Journal of the Operations Research
Society of Japan
2004, Vol. 47, No. 4, 244-264

A SURVEY OF COMBINATORIAL MAXIMUM FLOW ALGORITHMS ON A NETWORK WITH GAINS

Maiko Shigeno
University of Tsukuba

(Received September 29, 2003; Revised May 6, 2004)

Abstract Network optimization experienced a fast development, during the last few decades. While a traditional network assumes that each flow is conserved on every arc, a flow along an arc may increase or decrease by a fixed factor during its traversal of that arc in a generalized network. This paper surveys combinatorial maximum flow algorithms on the generalized network and compares algorithms for traditional network flows.

Keywords: Network flow, generalized maximum flow, algorithm

1. Introduction

Network flows are of growing interest from the point of view of both applications and theory. The topic of network flows has applications in such diverse fields as engineering, management science, computer science, urban traffic, communications, and economics to name but a few. Related to theory, network flows have been proven to be an excellent indicator of things to come in mathematical programming. Spurred by Ford-Fulkerson's seminal work [15] in the 60s, studies in this area have led to continued improvements in efficient network flow algorithms. This is especially in respect to polynomial time algorithms for shortest paths, maximum flows, and minimum cost flows that underwent intensive development in 80s to 90s. There have been a number of survey papers and books that have reported on network flow algorithms. For example, [2, 4, 6, 8, 37, 40, 53, 54] have been published in the last decade.

However, although interest on variants in flows such as generalized flows, multicommodity flows, and dynamic flows, there have been few reviews on such flow models. For example Goldberg-Tardos-Tarjan [24] and Wayne [61] reported on a maximum flow problem on a generalized network. This paper gives an overview of historical developments and recent algorithmic results for a generalized maximum flow problem.

In a generalized network model, each arc (i, j) of the network has a positive multiplier $\gamma(i, j)$, which implies that if we send one unit from the node i to the node j along the arc (i, j), then $\gamma(i, j)$ units arrive at the node j. The multiplier is called a gain. This network model arises in manufacturing, transportation, communication and financial analysis [2, 19, 21, 61]. A problem finding a maximum flow in the generalized network is called a generalized maximum flow problem.

Since Dantzig [9] and Jewell [36] did their first studies in the 1960s, a number of generalized maximum flow algorithms have been developed. The problem itself is a special case of the linear programming problem. Consequently, Dantzig [9] proposed a generalized network simplex algorithm, which is a specialization of the simplex method. Recently, a polynomial-time dual simplex algorithm has been proposed by Goldfarb-Jin-Lin [31] that is

based on the combinatorial algorithm by Goldfarb-Jin-Orlin [32].

Kapoor-Vaidya [38] proposed an $O(m^{1.5}n^2 \log B)$ time algorithm that is based on Karmarkar's interior-point method. Here n is the number of nodes, m is the number of arcs, and B is the largest integer in the representations of capacities and gain multipliers, assuming that these numbers are given as the ratios of two integers. Murray [41] also designed an interior-point algorithm for generalized flows. Goldfarb-Lin [33] presented an interior-point algorithm with a combinatorial method, i.e., flow augmentations. These interior-point algorithms can be applied to generalized minimum cost flows.

A number of combinatorial algorithms for the generalized maximum flows have also been developed. In the 60s, Jewell [36] and Onaga [44] independently proposed the first combinatorial algorithms for generalized maximum flows. In 1991, Goldberg-Plotkin-Tardos [23] designed the first polynomial-time combinatorial algorithms. Since then, many researchers have been investigating efficient combinatorial algorithms. These algorithms will be discussed in Section 4. The current best time complexity of a combinatorial algorithm is $O(nm(m + n \log n) \log B)$ due to Radzik [50].

From algorithmic aspect, interesting problems on generalized maximum flows are:

1. to develop a strongly polynomial time algorithm,
2. to develop a combinatorial algorithm faster than the $O(m^{1.5}n^2 \log B)$ interior point algorithm when $m = \Theta(n^2)$, and
3. to develop a combinatorial algorithm that works well in practice.

The generalized maximum flow problem is one of the simplest linear programming problems for which no strongly polynomial algorithm is known. Many researchers believe that a combinatorial approach is more likely to lead to a strongly polynomial algorithm, because, for simpler network flow problems (e.g., the maximum flow problem and the minimum cost flow problem), a combinatorial approach has succeeded in attaining the strongly polynomial time bounds. The combinatorial approach also leads to algorithms whose running times are clearly superior to the running time bounds of the algorithms based on general-purpose linear programming methods, in the simpler network flow problems. Moreover, for an approximate generalized network flow problem, the combinatorial approach solves in strongly polynomial time, although we do not know about any algorithm for the approximate generalized network flow problem which is based on a general linear programming method and has the running time better than one for the exact generalized flow problem. Hence, the development of the faster combinatorial algorithm for generalized maximum flows fascinates us. Another research direction is to design an algorithm which runs practically fast. Radzik-Yang [51] investigated whether some polynomial-time combinatorial algorithms for the generalized maximum flows can be led to practically efficient implementations. In their result, a commercial linear programming package performed better than their implementations of combinatorial algorithms, but on large networks its dominance was small enough to believe that combinatorial algorithms, most likely in combination with the ideas underlying general-purpose linear programming methods, will lead to the fastest implementations.

This paper reviews combinatorial algorithms for generalized maximum flows. A generalized maximum flow problem is a natural extension of a (no gain) maximum flow problem. Therefore, many generalized maximum flow algorithms are related to maximum flow algorithms. Moreover, there is a relationship between the generalized maximum flow problem and the minimum cost flow problem as we will discuss in Section 2.3. Truemper [60] compared classical algorithms for generalized maximum flows and minimum cost flows. Similarly, we will review recent generalized maximum flow algorithms and compare them with maximum flow and minimum cost flow algorithms. This paper will also highlight the

main ideas involved in developing efficient algorithms, i.e., we discuss a technique using arc excesses, and will present a brief analysis for a flow-generating cycle-canceling procedure, because we believe that it holds clues on how to develop a strongly polynomial time algorithm.

This paper is organized as follows. We first define a generalized maximum flow problem and review fundamental facts about it in Section 2. Section 3 discusses a framework that has been adopted in many approaches to combinatorial algorithms. We present a flow-generating cycle-canceling procedure, which is a preprocessing. Section 4 reviews combinatorial algorithms and compares maximum flow algorithms and minimum cost flow algorithms. We describe a most-helpful augmenting path algorithm, which is not faster than existing algorithms but which is naturally derived from the development stream of efficient algorithms.

2. Generalized Maximum Flow Problem

2.1. Problem formulation

Let $G = (N, F)$ be a directed graph with the set of n nodes N and the set of m arcs F. For notational convenience, let us assume that G has at most one arc between any pair of nodes so that each arc can be uniquely specified by its endpoints. The graph G has a distinguished node t called a *sink*. On the arcs, a nonnegative *capacity function* $u : F \to \mathbb{R}_+$ and a positive *gain function* $\gamma : F \to \mathbb{R}_{++}$ are given. A gain $\gamma(i, j)$ of an arc (i, j) implies that, if x units of flow enter the arc (i, j) at its tail node i, then $\gamma(i, j) \cdot x$ units arrive at its head node j. For a function $g : F \to \mathbb{R}$, we define an *excess* function $e_g : N \to \mathbb{R}$ by

$$e_g(i) := -\sum_{\{j \in N \mid (i,j) \in F\}} g(i, j) + \sum_{\{j \in N \mid (j,i) \in F\}} \gamma(j, i) g(j, i),$$

which means the net flow into a node i. A *generalized flow* is a function $g : F \to \mathbb{R}$ that satisfies the following conditions:

capacity constraint: $0 \le g(i, j) \le u(i, j), \quad \forall (i, j) \in F, \quad$ and
flow conservation constraint: $e_g(i) = 0, \quad \forall i \in N \setminus \{t\}$.

In contrast to a generalized flow, we say a "traditional" flow when all gains are given by ones. A function g satisfying only the capacity constraint is called a *pseudo flow*.

The *flow value* with respect to a flow g is given by the sink's excess $e_g(t)$. The generalized maximum flow problem is to find a generalized flow g maximizing its flow value. It is formulated as

$$\text{(GMF)} \quad \max\{e_g(t) \mid 0 \le g \le u, e_g(i) = 0 \text{ for all } i \in N \setminus \{t\}\}.$$

The traditional maximum flow problem is a special case of (GMF), where a special node s called a source has a loop with $\gamma(s, s) > 1$ and $u(s, s) = \infty$, and $\gamma(i, j) = 1$ holds for all arcs (i, j) but (s, s).

There is some literature on the generalized maximum flow problem that has used the following definition:

$$\text{(GMF}') \quad \max\{e_g(t) \mid 0 \le g \le u, e^0(i) \ge -e_g(i) \text{ for all } i \in N \setminus \{t\}\},$$

where $e^0(i)$ is an initial excess of a node i. We assume that, as usual, the initial excess is given by a nonnegative number. For convenience, we define a function $\hat{e}_g : N \to \mathbb{R}$ as

$$\hat{e}_g(i) := e^0(i) + e_g(i).$$

We will discuss the relation between (GMF) and (GMF′) later.

