(S)$ of $N(f)$ of minimum capacity.

iv. If $cap(S) \leq F/2$ then set $F := cap(S)$ and $\Delta := \lceil \frac{F}{\min\{m^{1/2}, n^{2/3}\}} \rceil$ and go to 2 (terminate the update step and go to the next phase step). Otherwise, proceed to the following.

v. Consider $a = (u, v) \in E(f)$ satisfying $\alpha_L \leq cap_f(a) < \alpha$, $d_\ell(u) = d_\ell(v)$ and $cap_f(a^R) \geq \alpha$ to be a *special edge* and modify the length function ℓ above to the following length function $\bar{\ell}$ (Figure 5(b)):

$$\bar{\ell}(e) = \begin{cases} 0 & \text{(if } e \text{ is a special edge)} \\ \ell(e) & \text{(otherwise).} \end{cases}$$

(Note that $d_\ell = d_{\bar{\ell}}$, i.e., $d_\ell(v) = d_{\bar{\ell}}(v)$ for each $v \in V$).

vi. Compute the shortest residual network $N_{\bar{\ell}}(f)$ of $N(f)$ with respect to $\bar{\ell}$ (Figure 6(a)). (Note that $N_{\bar{\ell}}(f)$ contains no cycle of positive length.)

vii. Compute the network $N''_{\bar{\ell}}(f)$ obtained from $N_{\bar{\ell}}(f)$ by contracting each strongly connected component (it consists of only edges of length 0) into a vertex (Figure 6(b)). (Note that $N''_{\bar{\ell}}(f)$ contains no directed cycle.)

viii. Find a flow f'' of value Δ or a blocking flow f'' of value at most Δ in $N''_{\bar{\ell}}(f)$.

ix. Extend the flow f'' in $N''_{\bar{\ell}}(f)$ to a flow g in $N_{\bar{\ell}}(f)$ by extending each contracted vertex v of $N''_{\bar{\ell}}(f)$ to the corresponding strongly connected component $ST(v)$ of $N_{\bar{\ell}}(f)$ (Figure 7(a)) and properly distributing the flow going through v in $ST(v)$ as follows if $ST(v)$ has positive or negative excess vertices (for flow f'' and $ex_{f''}(u) = \sum_{e\in\delta^+(u)} f''(e) - \sum_{e\in\delta^-(u)} f''(e)$, a vertex $u \in V - \{s, t\}$ is called a *positive excess* vertex if $ex_{f''}(u) > 0$ and a *negative excess* vertex if $ex_{f''}(u) < 0$). Choose a vertex r_v in $ST(v)$ as

$$r_v := \begin{cases} s & (s \in ST(v)) \\ t & (t \in ST(v), s \notin ST(v)) \\ \text{any positive excess vertex} & (s, t \notin ST(v)) \end{cases}$$

and construct two directed rooted trees $IN(v)$ and $OUT(v)$ with the same root r_v in $ST(v)$ where $IN(v)$ is an intree and $OUT(v)$ is an out tree (Figure

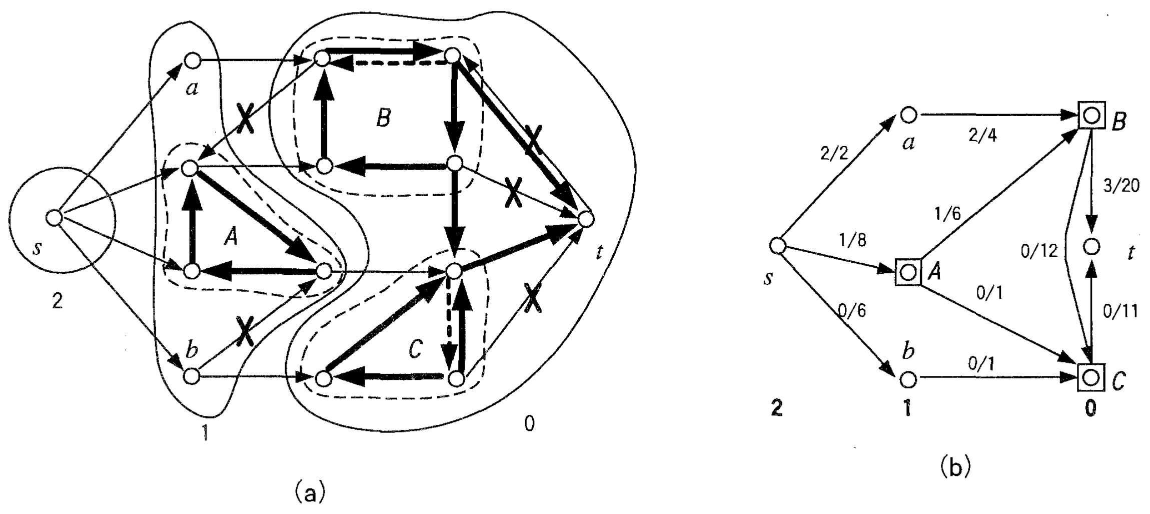

Figure 6: (a) Maximum distance labeling d_ℓ of the residual network $N(f)$ in Figure 5 with $\Delta = 3$, $\alpha = 11$ and $\alpha_L = 5$. A, B, C are strongly connected components and edges marked with $\times$ are deleted in $N_{\bar{\ell}}(f)$. (b) $N''_{\bar{\ell}}(f)$ obtained from $N_{\bar{\ell}}(f)$ by contracting the strongly connected components and a flow f'' of value Δ in $N''_{\bar{\ell}}(f)$ $(f''(e)/cap_f(e))$.

7(b)) (there is a unique path from each vertex to r_v in $IN(v)$ and there is a unique path from r_v to each vertex in $OUT(v)$). Along a path from each positive excess vertex u to r_v in $IN(v)$ we distribute a flow of value $ex_{f''}(u) > 0$. Then along a path from r_v to each negative excess vertex u in $OUT(v)$ we distribute a flow of value $-ex_{f''}(u) > 0$. The flow g in $N_{\bar{\ell}}(f)$ is obtained in this way (Figure 8).

x. Update the flow f to $f := f + g$ and go to (b).

This algorithm uses a flow f'' of value at most Δ in the network $N''_{\bar{\ell}}(f)$ with no positive cycle and satisfies the following Δ-*invariant*:

for each vertex v, the incoming flow value $\sum_{e\in\delta^-(v)} f''(e)$ and the outgoing flow value $\sum_{e\in\delta^+(v)} f''(e)$ are both at most Δ.

Thus, $f''(a) \leq \Delta$ for each edge a in $N''_{\bar{\ell}}(f)$. Furthermore, for each strongly connected component $ST(v)$ corresponding to a contracted vertex v in $N''_{\bar{\ell}}(f)$, a flow of value $ex_{f''}(u)$ is augmented along a path from each positive excess vertex u to r_v in $IN(v)$ and a flow of value $-ex_{f''}(u)$ is augmented along a path from r_v to each negative excess vertex u in $OUT(v)$. Thus, the total flow value through each vertex in $IN(v)$ is at most $\Delta - 1$ if $ST(v)$ does not contain s, t since r_v is positive excess vertex. Similarly, the total flow value through each vertex in $OUT(v)$ is at most Δ. This implies that $g(a) \leq 2\Delta - 1$ for each edge a in a strongly connected component $ST(v)$ without s, t for the extended flow g of f''. If $r_v = t$ then there is no negative excess vertex in $ST(v)$ since $N''_{\bar{\ell}}(f)$ has no directed cycle and $g(a) \leq \Delta$ for each edge a in $ST(v)$. Similarly, if $r_v = s$ then there is no positive excess vertex in $ST(v)$ and $g(a) \leq \Delta$ for each edge a in $ST(v)$ (note that if $ST(v) \ni s$ then $ST(v) \not\ni t$, since otherwise $d_\ell(s) = 0$ and iii to ix (2(b)iii to 2(b)ix) are skipped).

