Journal of the Operations Research
Society of Japan
2003, Vol. 46, No. 3, 243-250

NEW MAXIMUM FLOW ALGORITHMS BY MA ORDERINGS AND SCALING

Satoru Fujishige Shigueo Isotani
Kyoto University

(Received December 20, 2002)

Abstract Maximum adjacency (MA) ordering has effectively been applied to graph connectivity problems by Nagamochi and Ibaraki. We show an application of MA ordering to the maximum flow problem to get a new polynomial-time algorithm and propose its scaling versions that run in $O(mn \log U)$ time, where m is the number of arcs, n the number of vertices, and U the maximum capacity. We give computational results, comparing our algorithms with those of Goldberg-Tarjan and Dinitz, to show behaviors of our proposed algorithms.

Keywords: Maximum flow, MA ordering, scaling, algorithm

1. Introduction

Maximum adjacency (MA) ordering has effectively been applied to graph connectivity problems by Nagamochi and Ibaraki (see [9, 10]).

One of the authors [5] showed an application of MA ordering to the maximum flow problem to get a new polynomial-time algorithm. For a flow network with n vertices, m arcs, and integral arc capacities $c(a)$ ($\leq U$) our MA ordering algorithm finds a maximum flow by $O(n \log nU)$ augmentations, in $O(n(m + n \log n) \log nU)$ time. A full description and a complexity proof of the algorithm are given in Section 3.

We propose scaling versions of our MA-ordering algorithm in Section 4. The scaling algorithms require $O(mn \log U)$ running time. The complexity is the same as that of Gabow's scaling algorithm [6]. Furthermore, in Section 5 we give computational results, comparing our algorithms with those of Dinitz [3] and Goldberg-Tarjan [7], to show behaviors of our algorithms.

2. Maximum Flow and Residual Network

Let $\mathcal{N} = (G = (V, A), s^+, s^-, c)$ be a flow network, where $G = (V, A)$ is a directed graph with a vertex set V and an arc set A, $s^+ \in V$ an entrance (or a source), $s^- \in V$ an exit (or a sink), and $c : A \to \mathbf{Z}_+$ a capacity function taking on nonnegative integers.

A function $\varphi : A \to \mathbf{Z}_+$ is called a *flow* in $\mathcal{N}$ if it satisfies

(a) (Capacity constraints) $\forall a \in A : \ 0 \leq \varphi(a) \leq c(a)$

(b) (Flow conservation) $\forall v \in V \setminus \{s^+, s^-\} : \ \partial\varphi(v) = 0$, where for each $v \in V$

$$\partial\varphi(v) = \sum_{a=(v,w)\in A} \varphi(a) - \sum_{a=(w,v)\in A} \varphi(a).$$

For a flow φ in $\mathcal{N}$ the *value* of flow φ is defined to be $\partial\varphi(s^+)(= -\partial\varphi(s^-))$ and is denoted by $\hat{v}(\varphi)$. A *maximum flow* is a flow of maximum value.

Given a flow φ in $\mathcal{N}$, a *residual network* $\mathcal{N}_\varphi = (G_\varphi{=}(V, A_\varphi), s^+, s^-, c_\varphi)$ with an underlying graph G_φ and a capacity function $c_\varphi : A_\varphi \to \mathbf{Z}_+$ is defined by

$$A_\varphi = A_\varphi^+ \cup A_\varphi^-, \tag{2.1}$$

$$A_\varphi^+ = \{a \mid a \in A, \varphi(a) < c(a)\}, \tag{2.2}$$

$$A_\varphi^- = \{\bar{a} \mid a \in A,\ 0 < \varphi(a)\} \quad (\bar{a}:\ \text{a reorientation of } a), \tag{2.3}$$

$$c_\varphi(a) = \begin{cases} c(a) - \varphi(a) & (a \in A_\varphi^+) \\ \varphi(\bar{a}) & (a \in A_\varphi^-). \end{cases} \tag{2.4}$$