Generalized flows can successfully model many application settings that cannot adequately be captured by a traditional network representation. The gain factors can represent physical transformations of one commodity into a lesser or greater amount of the same commodity. This interpretation allows us to model processes such as money expressed in terms of purchasing power, perishable goods held in inventory, electrical power carried on transmission lines, and so on. The gain factors can also model the transformation of one commodity into a different commodity, involving the transformation of flow from one unit of measure to another. Some examples include converting raw material into finished goods, currency conversion, and machine scheduling. For specific applications see [2, 19, 21, 22].

Throughout this paper, it has been assumed that each capacity is given by an integer and each gain is given by a ratio of two integers. We denote the largest integer used to represent capacities and gains by B.

2.2. Notations and properties

For a path (or cycle) P, the gain of it is given by the product of the gains of arcs participating in that path (or cycle), i.e., $\prod_{(i,j)\in P}\gamma(i,j)$, and denoted by $\gamma(P)$. Simple cycles whose gains are exactly one, more than one, and less than one are called a *unit cycle*, *flow-generating cycle* and *flow-absorbing cycle*, respectively.

The flow decomposition theorem is a fundamental property. We first define *elementary flows*. An elementary flow g is a pseudo flow satisfying one of the following five types, where $F^g = \{(i,j) \in F \mid g(i,j) > 0\}$.

(Type 1) F^g constitutes a simple path from a node with negative excess to a node with positive excess. Each node, except both end nodes of the path, has zero excess.

(Type 2) F^g constitutes a unit cycle. All nodes have zero excess.

(Type 3) F^g constitutes a flow-generating cycle and a path from a node in the cycle to a node with positive excess. Each node, except the tail node of the path, has zero excess.

(Type 4) F^g constitutes a flow-absorbing cycle and a path from a node with negative excess to a node in the cycle. Each node, except the head node of the path, has zero excess.

(Type 5) F^g constitutes a flow-generating cycle, a flow-absorbing cycle, and a path from the first cycle to the second. All nodes have zero excess.

An empty path is possible in Types 3, 4 and 5. Then the flow decomposition theorem is given as follows.

Theorem 2.1 ([23, 34]) *Any pseudo flow g can represent a sum of elementary flows $g^1, g^2, \ldots, g^k$, i.e., $g = \sum_{h=1}^{k} g^h$, where $k \le m$ and*

$$g^h(i,j) > 0 \Longrightarrow g(i,j) > 0 \quad \forall (i,j) \in F$$

$$e_{g^h}(i) > 0 \Longrightarrow e_g(i) > 0 \ ; \ \textit{and} \quad e_{g^h}(i) < 0 \Longrightarrow e_g(i) < 0 \quad \forall i \in N$$

hold for all h. ∎

When all gains are equal to one, there is neither a flow-generating cycle nor a flow-absorbing cycle. In this case, any elementary flow constructs a simple path or a simple cycle. Thus, the above theorem is a generalization of the traditional flow decomposition theorem.

Let us next consider optimality conditions of the generalized maximum flow problem. Let $\overleftarrow{F} := \{(j,i) \mid (i,j) \in F\}$ denote the set of reverse arcs and $A := F \cup \overleftarrow{F}$. An arc $(j,i) \in \overleftarrow{F}$ represents the possibility of pushing flow back on the arc (i,j). For each $(j,i) \in \overleftarrow{F}$, we define $\gamma(j,i) := 1/\gamma(i,j)$. With a pseudo flow g, we associate a *residual graph* $G^g = (N, A^g)$, where

$A^g = \{(i,j) \in F \mid g(i,j) < u(i,j)\} \cup \{(j,i) \in \overleftarrow{F} \mid g(i,j) > 0\}$. A residual capacity function $u_g : A^g \to \mathbb{R}$ is given by

$$u_g(i,j) := \begin{cases} u(i,j) - g(i,j) & (i,j) \in F, \\ \gamma(j,i)g(j,i) & (i,j) \in \overleftarrow{F}. \end{cases}$$

Theorem 2.2 ([35, 45]) *1. Let $g : F \to \mathbb{R}$ be a generalized flow. The following statements are then equivalent:*

(a) *g is an optimal flow to (GMF).*

(b) *The residual network $G^g = (N, A^g)$ does not contain any flow-generating cycle from which the sink t is reachable.*

(c) *There exists $\mu : N \to \mathbb{R}_{++} \cup \{\infty\}$ such that $\mu(t) = 1$, and*

$$\gamma(i,j)\mu(i) \leq \mu(j) \tag{2.1}$$

holds for every $(i,j) \in A^g$.

2. Let $g' : F \to \mathbb{R}$ be a feasible flow of (GMF'). The following statements are then equivalent:

(a') *g' is an optimal flow to (GMF').*

(b') *The residual network $G^{g'} = (N, A^{g'})$ does not contain any path from a node i with $\hat{e}_{g'}(i) > 0$ to the sink t or any flow-generating cycle from which t is reachable.*

(c') *There exists $\mu : N \to \mathbb{R}_{++} \cup \{\infty\}$ such that $\mu(t) = 1$, $\mu(i) = \infty$ for all nodes i with $\hat{e}_{g'}(i) > 0$, and Eq.(2.1) holds for every $(i,j) \in A^{g'}$.*

Proof. The relation between (a) and (b) (resp. (a') and (b')) is derived from the flow decomposition theorem. Conditions (c) and (c') originate from the primal-dual theorem of linear programming, where μ is the reciprocal of a dual variable. ∎

Let us refer to a function $\mu : N \to \mathbb{R}_{++} \cup \{\infty\}$ with $\mu(t) = 1$ as a *label*. A label $\mu(i)$ can be interpreted as a local flow unit in a node i. To use the new unit of measurement, the capacity, gain, and initial excess must be normalized by the label μ. The operation of normalizing is called *relabeled*. With respect to a label μ, the relabeled capacity and relabeled gain are given by

$$u^\mu(i,j) := u(i,j)/\mu(i), \quad \text{and} \quad \gamma^\mu(i,j) := \gamma(i,j)\mu(i)/\mu(j),$$

respectively, for each arc (i,j). The relabeled initial excess is given by $e^0(i)/\mu(i)$. Any flow g is also relabeled as

$$g^\mu(i,j) := g(i,j)/\mu(i).$$

Note that, if g is a feasible flow in the original network $(G = (N,F), u, \gamma)$, then the relabeled flow g^μ with respect to a label μ is also a feasible flow in the relabeled network $(G = (N,F), u^\mu, \gamma^\mu)$, because the excess $e_{g^\mu}(i)$ in a relabeled network is equivalent to $e_g(i)/\mu(i)$. Moreover, we have $e_g(t) = e_{g^\mu}(t)$, since $\mu(t) = 1$. Therefore, the relabeled network is an equivalent instance of the generalized maximum flow problem. This relabeling operation was initially introduced by Glover-Klingman [20] to prove that if the incidence matrix of a network has rank $n-1$, the network with gains can be converted into a traditional no gain network. A residual relabeled network can also be defined similarly.

A label μ satisfying Eq.(2.1) for every arc $(i,j) \in A^g$ is called a *canonical label* for g if, for any node i from which the sink t is reachable in G^g, there is an i-t path in G^g with arcs of $\gamma_\mu(i,j) = 1$. The canonical label $\mu(i)$ is given by the inverse of the maximum gain of an i-t path in the residual network. A maximum gain i-t path is called a *highest-gain path from i*.

2.3. Generalized maximum flows versus minimum cost flows

While the generalized maximum flow problem is a generalization of the traditional maximum flow problem, it is associated with the traditional minimum cost circulation problem, which is formulated as

(MCF)

$$\min\{\sum_{(i,j)\in F} c(i,j)g(i,j) \mid 0 \le g \le u, \sum_{\{j\in N|(i,j)\in F\}} g(i,j) = \sum_{\{j\in N|(j,i)\in F\}} g(j,i) \text{ for all } i \in N\},$$

where $c : F \to \mathbb{R}$ is a cost function. This relationship was first pointed out by Truemper [60]. There is one-to-one correspondence between flow-generating cycles and negative-cost cycles, by substituting $c(i,j) = -\log\gamma(i,j)$. Thus, the optimality conditions resemble each other. The corresponding optimality conditions for minimum cost flows are as follows. For a flow g satisfying the restrictions of (MCF), the following statements are equivalent:

MCF-(a) g is minimum cost flow.
MCF-(b) the residual network G^g does not have any negative-cost cycles.
MCF-(c) there exists $\pi : N \to \mathbb{R}$ such that $c(i,j) + \pi(i) \ge \pi(j)$ holds for every $(i,j) \in A^g$.

Indeed, the minimum cost flow algorithms are good guides to developing algorithms for generalized maximum flow problems. In Section 4, we will overview generalized maximum flow algorithms and contrast then with minimum cost flow algorithms.

The transformation from $-\log\gamma(i,j)$ to the cost $c(i,j)$ also makes it easy to compute highest-gain paths and the canonical label. Under this transformation, a highest-gain path corresponds to a shortest path. Hence, when G^g does not contain any flow-generating cycles, highest-gain paths and the canonical label can be determined using Bellman-Ford shortest path computation.