Note that we have chosen $\alpha = 5\Delta - 4$ and $\alpha_L = 2\Delta - 1$. Thus, an edge in a strongly connected component of $N_{\bar{\ell}}(f)$ is of length 0 and its residual capacity is large enough to augment the flow since $\alpha = 5\Delta - 4 > \alpha_L = 2\Delta - 1$ if $\Delta \geq 2$. Otherwise ($\Delta = 1$), $\alpha = 5\Delta - 4 = 1$ and every edge is of length 0. In this case, a flow of value $\Delta = 1$ is a

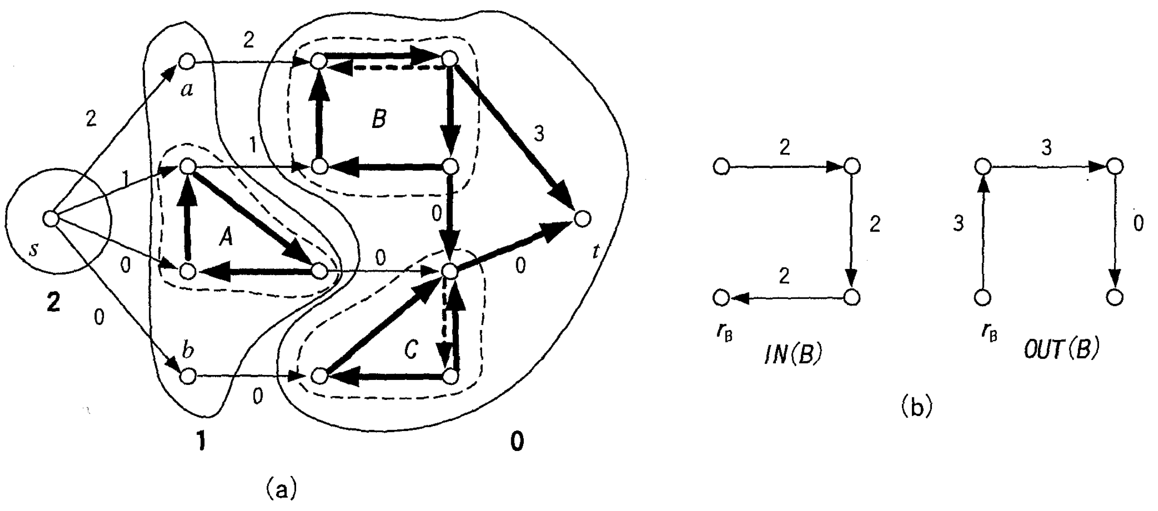

Figure 7: (a) Flow f'' on $N_{\bar{\ell}}(f)$. (b) Flow g on $IN(B)$ and $OUT(B)$ of the strongly connected component $ST(B)$.

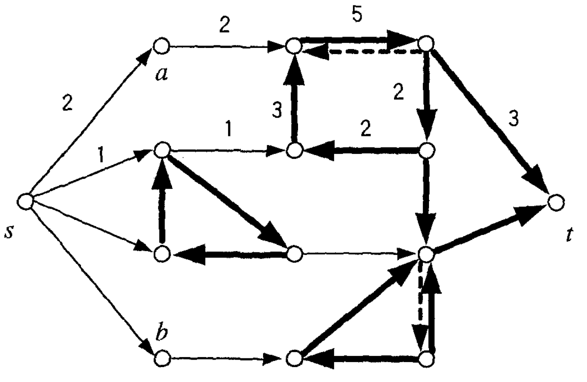

Figure 8: Flow g on $N_{\bar{\ell}}(f)$ ($g(e) = 0$ is omitted).

path from s to t and each $g(a)$ in $ST(v)$ is at most 1 and thus, each strongly connected component $ST(v)$ corresponding to a contracted vertex v has at most one positive excess vertex and at most one negative excess vertex. Similarly, If $d_\ell(s) = 0$ in 2(b)ii, then a shortest path P from s to t is of length 0, and the residual capacities of edges in P are all at least α and sufficient enough to update the flow. Thus, we have the following lemma.

Lemma 3.1 *In each update step of the algorithm, $g(a) = 0$ for an edge a in $N(f)$ but not in $N_{\bar{\ell}}(f)$, $g(a) \leq \Delta$ for an edge a with $\bar{\ell}(a) = 1$ and $g(a) \leq 2\Delta - 1$ for an edge a with $\bar{\ell}(a) = 0$. Thus, each update step can be done correctly.*

For each update step, let f, g and $\bar{\ell}$ denote the flows and the length function just before x (2(b)x) and let f' and ℓ' denote the flow and the length function in the next iteration of update step (before considering special edges). Since we consider special edges and thus the lengths of some edges may become longer even if their residual capacities are increased, we need proofs to say that Lemmas 2.5 and 2.7 still hold. The lemmas below hold in the Goldberg-Rao algorithm and give an answer to this question. Recall there that edge $e = (u, v) \in E(f)$ was called admissible in $\bar{\ell}$ before if $d_{\bar{\ell}}(u) = d_{\bar{\ell}}(v) + \bar{\ell}(e)$, and this coincides with the following: edge $e = (u, v) \in E(f)$ is admissible in $\bar{\ell}$ if $(d_\ell(u) = d_\ell(v) + 1)$ or $(d_\ell(u) = d_\ell(v)$ and $\bar{\ell}(u, v) = 0)$. Thus, $N_{\bar{\ell}}(f)$ is a subnetwork of the network consisting of all admissible edges.

Lemma 3.2 *If $cap_{f'}(a) > cap_f(a)$, then $g(a^R) > 0$. Furthermore, if $g(a) = 0$, then $cap_{f'}(a) > cap_f(a)$ if and only if $g(a^R) > 0$. (We assume $g(e) = 0$ if $e \in E$ is not an edge of $N_{\bar{\ell}}(f)$ and, thus, $g(e) \geq 0$ for each edge $e \in E$.)*

Proof. Since $cap_{f'}(a) = cap(a) - f'(a) + f'(a^R)$ and $cap_f(a) = cap(a) - f(a) + f(a^R)$, $cap_{f'}(a) > cap_f(a)$ if and only if $f'(a^R) - f'(a) > f(a^R) - f(a)$. Let $\beta = \min\{f(a) + g(a), f(a^R) + g(a^R)\}$. Then

$$f'(a) = f(a) + g(a) - \beta = \max\{0, f(a) + g(a) - (f(a^R) + g(a^R))\},$$
$$f'(a^R) = f(a^R) + g(a^R) - \beta = \max\{0, f(a^R) + g(a^R) - (f(a) + g(a))\}.$$

Thus, $f'(a^R) - f'(a) = f(a^R) + g(a^R) - (f(a) + g(a))$ and $cap_{f'}(a) > cap_f(a)$ (i.e., $f'(a^R) - f'(a) > f(a^R) - f(a)$) if and only if $g(a^R) > g(a) \geq 0$. □

Lemma 3.3 *d_ℓ is a distance labeling with respect to length function ℓ' and $d_{\ell'} \geq d_\ell$.*

Proof. Suppose that there were an edge $a = (u, v) \in E(f')$ with $d_\ell(u) > d_\ell(v) + \ell'(a)$.

If $a = (u, v) \in E(f)$ then $d_\ell(v) + \bar{\ell}(a) \geq d_\ell(u) > d_\ell(v) + \ell'(a)$ since $d_\ell = d_{\bar{\ell}}$ is a distance labeling with respect to length functions ℓ and $\bar{\ell}$ and $d_\ell(u) \leq d_\ell(v) + \bar{\ell}(a)$. Thus, we have $\bar{\ell}(a) = 1$ $(cap_f(a) < \alpha)$ and $\ell'(a) = 0$ $(cap_{f'}(a) \geq \alpha)$ and the residual capacity of a is increased. Similarly, if $a = (u, v) \notin E(f)$ then $cap_f(a) = 0$ and the residual capacity of a is increased in this case since $a = (u, v) \in E(f')$ $(cap_{f'}(a) > 0)$.