3. A Maximum Flow Algorithm Using MA Orderings

Suppose that we are given a flow φ in $\mathcal{N}$. For any flow ψ in the residual network $\mathcal{N}_\varphi$ such that $a \in A_\varphi^+$ and $\bar{a} \in A_\varphi^-$ imply $\psi(a) = 0$ or $\psi(\bar{a}) = 0$, we define a flow $\varphi \oplus \psi$ in the original network $\mathcal{N}$ by

$$\varphi \oplus \psi(a) = \begin{cases} \varphi(a) + \psi(a) & \text{if } a \in A_\varphi^+ \text{ and } \psi(a) > 0 \\ \varphi(a) - \psi(\bar{a}) & \text{if } \bar{a} \in A_\varphi^- \text{ and } \psi(\bar{a}) > 0 \\ \varphi(a) & \text{otherwise.} \end{cases} \tag{3.1}$$

The value of the new flow $\varphi \oplus \psi$ in $\mathcal{N}$ increases by the value of ψ in $\mathcal{N}_\varphi$. While ordinary augmenting path algorithms choose an appropriate flow ψ along a single directed path for each augmentation, we will make an augmentation by a multiple-path flow ψ found by an MA ordering.

The argument in this section is found in [5] but we give it here together with some additional remarks for completeness.

An MA ordering from s^+ to s^- in $\mathcal{N}_\varphi$ is obtained as follows.

Procedure MA-Ordering($\mathcal{N}_\varphi$)
Step MA0: Put $i \leftarrow 0$ and $b(u) \leftarrow 0$ for each $u \in V$. Also put $W \leftarrow \{s^+\}$ and $v_0 \leftarrow s^+$. For each $u \in V$ let L_u be an empty list.
Step MA1: For each $w \in V \setminus W$ with $(v_i, w) \in A_\varphi$ put $b(w) \leftarrow b(w) + c_\varphi(v_i, w)$ and add arc (v_i, w) to list L_w.
Step MA2: Let v_{i+1} be a vertex that attains the maximum of $b(w)$ $(w \in V \setminus W)$. If $v_{i+1} = s^-$, then return $(v_0{=}s^+, v_1, \cdots, v_{i+1}{=}s^-)$, b, and L_u $(u \in V)$, and otherwise put $W \leftarrow W \cup \{v_{i+1}\}$ and $i \leftarrow i + 1$ and go to Step MA1.

The time required for Procedure MA-Ordering is $O(m + n \log n)$ by adapting Dijkstra's shortest path algorithm with the Fibonacci heap. It should be noted here that vertex set $U = \{v_0{=}s^+, v_1, \cdots, v_{i+1}{=}s^-\}$ and lists L_u $(u \in U \setminus \{s^+\})$ of in-coming arcs form an acyclic subgraph H_φ of G_φ and that $(v_0{=}s^+, v_1, \cdots, v_{i+1}{=}s^-)$ gives a topological ordering of vertices in H_φ.

Now our MA ordering algorithm for maximum flows is described as follows.

A Maximum Flow Algorithm
Step 0: Put $\varphi(a) \leftarrow 0$ for each $a \in A$.
Step 1: Perform MA-Ordering($\mathcal{N}_\varphi$). Let k be a positive integer such that $v_k = s^-$. Put $\delta \leftarrow \min\{b(v_j) \mid j = 1, 2, \cdots, k\}$. If $\delta = 0$, then return φ (a maximum flow) and otherwise

put $\beta(s^-) \leftarrow \delta$ and $\beta(u) \leftarrow 0$ for each $u \in V \setminus \{s^-\}$.
Step 2: For each $a \in A_\varphi$ put $\psi(a) \leftarrow 0$.
For $i = k,\ k-1,\ \cdots,\ 1$ do the following:
$(*)$ For each arc (u, v_i) in list L_{v_i}
$\quad\psi(u, v_i) \leftarrow \min\{\beta(v_i), c_\varphi(u, v_i)\}$
$\quad\beta(v_i) \leftarrow \beta(v_i) - \psi(u, v_i)$
$\quad\beta(u) \leftarrow \beta(u) + \psi(u, v_i)$
Step 3: Put $\varphi \leftarrow \varphi \oplus \psi$ and go to Step 1.