3. Framework of Algorithms

Almost all combinatorial algorithms for (GMF) transform the problem to (GMF′) on networks with no flow-generating cycles and solve this as (GMF′). These algorithms can be outlined as follows:

Step 1: Transform (GMF) into (GMF′) on a network having no flow-generating cycles.
Step 2: Find an optimal flow g' of (GMF′).
Step 3: Convert g' into an optimal flow of (GMF).

Step 1 is implemented by repeatedly finding a flow-generating cycle in a residual network and pushing as much flow as possible along the cycle. This process, called the *flow-generating cycle-canceling procedure*, creates positive excesses at nodes on the flow-generating cycles, but no negative excesses at any node. The details on this procedure will be described in Section 3.1. We assume that a flow-generating cycle-canceling procedure finds a flow g^0. Step 2 solves (GMF′) on the network $(G^{g^0}, \gamma, u_{g^0})$ with the initial excess e_{g^0}. There are several algorithms solving (GMF′), which will be discussed in Section 4. Almost all polynomial time algorithms for (GMF′) do not use the optimality conditions of Theorem 2.2, but depend on a property such that if the difference between the value of an optimal flow and the current flow is very small and there are no flow-generating cycles on the residual network, then flow can be rounded to an optimal flow.

Lemma 3.1 ([61, Lemma 2.5.1]) *Let g^* be an optimal flow and g a feasible flow with $e_{g^*}(t) - e_g(t) < B^{-m}$ and G^g does not contain any flow-generating cycle. We can then find an optimal flow by solving a traditional maximum flow in the subgraph of G^g induced by arcs with $\gamma^\mu(i,j) = 1$, where μ is the canonical label for g.* ∎

We can find the similar results in Goldberg-Plotkin-Tardos [23] and Goldfarb-Jin-Orlin [32].

Finally, Step 3 is used to find an optimal flow of (GMF) from g^0 and g' as follows. Let us first decompose a flow $g^0 + g'$ into elementary flows. Since $e_{g^0+g'}(i) \geq 0$ for any $i \in N$, the decomposed elementary flow is Type 2, 3, or 5. Let $g^1, g^2, \ldots, g^l$ be the decomposed elementary flow of Type 3 where the tail node of the path is not the sink. Then $g^0 + g' - (g^1 + g^2 + \cdots + g^l)$ is a generalized flow and its flow value does not decrease from $g^0 + g'$. Because (GMF$'$) is a relaxation of (GMF), the resulting flow is optimal for (GMF). Since the flow decomposition can be performed in $O(nm)$, Step 3 runs in $O(nm)$ time.

3.1. Flow-generating cycle-canceling procedure

Since flow-generating cycles correspond to negative-cost cycles, efficient selection rules for canceled cycles are derived from negative-cost cycle-canceling algorithms for minimum cost flows. The following *maximum-mean-gain cycle-canceling procedure* is an adaptation of the minimum-mean cycle-canceling algorithm in Goldberg-Tarjan [26], which is one of the basic polynomial time algorithms for minimum cost flows.

Step 1: If the residual network does not have any flow-generating cycles, stop.

Step 2: Find a cycle C maximizing the mean-gain value $\gamma(C)^{1/|C|}$ in the residual network.

Step 3: Send flow around C until an arc is saturated. Go back to Step 1.

We can directly extend an analysis in Goldberg-Tarjan [26], which gives a weakly polynomial time bound on the number of iterations for the minimum-mean cycle-canceling algorithm, to the above procedure. Let ε_h be the maximum mean-gain value $\gamma(C)^{1/|C|}$ of the cycle C canceled at the hth iteration. The following lemma is associated with Lemmas 3.5 and 3.6 in [26].

Lemma 3.2 *The sequence of $\varepsilon_1, \varepsilon_2, \ldots,$ is nonincreasing. Moreover, after m cancellations, the maximum mean-gain value ε_h reduces at most $\varepsilon_h^{\frac{n-1}{n}}$. Hence we have $\varepsilon_{h+nm} < (\varepsilon_h)^{1/2}$ for any h with $\varepsilon_h > 1$.* ∎

It follows from this lemma that the iteration number of the procedure is bounded by $O(mn^2 \log B)$.

By modifying the maximum-mean-gain cycle-canceling procedure, Goldberg-Plotkin-Tardos [23] developed an $O(mn^2 \log n \log B)$ time procedure, which is parallel with the cancel-and-tighten algorithm for minimum cost flows [26]. We can construct another flow-generating cycle canceling procedure, which is an adaptation of the other minimum cost flow algorithm that iteratively cancels negative-cost cycles (See [56], for a survey of cycle-canceling algorithms for minimum cost flows). Radzik [49] and Tardos-Wayen [58], respectively, proposed flow-generating cycle-canceling procedures accommodated to the algorithms solving (GMF$'$).

The end of this section discusses a strongly polynomial time bound for the maximum-mean-gain cycle-canceling procedure. Unfortunately, we cannot obtain any strongly polynomial time bound on the iteration number of either procedure, the maximum-mean-gain canceling or Goldberg-Plotkin-Tardos' cancel-and-tighten, by extending the minimum cost flow case in [26], because the flow decomposition theorem is used in analysis. Applying a more general analysis, Radzik [48] found that an $O(m^2 n \log^2 n)$ bound on the running time of Goldberg-Plotkin-Tardos' procedure, which matches the strongly polynomial bound of Goldberg-Tarjan[26]. We will give a brief outline of Radzik's analysis for the maximum-mean-gain cycle-canceling procedure in the Appendix, since his analysis may give a clue in a strongly polynomial bound for finding a generalized maximum flow.

4. Combinatorial Algorithms

This section surveys algorithms for (GMF′) on a network with no flow-generating cycles. We also compare generalized maximum flow algorithms with maximum flow algorithms and minimum cost flow algorithms.

4.1. Basic algorithms

Jewell [36] and Onaga [44] proposed first combinatorial algorithms generalizing on Ford-Fulkerson's augmenting path algorithm of traditional maximum flows [14]. Onaga [44] observed that if flow augmentation is done along a highest-gain path in a network with no flow-generating cycles, then the residual network of the resulting flow contains no flow-generating cycles. Thus, his algorithm iteratively augments a flow along a highest-gain path from a node with a positive excess until no such path exists. Truemper [59] proposed an algorithm that augments flows by solving traditional maximum flow on a residual network induced by arcs which may construct some highest-gain path, i.e., induced by arcs with $\gamma^{\mu}(i,j) = 1$ for the canonical label μ.

Both algorithms maintain a feasible flow and a label function μ that satisfies Eq.(2.1).

After each maximum flow is computed in Truemper's algorithm, the canonical label $\mu(i)$ with a positive excess strictly increasing. Hence, we have

Lemma 4.1 *The number of iterations in Truemper's algorithm is bounded by n plus the number of different gains of paths in the original network.* ∎

However, both Onaga's and Truemper's algorithms run in pseudo-polynomial time, since they are specialized to Ford-Fulkerson's augmenting path algorithm.

Truemper [60] surveyed other basic algorithms and overviewed the relationship between these algorithms and algorithms for traditional minimum cost flows. By using the cost function defined by $-\log\gamma$, he pointed out that Onaga's algorithms was analogous to the successive shortest path algorithm and Truemper's to the primal-dual algorithm for minimum cost flows. (Here, we call minimum cost flow algorithms by following Ahuja-Magnanti-Orlin [2].)

For traditional maximum flows, on the other hand, Edmonds-Karp [11] suggested two polynomial-time implementations of the augmenting path algorithm. One is to augment flow along a path which produces a maximum increase in the flow value. The other is to augment flow along a path with a minimum number of arcs. These respective implementations are closely related to the most-helpful cycle-canceling algorithm [5] and the minimum-mean cycle-canceling algorithm [26] for minimum cost flows, in the translated network that follows. For a network of a maximum flow instance, add a new arc (t,s) with infinite capacity. Define the costs of all original arcs as zero and the cost of (t,s) as -1. Then, a minimum cost flow in the resulting network restricted to the original network is a maximum flow. In this network, Ford-Fulkerson's algorithm corresponds to Klein's negative cycle-canceling algorithm [39].

We now describe a polynomial-time implementation of the augmenting path algorithm for generalized maximum flows using Edmonds-Karp's idea. Instead of a highest-gain path, we use a most-helpful path, which is a path from a node with positive excess to the sink and which produces a maximum increase in the flow value.

Lemma 4.2 *Assume that a residual network G^g for a feasible flow g does not have any flow-generating cycles. When we augment a flow along a most-helpful path in G^g and obtain a feasible flow g', then*

$$e_{g^*}(t) - e_{g'}(t) \leq \frac{m-1}{m}(e_{g^*}(t) - e_g(t)),$$

where g^* denotes an optimal flow.