Thus, in either case, the residual capacity of a is increased and $g(a^R) > 0$ by Lemma 3.2. This implies that $a^R = (v, u)$ is not only in $E(f)$ but also in the shortest residual network $N_{\bar{\ell}}(f)$ with respect to $\bar{\ell}$. Thus, $a^R = (v, u)$ is an admissible edge and $d_\ell(v) = d_\ell(u) + \bar{\ell}(a^R) \geq d_\ell(u)$. This, however, contradicts $d_\ell(u) > d_\ell(v) + \ell'(a) \geq d_\ell(v)$. Thus, we have shown that there is no edge $a = (u, v) \in E(f')$ satisfying $d_\ell(u) > d_\ell(v) + \ell'(a)$. □

Lemma 3.4 *Let $\alpha = 5\Delta - 4$ and $\alpha_L = 2\Delta - 1$. If f'' is a blocking flow on $N''_{\bar{\ell}}(f)$, then g is a blocking flow on the shortest residual network $N_{\bar{\ell}}(f)$ of $N(f)$. Furthermore, if $\Delta \geq 2$ then $d_{\ell'}(s) > d_\ell(s)$.*

Proof. Since f'' is a blocking flow on $N''_{\bar{\ell}}(f)$, there is no path from s to t in the network obtained from $N''_{\bar{\ell}}(f)$ by deleting all edges a with $f''(a) = cap_f(a)$. Thus, for an extended flow g obtained from f'' by extending each contracted vertex of $N''_{\bar{\ell}}(f)$ to the corresponding strongly connected component in $N_{\bar{\ell}}(f)$, there is no path from s to t in the network $N^-_{\bar{\ell}}(f)$ obtained from $N_{\bar{\ell}}(f)$ by deleting all edges a with $g(a) = cap_f(a)$. Thus, g is also a blocking flow on $N_{\bar{\ell}}(f)$.

We will show below that $d_{\ell'}(s) > d_\ell(s)$ if $\Delta \geq 2$. Let $P = (s = v_0, v_1, ..., v_k = t)$ be a shortest path from s to t in the residual network $N(f')$ with respect to length ℓ' and $f' := f + g$. Of course, if such a path P does not exist, then $d_{\ell'}(s) = \infty$ and the lemma holds. Thus, we assume here such a path P exists. If the length $d_{\ell'}(s)$ of P is at least $d_\ell(s) + 1$ then the lemma also holds. Thus, we will assume P is of length $d_{\ell'}(s) = d_\ell(s)$ and derive a contradiction. Let $a_i = (v_i, v_{i+1})$. Then the length $d_{\ell'}(s)$ of P can be written by

$$d_{\ell'}(s) = \sum_{i=0}^{k-1} \ell'(a_i) = d_\ell(s) + \sum_{i=0}^{k-1} (\ell'(a_i) + d_\ell(v_{i+1}) - d_\ell(v_i))$$

since $d_\ell(t) = 0$. By Lemma 3.3, $\ell'(a_i) + d_\ell(v_{i+1}) - d_\ell(v_i) \geq 0$ since d_ℓ is a distance labeling with respect to ℓ' and thus, $d_{\ell'}(s) = d_\ell(s)$ if and only if $\ell'(a_i) + d_\ell(v_{i+1}) = d_\ell(v_i)$ for $i = 0, 1, ..., k - 1$.

Now suppose that $d_{\ell'}(s) = d_\ell(s)$. Then, by the argument above,

$$\ell'(a_i) + d_\ell(v_{i+1}) = d_\ell(v_i) \text{ for all } i = 0, 1, ..., k-1. \tag{3.1}$$

Since g is a blocking flow of $N_{\bar{\ell}}(f)$, there is an edge a in P which is not contained in $N_{\bar{\ell}}(f)$. We choose $a_i = (v_i, v_{i+1})$ as the first such edge in P (that is, all $a_j = (v_j, v_{j+1})$ with $j = 0, 1, ..., i-1$ are in $N_{\bar{\ell}}(f)$ and $a_i = (v_i, v_{i+1})$ is not in $N_{\bar{\ell}}(f)$). Since each a_j with $j = 0, 1, ..., i-1$ is in $N_{\bar{\ell}}(f)$, we have $\bar{\ell}(a_j) + d_\ell(v_{j+1}) = d_\ell(v_j)$. Thus, $\ell'(a_j) = \bar{\ell}(a_j)$ for each a_j with $j = 0, 1, ..., i-1$. If a_i is an edge in $N(f)$ and $d_\ell(v_{i+1}) + \bar{\ell}(a_i) = d_\ell(v_i)$, then, by (3.1), $\bar{\ell}(a_i) = \ell'(a_i)$ and the path joining the subpath $P(s, v_i)$ of P from s to v_i, which is a path in $N_{\bar{\ell}}(f)$, and edge $a_i = (v_i, v_{i+1})$ in $N(f)$ and a path $Q(v_{i+1}, t)$ of length $d_\ell(v_{i+1})$ from v_{i+1} to t in $N_{\bar{\ell}}(f)$ becomes a path of $N(f)$ of length $d_\ell(s)$ from s to t. Thus, a_i is in $N_{\bar{\ell}}(f)$, a contradiction. Thus, we have:

(a) a_i is an edge in $N(f)$ and $d_\ell(v_{i+1}) + \bar{\ell}(a_i) > d_\ell(v_i)$; or

(b) a_i is not an edge in $N(f)$.

If (a) occurs, then, by (3.1), $\bar{\ell}(a_i) = 1$ ($cap_f(a_i) < \alpha$) and $\ell'(a_i) = 0$ ($cap_{f'}(a_i) \geq \alpha$). If (b) occurs, then $cap_f(a_i) = 0$ but $cap_{f'}(a_i) > 0$ since a_i is in $N(f')$. Thus, in either case, we have $cap_{f'}(a_i) > cap_f(a_i)$. Then $g(a_i^R) > 0$ by Lemma 3.2 and $a_i^R = (v_{i+1}, v_i)$ is an edge in $N_{\bar{\ell}}(f)$. Thus, $d_\ell(v_{i+1}) = \bar{\ell}(a_i^R) + d_\ell(v_i)$ and $d_\ell(v_i) = \ell'(a_i) + d_\ell(v_{i+1}) \geq d_\ell(v_{i+1}) = \bar{\ell}(a_i^R) + d_\ell(v_i) \geq d_\ell(v_i)$ by (3.1) and we have $\ell'(a_i) = \bar{\ell}(a_i^R) = 0$ and $d_\ell(v_i) = d_\ell(v_{i+1})$. Then, by the definition of ℓ', $cap_{f'}(a_i) \geq \alpha$. On the other hand, since a_i is not in $N_{\bar{\ell}}(f)$, we have $g(a_i) = 0$ and $cap_{f'}(a_i) - cap_f(a_i) = g(a_i^R)$ by the proof of Lemma 3.2. Thus, $cap_f(a_i) = cap_{f'}(a_i) - g(a_i^R) \geq \alpha - (2\Delta - 1) \geq \alpha_L > 0$ (note that $g(a_i^R) \leq 2\Delta - 1$ by Lemma 3.1) and only (a) can occur ((b) cannot occur). Then, however, a_i is a special edge or $cap_f(a_i) \geq \alpha$, and we have $\bar{\ell}(a_i) = 0$, a contradiction. Thus, we have a contradiction in any case when we assume that $d_{\ell'}(s) = d_\ell(s)$. □

Next we estimate the number of iterations of update step and that of phase step. We first consider the number of iterations of update step in each phase.