It should be noted that by the definition of δ computed in Step 1 we have $\delta \le b(v_i)$ $(i = 1, 2, \cdots, k)$. Hence in Step 2 $\beta(v_i) \le \delta \le b(v_i)$, which makes it possible to compute ψ in Step 2 such that $\partial\psi(v_i) = 0$ $(i = 1, 2, \cdots, k-1)$ in $\mathcal{N}_\varphi$. The method of constructing such a flow ψ in the residual graph is similar to an ingredient in that of finding a blocking flow in an acyclic network proposed in [8].

It should also be worth mentioning the following

Lemma 3.1: *If there exists an augmenting path-flow of value α in $\mathcal{N}_\varphi$, then the value of the flow ψ computed in* Step 3 *of our algorithm is greater than or equal to α. In particular, if $\delta = 0$ in* Step 1*, there is no augmenting path in $\mathcal{N}_\varphi$.*

Proof: Let P be a path defining an augmenting path-flow of value α. Then, for each $i = 1, 2, \cdots, k$, putting $W = \{v_0, v_1, \cdots, v_i\}$, there exists an arc (u, w) of P such that $u \in W$ and $w \in V \setminus W$ since $s^+ \in W$ and $s^- \in V \setminus W$. Such an arc (u, w) of P has a capacity $c_\varphi(u, w) \ge \alpha$ by the definition of P. Hence we have $b(w) \ge \alpha$ because of the definition of b. It follows that $b(v_i)$ $(i = 1, 2, \cdots, k)$ computed by MA-Ordering($\mathcal{N}_\varphi$) satisfy $b(v_i) \ge \alpha$.

Now, we examine the complexity of our algorithm. First note that Step 1 requires $\mathrm{O}(m + n\log n)$ time and that Step 2 and Step 3 require $\mathrm{O}(m)$ time. We consider how many times the cycle of Steps 1$\sim$3 is repeated.

Lemma 3.2: *Suppose that $\delta > 0$ in* Step 1*. Then ψ computed in* Step 2 *has a value not less than $(\hat{v}(\varphi^*) - \hat{v}(\varphi))/n$, where φ^* is a maximum flow in $\mathcal{N}$.*

Proof: Suppose that in Step 1 $\delta = b(v_i)$. Then define $W = \{v_0, v_1, \cdots, v_{i-1}\}$. It follows from the definition of v_i that, using the current b when v_i is chosen,

$$\sum\{c_\varphi(u, w) \mid u \in W,\ w \in V \setminus W,\ (u, w) \in A_\varphi\} = \sum_{w \in V \setminus W} b(w) \le |V \setminus W| b(v_i). \tag{3.2}$$

Note that $\hat{v}(\varphi^*) - \hat{v}(\varphi)$ is equal to the value of a maximum flow in the residual network $\mathcal{N}_\varphi$. Hence, from (3.2) and the max-flow min-cut theorem we have

$$\hat{v}(\varphi^*) - \hat{v}(\varphi) \le |V \setminus W| b(v_i) \le n\delta. \tag{3.3}$$

Recall that δ is the value of flow ψ in $\mathcal{N}_\varphi$.

Lemma 3.2 shows that, denoting by $\varphi^{(i)}$ the flow φ computed at the end of the ith execution of Step 3, we have

$$\hat{v}(\varphi^*) - \hat{v}(\varphi^{(i+1)}) \le (1 - \frac{1}{n})(\hat{v}(\varphi^*) - \hat{v}(\varphi^{(i)})). \tag{3.4}$$

This implies that every $\mathrm{O}(n)$ iterations of Steps 1$\sim$3 at least halve the difference $\hat{v}(\varphi^*) - \hat{v}(\varphi)$. Since initially $\hat{v}(\varphi^*) - \hat{v}(\varphi) \le nU - 0$ where U denotes the maximum arc capacity in $\mathcal{N}$ and since φ computed while executing our algorithm is integer-valued, our algorithm finds a maximum flow by repeating Steps 1$\sim$3 $\mathrm{O}(n \log nU)$ times. Hence, we have

Lemma 3.3: *Our MA-ordering max-flow algorithm finds a maximum flow by repeating* Steps 1∼3 $\mathrm{O}(n \log nU)$ *times and requires in total* $\mathrm{O}(n(m + n \log n) \log nU)$ *time.*

Queyranne [11] showed that the maximum-capacity augmenting path algorithm "Capacity" for maximum flows, due to Edmonds and Karp [4], requires $\mathrm{O}(m \log U)$ augmentations (also see [1, Sec. 7.3]). Our MA-ordering algorithm can be regarded as acceleration of "Capacity."