Proof. Since G^g does not have any flow-generating cycles, the flow $g^* - g$ can be decomposed into at most m elementary pseudo flows of types 1, 2 and 4. Then a similar argument to the proof of Lemma 2 in Barahona-Tardos [5] establishes the lemma. That is, there exist simple paths P_h in G^g from a node with positive excess $e_g(i) > 0$ to the sink t and positive scalar α_h for $h = 1, \ldots, k$, such that $k \leq m$, $\sum_{h=1}^{k} \alpha_h \gamma(P_h) = e_{g^*}(t) - e_g(t)$ and $\alpha_h \leq u_g(P_h)$ holds for every $h = 1, \ldots, k$, where $u_g(P_h) = \max\{\alpha \mid \alpha\gamma(P_h(i)) \leq u_g(i,j), \forall \text{arc } (i,j) \text{on } P_h\}$ and $P_h(i)$ is the subpath of P_h from the head node of P_h to a node i. Therefore, the path P_h in G^g that maximizes $u_g(P_h)\gamma(P_h)$ proves the lemma. ∎

Unfortunately, flow augmentation along a most-helpful path may result in flow-generating cycles in the residual network. Therefore, one needs to cancel such cycles in each iteration. A terminal condition is derived from Lemmas 3.1 and 4.2. The algorithm is expressed by the following:

Step 1 Cancel all flow-generating cycles.

Step 2 If the flow value increment was less than $1/(mB^m)$ at the last iteration, find the canonical label μ and obtain an optimal flow by computing a maximum flow in the subgraph of G^g induced by arcs with $\gamma^\mu(i,j) = 1$.

Step 3 Find a most-helpful path. If there is no path from a node with positive excess to the sink, then stop. Otherwise, augment a flow along the path and go to Step 1.

Let $\bar{g}$ and g' be feasible flows at the beginning and end of an iteration. Since Step 1 does not decrease the flow value, it follows from Lemma 4.2 that $e_{g'}(t) - e_{\bar{g}}(t) \geq (1/m)(e_{g^*}(t) - e_{g'}(t))$, where g^* is an optimal flow. Hence, when the increment of the flow value $e_{g'}(t) - e_{\bar{g}}(t)$ is less than $1/(mB^m)$, the conditions of Lemma 3.1 hold at Step 2 and we can obtain an optimal flow. Since the discrepancy $e_{g^*}(t) - e_g(t)$ decreases at least half after m iterations, after $O(m^2 \log B)$ iterations, $e_{g'}(t) - e_{\bar{g}}(t) \leq e_{g^*}(t) - e_{\bar{g}}(t) \leq 1/(mB^m)$ holds and the algorithm terminates. As a most-helpful path can be found in polynomial time at Step 3, the most-helpful augmenting path algorithm runs in polynomial time. However the time complexity cannot be strengthened to be strongly polynomial, since the algorithm is specialized to Edmonds-Karp's maximum-capacity augmenting path algorithm, which is not a strongly polynomial algorithm [47]. It is also costly to cancel flow-generating cycles in each iteration. One of the implementations of this algorithm is a fat-path algorithm that will be described in the next section.

The relationships among basic algorithms described in this section are in Figure 1. In Figure 1, the algorithms in the first, second and third columns are for maximum flows, minimum cost flows, and generalized maximum flows, respectively. The boxes with double lines mean that the algorithm in it runs in polynomial time. The boxes with thick double lines mean that the algorithm in it runs in strongly polynomial time. It seems that one of the keys to deriving a strongly polynomial time algorithm of generalized maximum flows is through a generalization of Edmonds-Karp's shortest path algorithm. Moreover, Dinic [10] introduced a concept of shortest path networks and established blocking flow for traditional maximum flow problems. His algorithm is meant to augment flows along several shortest paths, simultaneously. Recently, Fujishige [16] proposed an algorithm that augments flows along several big capacity paths, simultaneously. Extensions of these algorithms are also interesting.

4.2. Scaling algorithms

Scaling methods have been used extensively to derive polynomial-time algorithms for a wide variety of networks. In generalized maximum flows, there are two scaled factors, capacity and gain.

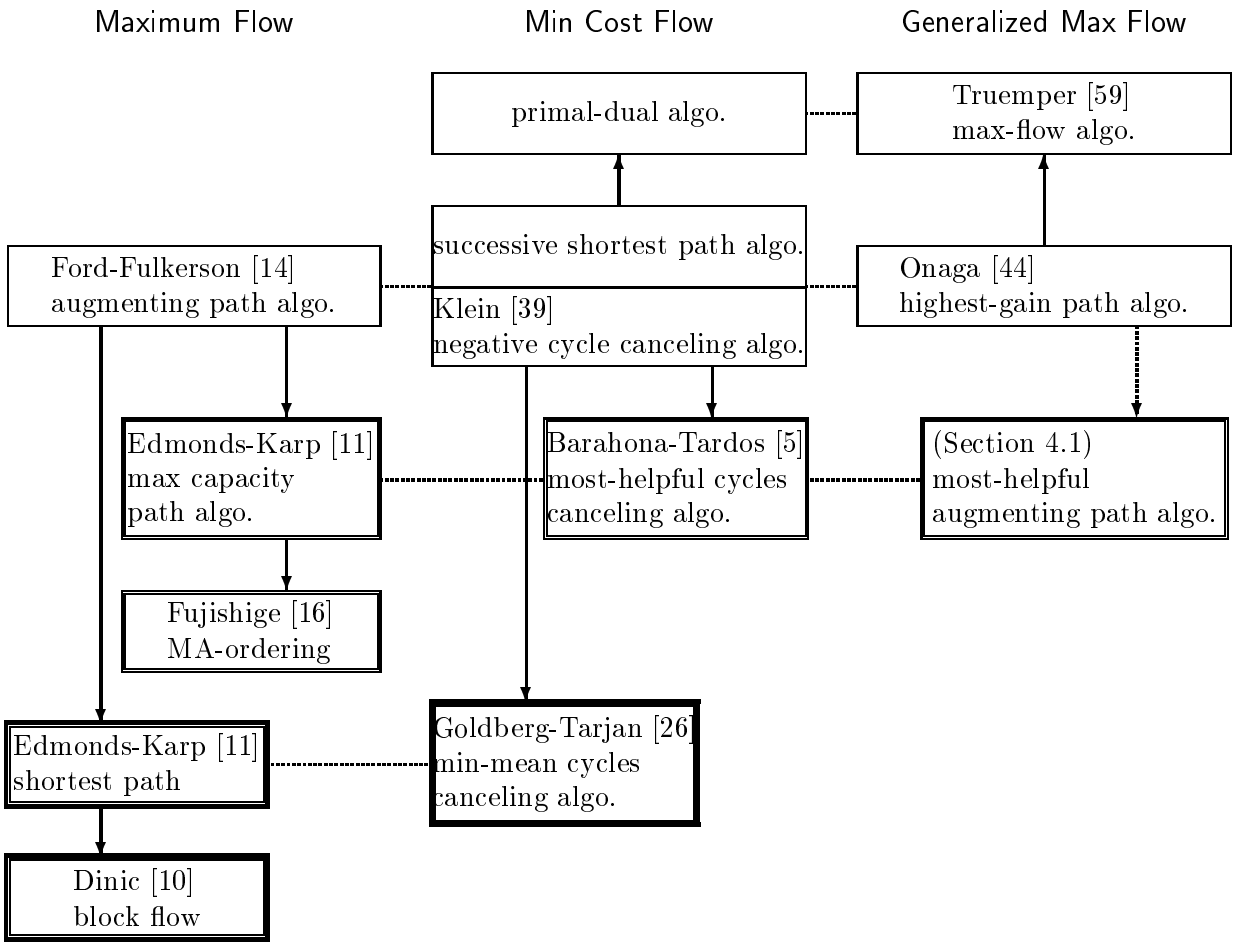

Figure 1: Relationships among basic algorithms for maximum flows, minimum cost flows, and generalized maximum flows

Let us first overview capacity scaling type algorithms for maximum flow, minimum cost flow, and generalized maximum flow problems. Let Δ be a scaling parameter. In a Δ-scaling phase, each flow augmentation carries Δ units of flow.

The first capacity scaling type algorithm for a generalized maximum flow problem is the fat-path algorithm by Goldberg-Plotkin-Tardos [23]. A simple i-t path is Δ-fat if, given unlimited excess at i, at least Δ units of additional flow can be sent to t along this path. Let $A^g_\Delta = \{(i,j) \in A^g \mid u_g(i,j)\gamma(P_i) \geq \Delta\}$, where P_i is a highest-gain path from i in G^g. Note that any path to the sink in $G^g_\Delta = (N, A^g_\Delta)$ is Δ-fat. Each Δ-scaling phase constructs G^g_Δ and augments flow along a highest-gain path in G^g_Δ. Similar to the most-helpful augmenting path algorithm, this flow augmentation may result in flow-generating cycles in the residual network G^g, although G^g_Δ does not contain any flow-generating cycles. Therefore, one needs to cancel such cycles in the beginning of each scaling phase. We describe the fat-path algorithm as follows:

Step 0 Set $\Delta := B^2$.

Step 1 Construct G^g_Δ.

Step 2 Find a highest-gain path and augment flow along this path. If there is no path from a node with positive excess to the sink in G^g_Δ, then go to Step 3. Otherwise, repeat Step 2.