Since $\Delta = \lceil \frac{F}{\min\{m^{1/2}, n^{2/3}\}} \rceil$, the number of updates of using a flow f'' of value Δ is at most $\min\{m^{1/2}, n^{2/3}\}$. The number of updates of using a blocking flow f'' of value at most Δ is analyzed as follows. We first consider the case when $\Delta \geq 2$. Let $M \leq \alpha - 1 = 5(\Delta - 1)$ be the maximum residual capacity of edges of length 1 in each phase. Then, by Lemmas 2.8 and 2.9, we have $Vol_{f,\ell} \leq mM$, $cap_f(S) \leq mM/d_\ell(s)$ and $cap_f(S) \leq (\frac{n}{d_\ell(s)})^2 M$. From this and Lemma 3.4, we can show that the number of updates of using a blocking flow f'' in each phase is $O(\min\{m^{1/2}, n^{2/3}\})$ as follows: If $\Delta = \lceil \frac{F}{m^{1/2}} \rceil$ ($m^{1/2} = \min\{m^{1/2}, n^{2/3}\}$), then, after the $10m^{1/2}$-th update of using a blocking flow, we have $d_\ell(s) \geq 10m^{1/2}$ and

$$cap_f(S) \leq \frac{mM}{d_\ell(s)} \leq \frac{5m(\Delta - 1)}{10m^{1/2}} \leq \frac{5mF}{10m^{1/2}m^{1/2}} = \frac{F}{2}.$$

If $\Delta = \lceil \frac{F}{n^{2/3}} \rceil$ ($n^{2/3} = \min\{m^{1/2}, n^{2/3}\}$), then, after the $4n^{2/3}$-th update of using a blocking flow, we have $d_\ell(s) \geq 4n^{2/3}$ and

$$cap_f(S) \leq M\left(\frac{n}{d_\ell(s)}\right)^2 \leq 5(\Delta - 1)\frac{n^2}{d_\ell(s)^2} \leq \frac{5(\Delta - 1)n^2}{16n^{4/3}} \leq \frac{5n^2F}{16n^{2/3}n^{4/3}} = \frac{5F}{16} \leq \frac{F}{2}.$$

We next consider the case when $\Delta = 1$ ($F \leq \min\{m^{1/2}, n^{2/3}\}$). In this phase, each update of using a blocking flow f'' increases the flow value by 1 and the number of updates using a blocking flow f'' is at most $\min\{m^{1/2}, n^{2/3}\}$.

The number of phases is at most $\log(nU)+1$ as mentioned before. Each update step can be done by finding a blocking flow on an acyclic network (i.e., network with no directed cycle). If a blocking flow is of value greater than Δ then we decrease the blocking flow until it becomes a flow of value Δ by post-processing. This post-processing can be done in $O(m)$ time. Finding the strongly connected components and computing a canonical cut of minimum capacity also require only $O(m)$ time. The shortest path problem is also solved in $O(m)$ time since the length function ℓ takes on only values 0 and 1 (d_ℓ can also be computed in $O(m)$ time). Adjusting a flow in strongly connected components $ST(v)$ using intree $IN(v)$ and out tree $OUT(v)$ also requires $O(m)$ time in total. Thus, dominating part in the algorithm is to find a blocking flow on an acyclic network. This can be done in $O(m\log(n^2/m))$ time by the Goldberg-Tarjan algorithm [27].

Thus, we have the following lemma.

Lemma 3.5 *A maximum flow of an integral capacity network $N=(G,cap,s,t)$ can be found in $O(m\min\{m^{1/2},n^{2/3}\}\log(n^2/m)\log(nU))$ time by the Goldberg-Rao algorithm (U is the maximum capacity of edges in N).*

We can reduce $\log(nU)$ to $O(\log U)$ in the above time complexity. The number of phases between the phase of $\Delta\le U$ and the phase of $\Delta=1$ is at most $\log U+1$ and the total time in these phases is $O(m\min\{m^{1/2},n^{2/3}\}\log(n^2/m)\log U)$. The total time in the phase of $\Delta=1$ is also $O(m\min\{m^{1/2},n^{2/3}\})$ as described above.

Now we consider the phases of $\Delta>U$. These phases occur first and we assume there are k such phases. During these k phases, all edge are of length 1 in the residual network $N(f)$ and the shortest residual network $N_{\bar\ell}(f)$ has no directed cycle. Thus, there is no strongly connected component with two or more vertices and no contracted vertex is generated. This implies that $N''_{\bar\ell}(f)=N_{\bar\ell}(f)$ which coincides with the level network $N_L(f)$ of Dinic. Let $F_1,F_2,\cdots,F_{k+1}$ and $\Delta_1,\Delta_2,\cdots,\Delta_{k+1}$ be the values F and Δ in the corresponding first $1,2,...,(k+1)$-st phases, respectively. Then $F_1\ge 2F_2\ge\cdots\ge 2^{k-1}F_k\ge 2^kF_{k+1}$ and $\Delta_i=\lceil\frac{F_i}{\min\{m^{1/2},n^{2/3}\}}\rceil$ $(i=1,2,...,k+1)$ and $\Delta_i>U$ $(i=1,2,...,k)$ and $\Delta_{k+1}\le U$. The maximum residual capacity M of edges of length 1 is at most $2U\le 2(\Delta_k-1)$ during these first k phases. Thus, if $\Delta_k=\lceil\frac{F_k}{m^{1/2}}\rceil$ (i.e., $m^{1/2}=\min\{m^{1/2},n^{2/3}\}$), then, after $4m^{1/2}$ times of updates of using a blocking flow, we have $d_\ell(s)\ge 4m^{1/2}$ and

$$cap_f(S)\le\frac{mM}{d_\ell(s)}\le\frac{2m(\Delta_k-1)}{4m^{1/2}}\le\frac{2mF_k}{4m^{1/2}m^{1/2}}=\frac{F_k}{2}.$$

Moreover, we have $M\le\frac{2F_k}{m^{1/2}}\le\frac{2F_{k-1}}{2m^{1/2}}\le\cdots\le\frac{2F_i}{2^{k-i}m^{1/2}}$ by $\Delta_i=\lceil\frac{F_i}{m^{1/2}}\rceil$ $(i=1,2,...,k)$. Thus, the number of updates using a blocking flow is at most $\frac{4m^{1/2}}{2^{k-i}}$ in the i-th phase since

$$cap_f(S)\le\frac{mM}{d_\ell(s)}\le\frac{2mM}{\frac{4m^{1/2}}{2^{k-i}}}\le\frac{\frac{2mF_i}{2^{k-i}m^{1/2}}}{\frac{4m^{1/2}}{2^{k-i}}}=\frac{F_i}{2}$$

after $\frac{4m^{1/2}}{2^{k-i}}$ times of updates of using a blocking flow, and the total number of updates using a blocking flow in these k phases is at most $8m^{1/2}$.

Similarly we can obtain that the total number of updates using a blocking flow in these k phases is at most $4n^{2/3}$ in the case when $\Delta_k=\lceil\frac{F_k}{n^{2/3}}\rceil$ $(n^{2/3}=\min\{m^{1/2},n^{2/3}\})$. Thus, total time required during the phases of $\Delta>U$ is $O(m\min\{m^{1/2},n^{2/3}\}\log(n^2/m))$.

By the argument above we have the following theorem.

Theorem 3.6 *[23] A maximum flow of an integral capacity network $N=(G,cap,s,t)$ can be found in $O(m\min\{m^{1/2},n^{2/3}\}\log(n^2/m)\log U)$ time by the Goldberg-Rao algorithm.*

4. Maximum Flow Algorithms for Undirected Unit Capacity Networks

In this section, we consider maximum flow algorithms for undirected unit capacity networks $N = (G = (V, E), cap, s, t)$ ($cap(e) = 1$ for each $e \in E$) with no parallel edges. We call such a network a simple undirected graph. Karzanov [36] and Even-Tarjan [18] have shown independently that the Dinic blocking flow algorithm runs in $O(\min\{m^{1/2}, n^{2/3}\}m)$ time on simple undirected graphs (Theorem 2.11). Until recently, this is the fastest algorithm. On the other hand, based on the sparsification technique by Nagamochi-Ibaraki [41], Goldberg-Rao [24] obtained an $O(\min\{m, n^{3/2}\}m^{1/2})$ time algorithm. They used the Nagamochi-Ibaraki sparsification technique in the context of residual graphs (networks) of flows in undirected graphs, which are not symmetric (the original Nagamochi-Ibaraki sparsification technique applies to undirected (e.g., symmetric) graphs). Karger-Levine [35] also used this technique and proposed an $O(m+n\nu^{3/2})$ time algorithm, where ν is the value of a maximum flow on a simple undirected graph. They also proposed an algorithm with $O(nm^{2/3}\nu^{1/6})$ time and a randomized algorithm with $O^*(m + n^{11/9}\nu)$ time. The latter algorithm suggests that the maximum flow problem on simple undirected graphs seems easier than the maximum bipartite matching problem.