Step 3 Reduce Δ by a factor of two and perform a flow-generating cycle-canceling procedure. If $\Delta < B^{-3m}/(n+m)$, then find the canonical label μ and obtain an optimal flow by computing a maximum flow in the subgraph of G^g induced by arcs with $\gamma^\mu(i,j) = 1$. Otherwise, go to Step 1.

Since $e_{g^*}(t) - e_g(t) \leq (n+m)\Delta$ holds for a flow g after Step 2, each scaling phase augments flow at most $2(n+m)$ times. When $\Delta < B^{-3m}/(n+m)$, the conditions of Lemma 3.1 hold at Step 3 and we can obtain an optimal flow. Hence the algorithm terminates after $O(m \log B)$ scaling phases. Since each augmentation can be done in $O(m+n\log n)$, the total time complexity of the algorithm is $O((m^2 + mn\log n + CC)m \log B)$, where CC denotes the running time of a flow-generating cycle-canceling procedure. The fat-path algorithm reduces the number of times a flow-generating cycle-canceling procedure is called, which is costly in the time complexity, from the most-helpful augmenting path algorithm. Radzik [48] improved the running time of the fat-path algorithm by canceling flow-generating cycles that only consists of big capacity arcs. Tardos-Wayne [58] improved the fat-path algorithm through the same idea, using a gain scaling technique, which will be described later.

When all gains are given by one, the algorithm corresponds to Gabow's capacity scaling algorithm [17] for maximum flows. Although Gabow originally designed a bit-scaling algorithm, a Δ-scaling type algorithm is described in Ahuja-Magnanti-Orlin [2]. For a traditional maximum flow problem with integral capacities, the algorithm can be found an optimal flow when $\Delta < 1$. Terefore, there are $O(\log B)$ scaling phases. Moreover, after a Δ-scaling phase, we only need to reduce the parameter Δ.

There are two types of corresponding capacity scaling algorithms for minimum cost flows. One is a variant of the successive shortest path algorithm, and the other is a variant of the negative cycle canceling algorithm. The former was developed by Edmods-Karp [11] and Röck [52], independently. Each scaling phase sends flow along a shortest path in G^g_Δ from a positive excess node to a negative excess node. At the beginning of each scaling phase, the flow is changed such that the optimality condition MCF-(c) holds. For the current π, if $c(i,j) + \pi(i) - \pi(j) < 0$ then the flow value $g(i,j)$ is updated to $u_g(i,j)$, which causes a positive excess and a negative excess at the end nodes of (i,j). This preprocessing of each

scaling phase is associated with the flow-generating cycle-canceling procedure in the fat-path algorithm. The second capacity scaling algorithm for a minimum cost flow problem is Sokkalingam-Ahuja-Orlin's cycle-canceling algorithm [57]. Their algorithm is regarded as a variant of the cancel-and-tighten algorithm by Goldberg-Tarjan [26], since it uses a dual variable. Unfortunately, this algorithm does not coincide with the capacity scaling algorithm for maximum flows, even if it is applied to a translated network from a maximum flow problem.

Excess scaling methods are similar to capacity scaling methods in that each flow augmentation carries Δ units of flow. Orlin [46] developed an excess scaling algorithm for minimum cost flows. The algorithm transforms the problem to an uncapacitated problem (for details, see [2, Section 2.4]) and repeats to augment flow along a path from a node with a positive excess more than Δ to a node with a negative excess less than $-\Delta$. Goldfarb-Jin [30] proposed an excess scaling algorithm that works directly with the original network without being transformed into an uncapacitated network. Their algorithm uses *arc excesses*, which were originally introduced by Goldfarb-Jin[29] for generalized flows. When one augments flow along a path from a node with excess more than Δ to a node with excess less than $-\Delta$, if the path contains an arc (i,j) with $u_g(i,j) < \Delta$, the overflow $\Delta - u_g(i,j)$ is stored in the excess of the arc (i,j). Let $e_g : A \to \mathbb{R}$ be an arc excess with respect to a flow g. When one augments Δ units of flow on an arc (i,j), the following process is performed.

Arc-augmentaion I

$e_g(i) := e_g(i) - \Delta$;
$\alpha := \min\{e_g(i,j) + \Delta, u_g(i,j)\}$;
$g(i,j) := g(i,j) + \alpha$; $\quad e_g(i,j) := e_g(i,j) + \Delta - \alpha$;
$\beta := \min\{e_g(j,i) + \alpha, \Delta\}$;
$e_g(j) := e_g(j) + \beta$; $\quad e_g(j,i) := e_g(j,i) + \alpha - \beta$.

Armstrong-Jin [3] developed a simpler augmentation rule which is used in a dual network simplex algorithm for minimum cost flows.

Arc-augmentaion II

$e_g(i) := e_g(i) - \Delta$;
$\alpha := \min\{\Delta, u_g(i,j)\}$;
$g(i,j) := g(i,j) + \alpha$; $\quad e_g(i,j) := e_g(i,j) + \Delta - \alpha$;
$e_g(j) := e_g(j) + \alpha$;
if $(\alpha < \Delta)$ **then** $\{e_g(j) := e_g(j) + e_g(j,i)$; $\quad e_g(j,i) = 0;\}$.

The flow augmentation along a shortest path can be implemented by applying either process to an arc (i,j) in the order of the shortest path from the tail node, until the process is done for all arcs on the path or $e_g(i) < \Delta$ holds. Both arc-augmentation rules satisfy the condition $0 \le e_g(i,j) + e_g(j,i) < \Delta$ for all arcs (i,j). Hence by putting each arc excess back into the excess of its head node after each Δ scaling phase, the number of flow augmentations in a scaling phase is $O(n+m)$.

For the generalized maximum flow problem, Goldfarb-Jin-Orlin [32] developed an excess scaling algorithm with arc excesses. Each scaling phase repeats to compute the canonical label and to augment flow along a highest-gain path from a node with more than Δ relabeled excess. Since $\gamma^\mu(i,j) = 1$ for any arc (i,j) in a highest-gain path with the canonical label μ, each flow augmentation adapts the rule of Arc-augmentation I for the relabeled network. That is, in Arc-augmentation I, $e_g(i), g(i,j), u_g(i,j)$ and $e_g(i,j)$ are replaced by $e_g^\mu(i), g^\mu(i,j), u_g^\mu(i,j)$ and $e_g^\mu(i,j) = e_g(i,j)/\mu(i)$, respectively. (Unfortunately, the rule Arc-augmentation II may not be appropriate for generalized flows.) Note that flow-generating cycles are not generated by these flow augmentations and we no longer need to perform a

flow-generating cycle-canceling procedure. Therefore, each scaling phase runs in $O(m(m + n \log n))$ time, and the total time complexity of Goldberg-Jin-Orlin's excess scaling algorithm is $O(m^2(m+n \log n) \log B)$. This excess scaling algorithm also gives us an idea for an efficient dual network simplex algorithm for generalized flow due to Goldfarb-Jin-Lin [31].

Let us return to Orlin's excess scaling algorithm [46] for minimum cost flows. He introduced not only an excess scaling technique, but also a graph contraction process. By contracting arcs whose flow is so large in a Δ-scaling phase that they are guaranteed to have positive flow in all subsequent scaling phases, his algorithm runs in strongly polynomial time, $O((m + n \log n)m \log n)$. Recently, Radzik [50] developed a framework that reduced the size of a given graph for generalized network flows. There are three operations: contracting large-capacity arcs, shortcutting small reverse-flow paths, and by-passing and removing some nodes. Combining this framework with Goldfarb-Jin-Orlin's excess scaling algorithm, he proved an $O(nm(m+n \log n) \log B)$ bound on the running time of the resulting algorithm.

Let us turn to gain scaling methods. Tardos-Wayne [58] introduced gain scaling technique. Each gain $\gamma(i, j)$ is rounded down to $\tilde{\gamma}(i, j) = b^{\lfloor \log_b \gamma(i,j) \rfloor}$, where $b > 1$. It is obvious that $\gamma(i, j)/b \leq \tilde{\gamma}(i, j) \leq \gamma(i, j)$ holds for all arcs (i, j). This gain scaling is associated with a scaling technique $\lfloor \frac{c(i,j)}{\varepsilon} \rfloor \varepsilon$ with a scaling parameter ε, which often appears in cost scaling algorithms.

At the beginning of a gain scaling algorithm, we find the canonical label and construct a relabeled network. Note that in this relabeled network, no gain is more than one. We next round down the relabeled gains. Denote by $\tilde{\mathcal{N}}$ the resulting network. If g is a feasible flow in $\tilde{\mathcal{N}}$, then g is also feasible in the original network, because this replacement may cause additional positive excesses at some nodes, but no negative excesses. The following lemma shows that the flow value of g, which is an optimal flow in $\tilde{\mathcal{N}}$, is close to the optimal value of the original network.

Lemma 4.3 ([58]) *If g^* and g are optimal flows in the original network and in $\tilde{\mathcal{N}}$, respectively, then $e_g(t) \geq e_{g^*}(t)/b^n$ holds. When $b = (1 + \xi)^{1/n}$ for $0 < \xi < 1$, we have $e_g(t) \geq (1 - \xi)e_{g^*}(t)$.* ∎

When $\xi = 1/(nB^{m+2})$, we can find an optimal flow using Lemma 3.1, since $e_{g^*}(t) \leq nB^2$. Whereas, gain-scaling algorithms find an optimal flow without reducing a scaling parameter b. Radzik [49] proposed a fascinating scheme called a recursion. It is used to speed up computations for generalized maximum flows. A flow g is called an η-optimal when $e_g(t) \geq (1 - \eta)e_{g^*}(t)$ holds, where g^* is an optimal flow.

Lemma 4.4 ([49]) *Let g_1 be an η_1-optimal flow in a network and g_2 an η_2-optimal flow in the residual network G^{g_1}. Then $g_1 + g_2$ is an $(\eta_1\eta_2)$-optimal flow in the original network.* ∎

Hence, if we have a subroutine that finds a 1/2-optimal flow, we can find a $1/(nB^{m+2})$-optimal flow by calling this subroutine $\log(nB^{m+2})$ times. The framework of gain scaling algorithms with the recursion can be described as follows.

Step 1 Find the canonical label and construct a relabeled network.

Step 2 Scale down the relabeled gains and find an optimal flow g in $\tilde{\mathcal{N}}$.

Step 3 Cancel all flow-generating cycles. If the total excess of all nodes is less than B^{-2m}, go to Step 4, otherwise, go to Step 1.

Step 4 Find the canonical label and obtain an optimal flow by computing a maximum flow in the subgraph of G^g induced by arcs with $\gamma^\mu(i, j) = 1$.

From now, let us assume that $b = (3/2)^{1/n}$, which implies that Step 2 in the above framework will find a 1/2-optimal flow.

Tardos-Wayne [58] presented three gain scaling algorithms where Truemper's maximum-flow algorithm, the fat-path algorithm, and a preflow-push type algorithm are adopted in Step 2. Since scaled gains are all integer powers of b, there are at most $1 + \log_b B^{2n}$ different gains of paths. Hence, it follows from Lemma 4.1 that Truemper's algorithm computes a maximum flows $O(n^2 \log B)$ times at each Step 2. Adapting Goldberg-Tarjan's $O(mn \log(n^2/m))$ maximum flow algorithm [25], this gain scaling algorithm runs in $O(m^2 n^3 \log(n^2/m) \log B \log(nB))$ time. If the fat-path algorithm is adopted in Step 2, we can speed up the flow-generating cycle-canceling procedure in each Δ-scaling. By substituting $c(i,j) = -\log_b \tilde{\gamma}(i,j)$, each c becomes an integer bounded by $O(n \log B)$. We can then improve the running time of a cycle-canceling procedure by using a weakly-polynomial negative-cycle canceling algorithm. Although this fat-path type gain scaling algorithm uses two scaling parameters, i.e., capacity and gain, the workings of the algorithm are different from Ahuja-Goldberg-Orlin-Tarjan's double scaling algorithm [1] for a minimum cost flow problem which also uses two scaling parameters, capacity and cost. The preflow-push type gain scaling algorithm is associated with a cost scaling algorithm due to Goldberg-Tarjan [27]. An *active node* is a node with a positive excess and a path to the sink t in the residual network. The algorithm consists of two operations: Operation Relabel(i) applies if a node i has a positive excess and $\tilde{\gamma}^{\mu}(i,j) < 1$ for all arcs $(i,j) \in A^g$ emanating from i. It increases a node label $\mu(i)$ by a factor of $b^{1/n}$. Operation Push(i,j) applies if i has a positive excess, $(i,j) \in A^g$ and $\tilde{\gamma}^{\mu}(i,j) > 1$. It pushs $\delta = \min\{e_g(i), u_g(i,j)\}$units of flow along the arc (i,j). The push-relabel algorithm repeats to apply these push and relabel operations to an active node until the residual network does not have any active nodes. An analogous analysis in Goldberg-Tarjan [27] leads to the time complexity of this algorithm. By using a data structure called dynamic tree, it runs in $O(m^2 n^3 \log n \log B)$ time.

We illustrate the relationships among scaling algorithms described in this section in Figure 2, where the algorithms in the first, second and third columns are for maximum flows, minimum cost flows, and generalized maximum flows, respectively. The boxes with double lines mean that the algorithm in it runs in polynomial time. The boxes with thick double lines mean that the algorithm in it runs in strongly polynomial time.

4.3. Other algorithms

Goldberg-Plotkin-Tardos[23] designed two polynomial-time algorithms, fat-path algorithm described in Section 4.2 and MCF algorithm. Algorithm MCF maintains a pseudo flow with no positive excess nodes except the sink. It repeats to perform a traditional minimum cost flow computation with cost $c(i,j) = -\gamma(i,j)$. In contrast to augmenting path type algorithms, this algorithm draws back flows from the sink to nodes with negative excesses. It is exciting that we can obtain a polynomial time bound without using any scaling technique. Goldfarb-Jin [29] improve the MCF algorithm. Instead of using a traditional minimum cost flow computation at each iteration, they augment flow along highest-gain paths using arc excesses.

The combinatorial approach turns out to be superior to the linear programming approach when only an approximate solution is sought. For generalized maximum flows, several strongly polynomial approximation algorithms are developed. Cohen-Megiddo [7] presented the first fully polynomial approximate algorithm based on solving linear programs with two variables per inequality (TVPI). Their algorithm finds an η-optimal flow in $O(m^2 n^2 (\log m + \log^2 n) \log \eta^{-1})$ time. Oldham [43] and Fleischer-Wayne [13] proposed fully polynomial approximation algorithms based on a fractional packing framework due to Garg-Könemann [18]. These approximation algorithms also solve a variant of generalized

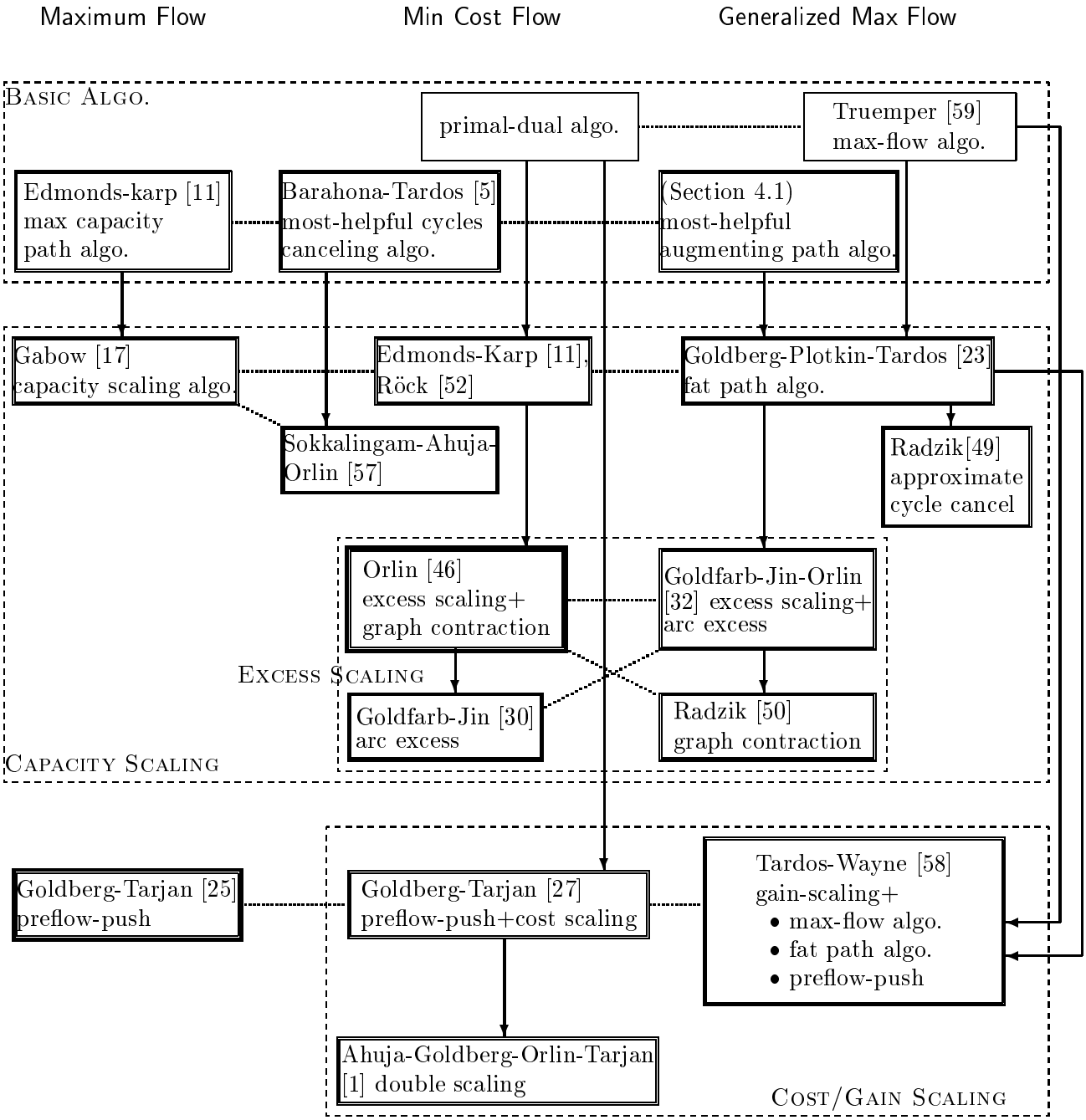

Figure 2: Relationships among scaling algorithms for maximum flows, minimum cost flows, and generalized maximum flows

network flows as generalized minimum cost flows and generalized multicommodity flows.