In this section, we give a brief overview of these two algorithms on simple undirected graphs proposed by Goldberg-Rao and Karger-Levine based on the properties of distance labelings and the Nagamochi-Ibaraki sparsification. The feature of their method is as follows:

(i) The number of edges used in a flow of value r is bounded above by $3n\sqrt{r}$;

(ii) The number of edges in the residual network is bounded by $O(n\nu)$.

We first explain these two points. We begin with (i). Since we treat each edge $e = (u, v)$ of an undirected graph as two directed edges $e^B = \{e' = (u, v), e'' = (v, u)\}$ of the corresponding directed graph, we consider, for a flow f, the directed edge set $E_f = \{e \in E \mid f(e) = 1\}$ (f can be represented by $val(f)$ edge-disjoint paths from s to t and the direction of an edge $e \in E_f$ coincides with the direction of the path from s to t containing e) and undirected edge set $E_0 = E - E_f = \{e \in E \mid f(e) = 0\}$ with no flow. Thus, the residual network $N(f)$ of flow f consists of the edge set E_f^R of reverse edges e^R with residual capacity 2 corresponding to directed edges e in E_f and of the edge set E_0^B of two edge sets e^B each with capacity 1 corresponding to undirected edges e in E_0 (i.e., $E(f) = E_f^R \cup E_0^B$). For the directed edge set E_f, we have the following lemma similar to Lemma 2.10.

Lemma 4.1 *In an undirected simple graph (undirected simple unit capacity network) N, there is a flow f of value r satisfying $|E_f| \leq 3n\sqrt{r}$. Actually, if E_f contains no directed cycle then $|E_f| \leq 3n\sqrt{r}$.*

Proof. Let f be a flow of value r such that E_f contains no directed cycle. Then the network E_f contains the unique maximum flow f. Actually, if f' is a maximum flow of value r in E_f, then $f'' = f - f'$ is a flow of value 0 in E_f. If $E_{f''} = \{e \in E_f \mid f''(e) = 1\}$ were not an empty set, then each connected component of $E_{f''} \subset E_f$ would be a directed eulerian graph and E_f would contain a directed cycle. This is a contradiction. The remaining parts can be shown by the argument as in Proof of Lemma 2.10 (note that all edges in the residual network have length 1, i.e., ℓ is a unit length function). Only one point to consider is that the maximum edge capacity β is equal to 2.

Now we tried to find the unique maximum flow f in E_f by first setting $f' := 0$ and augment the current flow f' along a shortest path from s to t in the residual network $E_f(f')$ of f' (finally obtained f' is f). In the residual network $E_f(f')$ of a flow f' of value r', a maximum residual flow $f - f'$ is of value $r - r' = val(f) - val(f') \leq 2(\frac{n}{d_\ell(s)})^2$. Thus, the length $d_\ell(s)$ of a shortest path satisfies $d_\ell(s) \leq \frac{\sqrt{2}n}{\sqrt{r-r'}}$. This implies that the number of the newly used edges in the r'-th update step of increasing flow value r' to $r'+1$ is at most $\frac{\sqrt{2}n}{\sqrt{r-r'}}$ and we have $|E_f| \leq \sum_{r'=0}^{r-1} \frac{\sqrt{2}n}{\sqrt{r-r'}} \leq 2\sqrt{2}n\sqrt{r} < 3n\sqrt{r}$. □

We do not have to use all undirected edges of $N(f)$ to search an augmenting path. Actually, if we use a maximal spanning forest T_1 in $G - E_f = (V, E_0)$, then we have the following: $N(f)$ has an augmenting path if and only if the subnetwork $E_f^R \cup T_1^B$ of $N(f)$ has an augmenting path. If we use the result of Nagamochi-Ibaraki [41], this can be further generalized. Let $T_1, T_2, ..., T_m$ be defined recursively as follows. T_1 is a maximal spanning forest of $G^1 = G - E_f$ and T_{k+1} is a maximal spanning forest of $G^{k+1} = G^k - T_k = G - (E_f \cup T_1 \cup \cdots \cup T_k)$ for each $k = 1, 2, ..., m-1$. $T_1 \cup \cdots \cup T_m$ is called a *sparse connectivity certificate* of $G - E_f$ and $T_1 \cup \cdots \cup T_k$ is called a *sparse k-certificate* of $G - E_f$. Then we have the following lemma.

Lemma 4.2 *[24] For each $i \leq k$, $N^k = E_f^R \cup T_1^B \cup \cdots \cup T_k^B$ has i augmenting paths simultaneously (flow of value i) if and only if $N(f)$ has i augmenting paths simultaneously (flow of value i).*

Proof. If N^k has i augmenting paths simultaneously then $N(f)$ has i augmenting paths simultaneously since N^k is a subnetwork of $N(f)$. We consider the converse. Suppose that $N(f)$ has i augmenting paths but N^k does not have i augmenting paths. Then, by the maximum-flow, minimum-cut theorem, there is an s-t cut $\delta^+(S)$ ($s \in S, t \in V - S$) such that its capacity is at most $i-1$ in N^k and at least i in $N(f)$. Thus, there is an edge $a = (u, v)$ ($u \in S$, $v \in V - S$) such that a is in $N(f)$ but not in N^k. Of course, $a \notin E_f^R$. Thus, a is an undirected edge with $a \in E_0 = E - E_f$. Since there are at most $i-1$ undirected edges in $N^k - E_f^R$ joining S and $V - S$, one of $T_1, T_2, ..., T_k$, say T_j, has no such edge joining S and $V - S$. Thus, $T_j \cup \{a\}$ is a spanning forest by $a \notin T_1 \cup \cdots \cup T_k$. This, however, contradicts that T_j is a maximal spanning forest in $G^j = G^{j-1} - T_{j-1}$. Thus, if $N(f)$ has i augmenting paths then N^k has i augmenting paths. □

Before going to the Goldberg-Rao and Karger-Levine algorithms, we briefly describe the Nagamochi-Ibaraki algorithm [41] which computes a sparse connectivity certificate of an undirected connected graph $G = (V, E)$ in $O(m)$ time based on a kind of the breadth-first search with some priority.

Nagamochi-Ibaraki algorithm for finding a sparse connectivity certificate

```
1. All the edges $e \in E$ are initialized to be unscanned.
   Set $r[v] := 0$ for each $v \in V$ and *insert* $v$ into priority queue $Q$ as key $r[v]$.
2. **while** $Q$ is not empty **do**
   (a) $v :=$ *deletemax* (Delete vertex $v$ of maximum $r[v]$ from $Q$).
   (b) **for** each unscanned edge $e = (v, u)$ incident to $v$ **do** ($e$ becomes scanned).
           Set $t[e] := r[u] + 1$; $r[u] := r[u] + 1$ (*increasekey* of $u$ by one in $Q$).
           {**comment:** If $r[v] < r[u]$ (i.e., $r[u] = r[v] + 1$) then set $r[v] := r[u]$.}
```

The example of this algorithm is shown in Figure 9. Since $r[v]$ takes an integer in $[0, n]$ for a simple graph (and $[0, m]$ for a multigraph), the priority queue Q can be implemented to manipulate each operation (insert, deletemax, increasekey by one ($r[u] := r[u] + 1$)) in $O(1)$ time, and thus the algorithm runs in $O(m)$ time. In the algorithm, $t[e]$ indicates that e is in the maximal spanning forest $T_{t[e]}$. At any time of the algorithm, for any $u, w \in Q$ and for any positive integer $j \leq \min\{r[u], r[w]\}$, there is a path between u and w consisting of only scanned edges e of $t[e] = j$. Furthermore, $u \in Q$ is not incident to a scanned edge e with $t[e] > r[u]$. The comment in 2(b) will make it easier to show these properties. Thus, the algorithm correctly computes a sparse connectivity certificate $T_1 \cup \cdots \cup T_m$ of $G = (V, E)$ in $O(m)$ time. For details, see [41].