Finally, the history and the state of the art of the combinatorial algorithms for generalized maximum flows are summarized in Table 1, where CC, MF, and MC stand for the time complexities for performing a flow-generating cycle canceling procedure, for computing a traditional maximum flow, and for computing a traditional minimum cost flow, respectively. The time complexity containing η implies the time bound to find an η-optimal flow.

5. Conclusion and Further Research

This paper emphasizes algorithmic aspects of the generalized maximum flows. A big open problem is development of a strongly polynomial time algorithm. Another research direction is to design an algorithm which runs practically fast. Radzik-Yang [51] presented experimental evaluations of Goldfarb-Jin-Orlin's excess-scaling algorithm and Tardos-Wayne's preflow-push algorithm for generalized maximum flows. In their result, the combinatorial generalized flow algorithms are not superior to a commercial linear programming package.

In generalized network, Wayne [62] proposed the first polynomial combinatorial algorithm for minimum cost flow. For multicommodity flow, there are no known exact polynomial combinatorial algorithms. It is plausible that further exploitation of the combinatorial structure of these problems will eventually lead to bounds unattainable by general-purpose linear programming methods.

Recently, general models of generalized maximum flows have been discussed. Nakayama-Su [42] extended Minoux's maximum balanced flow problem in generalized network, which finds maximum flow satisfying the constraint that a flow of any arc is bounded by a fixed proportion of the flow value. Eguchi-Fujishige-Takabatake [12] studied a generalized independent-flow problem. A maximum flow problem with concave gains was explored in [55].

Acknowledgments

The author grateful to anonymous referees for their helpful comments that improved the presentation of this paper.

References

[1] R. K. Ahuja, A. V. Goldberg, J. B. Orlin and R. E. Tarjan: Finding minimum-cost flows by double scaling. *Mathematical Programming*, **53** (1992), 243–266.

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

[3] R. D. Armstrong and Z. Jin: A new strongly polynomial dual network simplex algorithm. *Mathematical Programming*, **78** (1997), 131–148.

[4] T. Asano and Y. Asano: Resent developments in maximum flow algorithms. *Journal of the Operations Research Society of Japan*, **43** (2000), 2–31.

[5] F. Barahona and É. Tardos: Note on Weintraub's minimum-cost circulation algorithm. *SIAM Journal on Computing*, **18** (1989), 579–583.

[6] D. Bertsekas: *Network Optimization: Continuous and Discrete Models* (Athena Scientific, Belmont, 1998).

[7] E. Cohen and N. Megiddo: New algorithms for generalized network flows. *Mathematical Programming*, **64** (1994), 325–336.

Table 1: Combinatorial algorithms for generalized maximum flows

Reference (Year)	Technique Complexity
Jewell [36] (1962) Onaga [44] (1966)	highest gain augmenting path
Truemper [59] (1973)	highest gain path+maximum flow algo. $O(B^{3n} \cdot \mathrm{MF})$ pseudo-polynomial
(Section 4.1)	most-helpful path $O(m^2 \log B \cdot \mathrm{CC})$
Goldberg-Plotkin-Tardos [23] (1991)	fat-path(Δ-scaling) $O((m(m + n \log n) + \mathrm{CC})m \log B)$ $\Rightarrow o((m^2 n^2 \log n \log^2 B/(\log(n^2/m) + \log\log n + \log\log B))$
Radzik [49] (1998)	fat path algo.+ approximate cycle canceling $O(m^2(m + n \log n \log(n \log B)) \log B)$
Goldfarb-Jin-Orlin [32] (1997)	excess scaling (Δ-scaling)+ arc excess+highest gain path $O(m^2(m + n \log n) \log B)$
Radzik [50] (2004)	contracting the network+ Goldfarb-Jin-Orlin's algo. $O(mn(m + n \log n) \log B)$
Tardos-Wayne [58] (1998)	gain scaling+recursion+Truemper's algo. $O((n^2 \log B \cdot \mathrm{MF} + \mathrm{CC})m \log B)$ $\Rightarrow o(m^2 n^3 \log(n^2/m) \log B \log(nB))$ gain scaling+recursion+fat path algo. $O(m^2(m + n \log(n \log B)) \log B)$ gain scaling+recursion+preflow-push algo. $O(m^2 n^3 \log n \log B)$
Goldberg-Plotkin-Tardos [23] (1991)	min cost flow algo. (MCF) $O(n^2 \log B \cdot \mathrm{MC})$ $\Rightarrow o(mn^2(m + n \log n) \log n \log B)$
Goldfarb-Jin [29] (1996)	arc excess+framework of MCF $O(n^2 m(m + n \log n) \log B)$
Goldfarb-Jin [28] (1994)	dual network simplex+[29] $O(n^2 m(m + n \log n) \log B)$
Goldfarb-Jin-Lin [31] (2002)	dual network simplex + excess scaling $O(m^2(m + n \log n) \log B)$
Goldfarb-Lin [33] (2002)	interior point method+ augmenting path $O(m^3 n^2(\log m + \log^2 n) \log B)$
Cohen-Megiddo [7] (1994)	TVPI+packing $O(m^3 n^2(\log m + \log^2 n) \log B)$
Oldham[43] (2001)	generalized shortest path (GSP) +fractional packing $O(\eta^{-2} m^2 n^2 \log m \log n)$
Fleischer-Wayne [13] (2002)	approximate GSP+ fractional packing $O(\eta^{-2} m(m + n \log m) \log n)$ gain scaling + approximate GSP + fractional packing $O(m^2(m + n \log(n \log B)) \log B)$

[8] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank and A. Schrijver: *Combinatorial Optimization* (Wiley, New York, 1998).

[9] G. B. Dantzig: *Linear Programming and Extensions* (Princeton University Press, Princeton, 1962).

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

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

[12] A. Eguchi, S. Fujishige and T. Takabatake: A polynomial-time algorithm for the generalized independent-flow problem. *Journal of Operations Research Society of Japan*, **47** (2004), 1–17.

[13] L. K. Fleischer and K. D. Wayne: Fast and simple approximation schemes for generalized flow. *Mathematical Programming*, **91** (2002), 215–238.

[14] L. R. Ford and D. R. Fulkerson: Maximal flow through a network. *Canadian Journal of Mathematics*, **8** (1956), 399–404.

[15] L. R. Ford and D. R. Fulkerson: *Flows in Networks* (Princeton University Press, Princeton, 1962).

[16] S. Fujishige: A maximum flow algorithm using MA ordering. *Operations Research Letters*, **31** (2003), 176–178.

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

[18] N. Garg and J. Könemann: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *Proceedings of the 39th Annual Symposium on Foundations of Computer Science* (1998), 300–309.

[19] F. Glover, J. Hultz, D. Klingman and J. Stutz: Generalized networks: a fundamental computer-based planning tool. *Management Science*, **24** (1978), 1209–1220.

[20] F. Glover and D. Klingman: On the equivalence of some generalized network problems to pure network problems. *Mathematical Programming*, **4** (1973), 269–278.

[21] F. Glover, D. Klingman and N. Phillips: Netform modeling and applications. *Interfaces*, **20** (1990), 7–27.

[22] F. Glover, D. Klingman and N. Phillips: *Network Models in Optimization and Their Applications in Practice* (Wiley, New York, 1992).

[23] A. V. Goldberg, S. A. Plotkin and É. Tardos: Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, **16** (1991), 351–381.

[24] A. V. Goldberg, É. Tardos and R. E. Tarjan: Network flow algorithms. In B. Korte, L. Lovász. H. J. Prömel and A. Schrijver (eds.) : *Flows, Paths and VLSI* (Springer, Berlin, 1990), 101–164.

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

[26] A. V. Goldberg and R. E. Tarjan: Finding mimimum-cost circulations by canceling negative cycles. *Journal of the Association for Computing Machinery*, **36** (1989), 388–397.

[27] A. V. Goldberg and R. E. Tarjan: Finding minimum-cost circulation by successive approximation. *Mathematics of Operations Research*, **15** (1990), 430–466.

[28] D. Goldfarb and Z. Jin: A polynomial dual simplex algorithm for the generalized circulation problem. *Technical Report*, (Columbia University, 1994)

[29] D. Goldfarb and Z. Jin: A faster combinatorial algorithm for the generalized circulation problem. *Mathematics of Operations Research*, **21** (1996), 529–539.

[30] D. Goldfarb and Z. Jin: A new scaling algorithm for the minimum cost network flow problem. *Operations Research Letters*, **25** (1999), 205–211.

[31] D. Goldfarb, Z. Jin and Y. Lin: A polynomial dual simplex algorithm for the generalized circulation problem. *Mathematical Programming*, **91** (2002), 271–288.