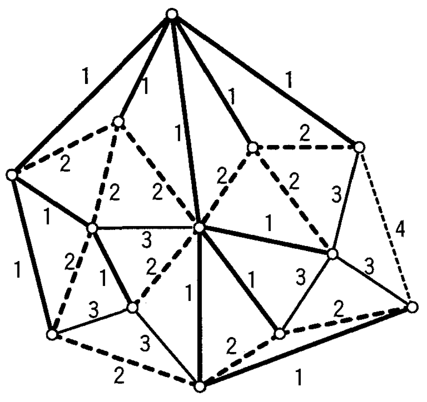

Figure 9: A simple graph $G = (V, E)$ and a connectivity certificate of G ($t[e]$ indicates that edge e is in $T_{t[e]}$).

Now we are ready to describe the Goldberg-Rao and Karger-Levine algorithms.

4.1. Goldberg-Rao algorithm for simple undirected graphs

The Goldberg-Rao algorithm is quite simple and only repeats the SparsifyAndBlock step. Initially $H^0 := N$, $f := 0$ and $H'^1 := H^0$. For convenience, we consider H^i is a directed graph and H'^i is an undirected graph ($i = 0, 1, 2, ...$). The SparsifyAndBlock step consists of the sparse connectivity certificate computation step and the blocking flow computation step. The step first applies the procedure Sparsify(H^i, f, k) ($i = 1, 2, ...,$ $k = (2n/d_\ell(s))^2$ and ℓ is a unit length function of H'^i) to obtain a sparse k-certificate $T_1, T_2, ..., T_k$ of $H'^i - E_f$ and sets $H^i := E_f^R \cup T_1^B \cup T_2^B \cup \cdots \cup T_k^B$ (Sparsify step). Then the step augments f to $f' := f + f''$ using a blocking flow f'' on the level network H_L^i of H^i based on the method of Dinic and sets $H'^{i+1} := H^i(f')$ and $f := f'$ (Block step). Note that if we neglect the sparse connectivity certificate computation step in the SparsifyAndBlock step, then the algorithm coincides with the Dinic algorithm.

Since obtaining H'^{i+1} from H^i in the Block step takes $O(m_i + n)$ time and obtaining H^{i+1} from H'^{i+1} in the Sparsify step also takes $O(m'_{i+1} + n)$ time and $m'_{i+1} = O(m_i)$, we can consider the dominating step of the algorithm is the Block step (m'_i is the number of edges in H'^i and m_i is the number of edges in H^i). Thus, we have only to consider the Block step in the analysis of the efficiency of algorithm below.

Theorem 4.3 *[24] The algorithm runs in time* $O(\min\{m, n^{3/2}\}m^{1/2})$.

Proof. If $m \leq n^{5/3}$, then $n^{2/3}m \leq n^{3/2}m^{1/2}$ and $\min\{m^{1/2}, n^{2/3}\}m \leq \min\{m, n^{3/2}\}m^{1/2}$. Thus, by Theorem 2.11, the running time of the algorithm is $O(\min\{m^{1/2}, n^{2/3}\}m)$, which is $O(\min\{m, n^{3/2}\}m^{1/2})$. For the rest of the proof, we assume $m = \Omega(n^{5/3})$.

During the initial iterations of the algorithm, when $n(2n/d_\ell(s))^2 > m$, the sparsification has no effect and each iteration takes $O(m)$ time. During these iterations, $d_\ell(s) < 2n^{3/2}/m^{1/2}$ where ℓ is a unit length function. Since $d_\ell(s)$ increases at each iteration, these iterations take a total of $O(n^{3/2}m^{1/2})$ time.

During (the i-th iteration of) the final iterations of the algorithm, when $n(2n/d_\ell(s))^2 \leq 4n^{3/2}$ ($d_\ell(s) \geq n^{3/4}$), we have $m_i \leq 4n^{3/2} + 3n^{3/2} = O(n^{3/2})$ since Sparsify(H^i, f, k) with $k = (2n/d_\ell(s))^2$ produces at most nk edges and $|E_f| \leq 3n^{3/2}$ by Lemma 4.1. Thus, by Lemma 2.9 and the argument above, these iterations take $O(n^2)$ time. This is $O(n^{3/2}m^{1/2})$

for $m \geq n$.

Finally we account for the remaining iterations $(2n^{3/2}/m^{1/2} \leq d_\ell(s) < n^{3/4})$. Each i-th iteration takes $O(m_i)$ time. If $d_\ell(s) = j$ during the i-th iteration, then $j \geq i$ by Lemma 2.7 and

$$m_i \leq n(2n/j)^2 + 3n^{3/2} \leq 2n\left(\frac{2n}{j}\right)^2,$$

since Sparsify$(H^i, f, (2n/j)^2)$ produces at most $n(2n/j)^2$ edges and $|E_f| \leq 3n^{3/2}$ by Lemma 4.1. Let $D_0 = 2n^{3/2}/m^{1/2}$. Then the total work during these iterations is at most

$$\sum_{i=D_0}^{\infty} 2n(2n/i)^2 = 8n^3 \sum_{i=D_0}^{\infty} \frac{1}{i^2} \leq 8n^3\left(\frac{1}{D_0^2} + \frac{1}{D_0}\right) = O(n^{3/2}m^{1/2}).$$

Thus, we have that the total work is $O(n^{3/2}m^{1/2})$, which is $O(\min\{m, n^{3/2}\}m^{1/2})$ for $m = \Omega(n^{5/3})$. □

4.2. Karger-Levine algorithm for simple undirected graphs

In this section we describe the Karger-Levine algorithms for finding a maximum flow of an undirected simple graph. They find a maximum flow in $O(m + n\nu^{3/2})$ and $O(nm^{2/3}\nu^{1/6})$ time (ν is the value of maximum flow) [35]. The key point is that they devised a method to find an augmenting path in $O(n\sqrt{\nu})$ time in an amortized sense. Now we are ready to describe the Karger-Levine algorithms.

The first algorithm initially sets $f := 0$ and then calls SparseAugment2(N, f) below. Using sparse k-certificates, it finds a flow of value $r \leq \nu$ in $O(m + r(n\sqrt{\nu} + \sqrt{mn}))$ time if we stop just after a flow of value r is obtained. Note that the flow obtained may contain a directed cycle since they augment a flow not using a shortest path. Thus, Karger-Levine included a step of deleting such cycles. This step can be done by a kind of depth-first search and finds a flow of the same value with no directed cycle in time proportional to the number of edges in the old flow.

```
SparseAugment2(N, f)
  1. Set $k := \lceil \sqrt{m/n}\, \rceil$.
  2. Repeat the following steps as many as possible.
     (a) Delete a cycle in $E_f$ (we use the same notation $E_f$ after this step).
     (b) Find a sparse $k$-certificate $T_1 \cup \cdots \cup T_k$ of $G - E_f$.
     (c) Set $N^k := E_f^R \cup T_1^B \cup \cdots \cup T_k^B$.
     (d) Find a set of $k$ augmenting paths in $N^k$ or a set of as many augmenting paths
         as possible if $k$ augmenting paths are not in $N^k$, and set it $f''$ and update $f$ to
         $f := f + f''$.
     (e) If $f''$ does not contain $k$ augmenting paths, return $f$ and terminate.
```

The time complexity of this algorithm is $O(m+r(n\sqrt{\nu}+\sqrt{mn}))$ if we stop just after a flow of value at least r is obtained. Actually, deleting cycles and finding a sparse k-certificate can be done in $O(m)$ time [41]. The time required to find k augmenting paths is $O(m'k)$ where m' is the number of edges in N^k. By the definition of a sparse k-certificate and by Lemmas 4.1 and 4.2, we have $m' \leq nk + 3n\sqrt{\nu}$. Since the number of iterations is $\lceil r/k \rceil$ to find a flow of value at least r, the time complexity is $O((m + m'k)\lceil r/k \rceil) = O(m + r(n\sqrt{\nu} + \sqrt{mn}))$ $(m\lceil r/k \rceil = O(r\sqrt{mn}))$. Thus, if $n\nu \geq m$ (i.e., $n\sqrt{\nu} \geq \sqrt{mn}$) then this algorithm finds an augmenting path in $O(n\sqrt{\nu})$ time in an amortized sense. If we can find ν (or an approximate value ν' of ν with $\frac{\nu}{2} \leq \nu' \leq 2\nu$) efficiently (about in $O^*(m + n\nu)$ time) [7], then even if

$n\nu \leq m$ this algorithm can be modified to find an augmenting path in $O(n\sqrt{\nu})$ time in an amortized sense. Such a method is described below. Here, N_w is the set of first w forests of a sparse connectivity certificate of $G - E_f$. Thus, N_w is a sparse w-certificate of $G - E_f$. $|N_w|$ denotes the number of edges in N_w. The algorithm initially sets $f := 0$ and then calls SparseAugment3(N, f) below.

```
SparseAugment3(N, f)
  1. Find a sparse connectivity certificate of $G - E_f$ of $N$.
  2. Set $w := val(f)$.
  3. Repeat the following steps until $val(f) < w$.
     (a) Find the minimum number $w'$ satisfying $|N_{w'}| > 2|N_w|$. If such $w'$ exists then set
         $w := w'$. Otherwise, set $N_w := G - E_f$.
     (b) Perform SparseAugment2($N_w \cup E_f, f$).
  4. Return $f$ and terminate.
```

The time complexity of this algorithm is $O(m+rn\sqrt{\nu})$ if we stop just after a flow of value at least r is obtained. Actually, the first step can be done in $O(m)$ time. We denote by k the number of iterations. The time required in the i-th iteration is $O(m_i+r_i(n\sqrt{\nu}+\sqrt{m_i n}))$ where m_i is the number of edges in N_{w_i} (N_{w_i} denote N_w in the i-th iteration) and r_i is the number of augmenting paths found in the i-th iteration. In this time complexity, the first term m_i is set to be more than twice in each iteration (i.e., $2m_i < m_{i+1} \leq 2m_i + n$ for each $i = 1, 2, ..., k-2$ and $m_k \leq 2m_{k-1} + n$) and thus its total sum is $O(m)$. For the next term, $w_{k-1} \leq \nu$ since the $(k-1)$-st iteration is not the last iteration. Thus, $m_{k-1} \leq n\nu$ and we have $m_k \leq 2n\nu + n = O(n\nu)$. The total sum of r_i is r. Thus, the total sum of second term is $O(rn\sqrt{\nu})$ and the time complexity of the algorithm is $O(m + rn\sqrt{\nu})$. This implies that this algorithm finds an augmenting path in $O(n\sqrt{\nu})$ time in an amortized sense. If we set $r = \nu$ then the time complexity of this algorithm is $O(m + n\nu^{3/2})$ (if $\nu = \Theta(n)$ then it is $O(n^{5/2})$).

The algorithm below assumes that an approximate value ν' of ν with $\frac{\nu}{2} \leq \nu' \leq 2\nu$ is computed before in $O^*(m + n\nu)$ time.

```
BlockThenAugment(N, k)
  1. Set $f := 0$.
  2. Repeat finding a blocking flow $f''$ in the shortest residual network $N_\ell(f)$ of $N(f)$ and
     augmenting the current flow $f$ to $f := f+f''$ until $d_\ell(s) \geq k$ ($d_\ell(s)$ denotes the number
     of edges in a shortest path from $s$ to $t$ of $N(f)$).
  3. Return SparseAugment3($N, f$).
```

The time complexity of this algorithm is $O(nm^{2/3}\nu^{1/6})$ ($O(n^{2.5})$ in the worst case) if we set k to be $n\nu'^{1/6}/m^{1/3} = O(n\nu^{1/6}/m^{1/3})$. Actually, finding a blocking flow on $N_\ell(f)$ can be done in $O(m)$ time and it suffices to find at most k blocking flows by Lemma 2.7, which we can do it in $O(mk) = O(nm^{2/3}\nu^{1/6})$ time. At this point we have $d_\ell(s) \geq k$ and thus the residual flow is of value $O((n/k)^2)$ by Lemma 2.9. Note that the time complexity analyses for SparseAugment2(N, f) and SparseAugment3(N, f) can be modified to hold even when we augment a flow f of value r_0 to a flow f' of value r' with $r = r' - r_0$. Thus, SparseAugment3(N, f) requires $O(n^3\sqrt{\nu}/k^2) = O(nm^{2/3}\nu^{1/6})$ time and BlockThenAugment(N, k) also requires $O(n^3\sqrt{\nu}/k^2) = O(nm^{2/3}\nu^{1/6})$ time.

We will not present the randomized algorithm of Karger-Levine finding a maximum flow of a simple undirected graph in $O^*(m+n^{11/9}\nu)$ time [35] here, since it is rather complicated. We give only a remark on it: it is not applied to finding a maximum matching of a bipartite graph (see Figure 10).

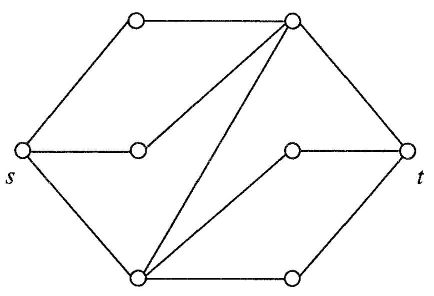

Figure 10: A network N with a maximum flow of value 3 and the bipartite graph obtained by deleting s and t which has a maximum matching of size 2.

5. Concluding Remarks

We have surveyed useful techniques for designing efficient algorithms for the maximum flow problem and presented representative maximum flow algorithms based on these techniques in a systematic way. We have given a focus on recently proposed algorithms by presenting detailed comments especially on the Goldberg-Rao algorithm since their original paper contains minor mistakes in the analysis [23], and described traditional algorithms including algorithms based on the preflow push-relabel method rather briefly. This is because nice survey papers are already published on these traditional algorithms not only in English but also in Japanese [46, 13, 25, 1, 2, 31, 29, 6].

The algorithms described in this survey are, however, mainly concerned with the theoretical efficiency in the worst case. Practical efficiencies of their algorithms were not thoroughly analyzed so far and implementations of representative algorithms have long been required. From this point of view, Ahuja-Magnanti-Orlin gave a nice survey on the practical efficiencies and implementations of representative algorithms. Imai [30] implemented representative algorithms proposed by 1980 including the Dinic algorithm, Karzanov algorithm, Malhotra-Kumar-Maheshwari algorithm [40], Galil-Naamad algorithm [21], and concluded that the Dinic algorithm was the fastest among the tested algorithms. Recent implementations of variants of push-relabel method based on heuristics and the Dinic algorithm by Cherkassky-Goldberg [12] showed that the push-relabel method based on the highest-label selection combined with both global and gap relabeling heuristics is the fastest among the algorithms described above. Goldberg told that he has implemented the Goldberg-Rao maximum flow algorithm based on the binary length function, but it is not as fast as the push-relabel method based on the highest-label selection combined with both global and gap relabeling heuristics. Further implementations of various maximum flow algorithms and the practical efficiencies evaluations on real world problem data are keenly desired.