[32] D. Goldfarb, Z. Jin and J. B. Orlin: Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Mathematics of Operations Research*, **22** (1997), 793–802.

[33] D. Goldfarb and Y. Lin: Combinatorial interior point methods for generalized network flow problems. *Mathematical Programming*, **93** (2002), 227–246.

[34] M. Gondran and M. Minoux: *Graphs and Algorithms* (J. Wiley & Sons, New York, 1984).

[35] W. S. Jewell: Optimal flow through networks. *Technical Report 8* (M.I.T, 1958).

[36] W. S. Jewell: Optimal flow through networks with gains. *Operations Research*, **10** (1962), 476–499.

[37] D. Jungnickel: *Graphs, Networks and Algorithms* (Springer, Berlin, 1999).

[38] S. Kapoor and P. M. Vaidya: Speeding up Karmarkar's algorithm for multicommodity flows. *Mathematical Programming*, **73** (1996), 111–127.

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

[40] B. Korte and J. Vygen: *Combinatorial Optimization* (Springer, Berlin, 2000).

[41] S. M. Murray: An interior point approach to the generalized flow problem with costs and related problems. *Ph. D thesis* (Stanford University, 1993).

[42] A. Nakayama and C. F. Su: Two efficient algorithms for the generalized maximum balanced flow problem. *Journal of Operations Research Society of Japan*, **45** (2002), 162–173.

[43] J. D. Oldham: Combinatorial approximation algorithms for generalized flow problems. *Journal of Algorithms*, **38** (2001), 135–169.

[44] K. Onaga: Dynamic programming of optimal flows in lossy communication nets. *IEEE Transactions on Circuit Theory*, **13** (1966), 308–327.

[45] K. Onaga: Optimal flows in general communication networks. *Journal of the Franklin Institute*, **283** (1967), 308–327.

[46] J. B. Orlin: A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, **41** (1993), 338–350.

[47] M. Queyranne: Theoretical efficiency of the algorithm 'Capacity' for the maximum flow problem. *Mathematics of Operations Research*, **5** (1980), 258–266.

[48] T. Radzik: Approximate generalized circulation. *Technical Report 93-2* (Cornell University, 1993).

[49] T. Radzik: Faster algorithms for the generalized network flow problem. *Mathematics of Operations Research*, **23** (1998), 69–100.

[50] T. Radzik: Improving time bounds on maximum generalised flow computations by contracting the network. *Theoretical Computer Science*, **312** (2004), 75–97.

[51] T. Radzik and S. Yang: Experimental evaluation of algorithmic solutions for the maximum generalised network flow problem. *Technical Report, TR-01-09* (King's College, 2001).

[52] H. Röck: Scaling techniques for minimal cost network flows. In V. Page (ed.): *Discrete Structures and Algorithms* (Carl Hansen, München, 1980), 101–191.

[53] A. Schrijver: On the history of the transportation and maximum flow problems. *Mathematical Programming*, **91** (2002), 437–445.

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

[55] M. Shigeno: Maximum network flows with concave gains. *Discussion Paper, 1027* (University of Tsukuba, 2003).

[56] M. Shigeno, S. Iwata and S. T. McCormick: Relaxed most negative cycle and most positive cut canceling algorithms for minimum cost flow. *Mathematics of Operations Research*, **25** (2000), 76–104.

[57] P. T. Sokkalingam, R. K. Ahuja and J. B. Orlin: New polynomial-time cycle-canceling algorithms for minimum-cost flows. *Networks*, **36** (2000), 53–63.

[58] É. Tardos and K. D. Wayne: Simple maximum flow algorithms in lossy networks. *Proceedings of 6th International Integer Programming and Combinatorial Optimization Conference, Lecture Notes in Computer Science, vol. 1412*, (Springer, 1998), 310–324.

[59] K. Truemper: Optimal flows in networks with positive gains, *Ph.D Thesis* (Case Western Reserve University, 1973).

[60] K. Truemper: On max flows with gains and pure min-cost flows. *SIAM Journal on Applied Mathematics*, **32** (1977), 450–456.

[61] K. Wayne: Generalized maximum flow algorithms, *Ph. D Thesis* (Cornell University, 1999).

[62] K. Wayne: A polynomial combinatorial algorithm for generalized minimum cost flow. *Mathematics of Operations Research*, **27** (2002), 445–459.

Appendix

We show a brief outline of Radzik's analysis [48] for the maximum-mean-gain cycle-canceling procedure described in Section 3.1, Although Radzik originally gives a bound for the cancel-and-tighten type algorithm, we sketch his analysis for the maximum-mean-gain cycle-canceling procedure.

Let g^0 be a flow output of the maximum-mean-gain cycle-canceling procedure. Since there is no flow-generating cycle in G^{g^0}, we have a label μ^0 such that $\gamma^{\mu^0}(a) \leq 1$ for all $a \in A^{g^0}$. Let $E := \{a \in A \mid \gamma^{\mu^0}(a) > 1\}$. We assume that $E = \{a_1, a_2, \ldots, a_{m'}\}$ and $\gamma^{\mu^0}(a_1) \geq \gamma^{\mu^0}(a_2) \geq \ldots \geq \gamma^{\mu^0}(a_{m'})$. By $k_0, k_1, \ldots, k_P$, denote a strictly increasing maximal sequence of indices of E such that $k_0 = 1$ and

$$\gamma^{\mu^0}(a_{k_p-1}) > (\gamma^{\mu^0}(a_{k_p}) \cdot \gamma^{\mu^0}(a_{k_p+1}) \cdot \ldots \cdot \gamma^{\mu^0}(a_{m'}))^2 \tag{A.1}$$

holds for each $p = 1, 2, \ldots, P$. For convenience, let us assume that $k_{P+1} := m' + 1$. Since, when $k \neq k_1, k_2, \ldots, k_P$, $\gamma^{\mu^0}(a_{k-1}) \leq (\gamma^{\mu^0}(a_k) \cdot \gamma^{\mu^0}(a_{k+1}) \cdot \ldots \cdot \gamma^{\mu^0}(a_{m'}))^2$ holds, we have

$$\gamma^{\mu^0}(a_{k_p}) \cdot \gamma^{\mu^0}(a_{k_p+1}) \cdot \ldots \cdot \gamma^{\mu^0}(a_{m'}) < \left(\gamma^{\mu^0}(a_{k_{p+1}-1})\right)^{\frac{3^{(k_{p+1}-k_p)}}{2}}, \tag{A.2}$$

for any $p = 0, 1, \ldots, P-1$.

We next partition the iterations of the procedure into several intervals of successive iterations $I_0, I_1, \ldots, I_P$, where after the iterations in I_p have ended, the canceled cycles do not contain any arc in $\{a_k \in E \mid 1 \leq k < k_{p+1}\}$ or its reverse arc. We estimate the number of iterations in I_p for any $p = 0, 1, \ldots, P-1$. Let I_p start at the lth iteration and end at the rth iteration. We then have

$$\varepsilon_l < \gamma(C_l) = \gamma^{\mu^0}(C_l) \leq \gamma^{\mu^0}(a_{k_p}) \cdot \ldots \cdot \gamma^{\mu^0}(a_{m'}),$$

where C_l is the cycle canceled at the lth iteration. Moreover, it holds that

$$\varepsilon_r \;\geq\; (\gamma(C_r))^{1/n} = (\gamma^{\mu^0}(C_r))^{1/n} \geq \left(\gamma^{\mu^0}(a_{k_{p+1}-1})/(\gamma^{\mu^0}(a_{k_{p+1}}) \cdot \ldots \cdot \gamma^{\mu^0}(a_{m'}))\right)^{1/n}$$

because the canceled cycle at the rth iteration contains at least one arc in $\{a_k \in E \mid k_p \leq k < k_{p+1}\}$ but do not contain any reverse arc in the set since $g^0(a) = u(a)$ for arcs in E. Combining these relations and Eq.(A.1) and (A.2), we obtain

$$(\varepsilon_r)^{3^{(k_{p+1}-k_p)}} > (\varepsilon_l)^{1/n}.$$

When $p = P$, we similarly obtain the same relation but using the inequalities $\gamma^{\mu^0}(a_{k_P}) \cdot \gamma^{\mu^0}(a_{k_P+1}) \cdot \ldots \cdot \gamma^{\mu^0}(a_{m'}) < \left(\gamma^{\mu^0}(a_{m'})\right)^{3^{(k_{p+1}-k_p)}}$, instead of Eq.(A.2), and $\varepsilon^r \geq (\gamma^{\mu^0}(a_{m'}))^{1/n}$. Hence we have $\varepsilon_s > (\varepsilon_1)^{1/(n^{P+1}3^{m'})} \geq (\varepsilon_1)^{1/(3n)^m}$, when the procedure terminates at the sth iteration. Together with Lemma 3.2, we can conclude that the total iteration number s is bounded by $O(m^2 n \log n)$.

Maiko Shigeno
Graduate School of Systems and Information Engineering
University of Tsukuba
Tsukuba, Ibaraki 305-8573, Japan
E-mail: maiko@sk.tsukuba.ac.jp