The maximum flow problem has a wide variety of scientific and engineering applications as mentioned in Introduction. Actually, Ahuja-Magnanti-Orlin gave a detailed description on applications of maximum flow problem [1, 2]. We believe that a lot of interesting researches are developing in the maximum flow problem and network optimization not only from the theoretical point of view but also from practical applications.

Acknowledgments

The first author was supported in part by Grant in Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan and The Institute of Science and Engineering of Chuo University. The authors would like to thank anonymous editor and referees for their valuable comments improving the presentation.

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin: Network flows. In G. L. Nemhauser et al. (eds.): *Handbooks of Operations Research and Management Science, Vol. 1, Optimization* (North-Holland, Amsterdam, 1989), 211–369.

[2] R. K. Ahuja, T. L. Magnanti and J. B. Orlin: *Network Flows: Theory, Algorithms, and Applications* (Prentice-Hall, NJ, 1993).

[3] R. K. Ahuja and J. B. Orlin: A fast and simple algorithm for the maximum flow problem. *Operations Research*, **37** (1989) 748–759.

[4] R. K. Ahuja, J. B. Orlin and R. E. Tarjan: Improved time bounds for the maximum flow problem, *SIAM Journal on Computing*, **18** (1989) 939–954.

[5] N. Alon: Generating pseudo-random permutations and maximum flow algorithms. *Information Processing Letters*, **35** (1990) 201–204.

[6] T. Asano: *Jouhou no Kouzou (Information Structures)* (Nihon Hyouron Sha, Tokyo, 1994 (in Japanese)).

[7] A. A. Benczur and D. R. Karger: Approximate s-t min-cuts in $\tilde{O}(n^2)$ time. *Proc. 28th ACM Symposium on Theory of Computing*, (1996) 47–55.

[8] J. Cheriyan and T. Hagerup: A randomized maximum-flow algoithm. *Proc. 30th IEEE Symposium on Foundations of Computer Science*, (1989) 118–123, and *SIAM Journal on Computing*, **24** (1995) 203–226.

[9] J. Cheriyan, T. Hagerup and K. Mehlhorn: Can a maximum flow be computed in $o(nm)$ time? *Proc. 17th International Colloquium on Automata, Languages and Programming* (1990) (Lecture Notes in Computer Science 443, Springer-Verlag), 235–248.

[10] J. Cheriyan and S. N. Maheshwari: Analysis of preflow push algorithms for the maximum network flow. *SIAM Journal on Computing*, **18** (1989) 1057–1086.

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

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

[13] T. H. Cormen, C. E. Leiserson and R. L. Rivest: *Introduction to Algorithms* (MIT Press, Cambridge, 1990).

[14] G. B. Dantzig: Application of the simplex method to a transportation problem. In T. C. Koopman (ed.): *Activity Analysis and Production and Allocation* (Wiley, New York, 1951), 359–373.

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

[16] E. A. Dinic: Metod porazryadnogo sokrashcheniya nevyazok i transportnye zadachi (Excess scaling and transportation problems). In *Issledovaniya po Diskretnoĭ Matematike* (Nauka, Moskva,1973).

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

[18] S. Even and R. E. Tarjan: Network flow and testing graph connectivity, *SIAM Journal on Computing*, **4** (1975) 507–518.

[19] L. R. Ford, Jr. and D. R. Fulkerson: Maximal flow through a network. *Canad. J. Math.*, **8** (1956) 399–404.

[20] H. N. Gabow: Scaling algorithms for network problems. *Journal of Computer and System Sciences*, **31** (1985) 148–168.

[21] Z. Galil and A. Naamad: An $O(|E||V|\log^2|V|)$ algorithm for the maximum flow problem. *Journal of Computer and System Sciences*, **21** (1980) 203–217.

[22] A. V. Goldberg: A new max-flow algorithm. *Technical Report MIT/LCS/TM-291, Laboratory for Computer Science* (MIT, Cambridge, 1985).

[23] A. V. Goldberg and S. Rao: Beyond the flow decomposition barrier. *Proc. 38th IEEE Symposium Foundations of Computer Science*, (1997) 2–11, and *Journal of the ACM*, **45** (1998) 783–797.

[24] A. V. Goldberg and S. Rao, Flows in undirected unit capacity networks, *Proc. 38th IEEE Symposium on Foundations of Computer Science*, (1997) 32–34, and *SIAM Journal on Discrete Mathematics*, **12** (1999) 1–5.

[25] A. V. Goldberg, É. Tardos and R. E. Tarjan: Network flow algorithms. In B. Korte et al. (eds.): *Paths, Flows, and VLSI-Layout* (Springer-Verlag, Berlin, 1990), 101–164.

[26] A. V. Goldberg and R. E. Tarjan: A new approach to the maximum flow problem. *Journal of the ACM*, **35** (1988) 921–940.

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

[28] J. E. Hopcroft and R. M. Karp: An $n^{2.5}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, **2** (1973) 225–231.

[29] T. Ibaraki: *Risan Saitekikahou to Arugorizumu (Discrete Optimization and Algorithms)* (Iwanami Shoten, 1992 (in Japanese)).

[30] H. Imai: On the practical effiency of various maximum flow algorithms. *J. Oper. Res. Soc. Japan*, **26** (1983) 61–83.

[31] K. Iwano: Recent developments in the network flow problem. In S. Fujishige (ed.): *Discrete Structures and Algorithms (Risan Kouzou to Arugorizumu), Vol. 2* (Kindai Kagaku Sha, 1993 (in Japanese)), 79–153.

[32] D. R. Karger: Minimum cuts in near-linear time. *Proc. 28th ACM Symposium on Theory of Computing*, (1996) 56–63.

[33] D. R. Karger: Using random sampling to find maximum flows in uncapacitated undirected graphs. *Proc. 29th ACM Symposium on Theory of Computing*, (1997) 240–249.

[34] D. R. Karger: Better random sampling algorithm for flows in undirected graphs. *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, (1998) 490–499.

[35] D. R. Karger and M. S. Levine: Maximum flows in undirected graphs seems easier than bipartite matching. *Proc. 30th ACM Symposium on Theory of Computing*, (1998) 69–78.

[36] A. V. Karzanov: O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh (On finding maximum flows in network with special structure and some applications). In *Mathematicheskie Voprosy Upravleniya Proizvodstvom, Vol.5* (Moscow State University Press, Moscow, 1973).

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

[38] V. King, S. Rao and R. E. Tarjan: A faster deterministic maximum flow algorithm. *Proc. the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, (1992) 157–163.

[39] V. King, S. Rao and R. E. Tarjan: A faster deterministic maximum flow algorithm. *Journal of Algorithms*, **17** (1994) 447–474.

[40] V. Malhotra, P. Kumar and S. Maheshwari: An $O(|V|^3)$ algorithm for finding maximum flows in network. *Information Processing Letters*, **7** (1978) 277–278.

[41] H. Nagamochi and T. Ibaraki: A linear time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, **7** (1992) 583–596.

[42] S. Phillips and J. Westbrook: Online load balancing and network flow. *Proc. 25th ACM Symposium on Theory of Computing*, (1993) 402–411, and *Algorithmica*, **21** (1998) 245–261.

[43] Y. Shiloach: An $O(n \cdot I \log^2 I)$ Maximum-Flow Algorithm. *Technical Report STAN-CS-78-802* (Computer Science Department, Stanford University, CA, 1978).

[44] D. D. Sleator and R. E. Tarjan: A data structure for dynamic trees. *Journal of Computer and System Sciences*, **16** (1983) 362–391.

[45] T. Sunaga and M. Iri: Theory of communication and transportation networks. *RAAG Memories*, **2** (1958) 444–468.

[46] R. E. Tarjan: *Data Structures and Network Algorithms* (SIAM, Philadelphia, 1983).

Takao Asano
Department of Information and System Engineering
Chuo University
Kasuga 1-13-27, Bunkyo-ku, Tokyo 112-8551, Japan
Email: asano@ise.chuo-u.ac.jp