4. Scaling Algorithms

We can also consider a scaling version of our algorithm as follows. Starting from $\delta = U$, instead of performing MA-Ordering we expand W to $W \cup \{v\}$ for $v \in V \setminus W$ if $b(v) \geq \delta$. When W can not be expanded in such a way, the current W is a cut of capacity less than $n\delta$ in $\mathcal{N}_\varphi$. Then we replace δ by $\lfloor \delta/2 \rfloor$ and continue the algorithm with the current W and new δ. We repeat this process until $\delta = 1$. Other part of the scaling algorithm is exactly the same as our original one. The scaling algorithm requires $\mathrm{O}(mn \log U)$ time without using sophisticated data structures such as the Fibonacci heap. This basic scaling version is suggested in [5].

We can further consider a modification of the scaling algorithm as follows. Suppose that in a current scaling phase with parameter δ we cannot expand W. Then, instead of putting $\delta \leftarrow \lfloor \delta/2 \rfloor$, we let $\gamma = \max\{b(v) \mid v \in V \setminus W\}$ and put $\delta \leftarrow \max\{\lfloor \sigma\gamma \rfloor, 1\}$, where σ is an appropriately chosen constant such that $0 < \sigma < 1$. We continue the algorithm with current W and updated δ, and the algorithm terminates after finishing a scaling phase with $\delta = 1$.

If we put $\sigma = 1$ in the modified scaling version of our algorithm, we get an algorithm without explicit scaling. This algorithm can be regarded as a modification of our MA-ordering algorithm without using the Fibonacci heap.

It should also be noted that while expanding W, as soon as we get $b(s^-) \geq \delta$, we choose s^- as a vertex to be added to W. The idea can also be incorporated into our original MA ordering algorithm as follows. If we have $\min\{b(v) \mid v \in W \setminus \{s^+\}\} \leq b(s^-)$, then we should choose s^- and finish MA-Ordering.

5. Computational Results

In this section we describe computational results on our algorithms compared with Dinitz's algorithm and Goldberg and Tarjan's.

5.1. Computional setup

We use a DELL Precision Workstation 330 with an Intel Pentium 4, CPU 1.80GHz, 512 megabytes of memory and running Linux RedHat version 2.4.7. All programs are written in *C* language and compiled with gcc using the -O3 optimization option. Program DF implements Dinitz's algorithm, Program H_PRF Goldberg and Tarjan's algorithm using highest label first criterion, and Program Q_PRF Goldberg and Tarjan's algorithm using a queue to select active vertices. The three programs are the same as used by Cherkassky and Goldberg in their paper [2].

Employing the adjacency list representation of input graphs, we implemented both the original version of our algorithm using MA orderings and its scaling versions. We denote by FMA the program of the original version of our algorithm, which uses the Fibonacci heap to select vertices in MA orderings. Programs FS, FS1/2, and FS4/5 are the scaling versions: the first (FS) replaces δ as $\delta \leftarrow \lfloor \delta/2 \rfloor$ and the others as $\delta \leftarrow \max\{\lfloor \sigma\gamma \rfloor, 1\}$ with $\sigma = 1/2$ for FS1/2, $\sigma = 4/5$ for FS4/5, and $\sigma = 1$ for FS1.

All the running times reported here are in seconds, and we only report the user CPU time, excluding the time required for inputs and outputs. We generated five instances for each problem family of specified size, using different random seeds. Each number shown in the figures is the averaged time over five runs.

5.2. Problem instances

We used the generator GENRMF available from DIMACS Challenge [12] for creating networks based on random seeds that are suggested by DIMACS Core Experiments [12].

The Genrmf family: Each generated network has b grid-like frames of size $(a \times a)$. The number of vertices is a^2b and that of arcs $5a^2b - 4ab - a^2$. All vertices in each frame are connected to its grid neighbors and each vertex is connected by an arc to a vertex randomly chosen from the next frame. Arc capacities within a frame are $c_2 \times a \times a$ and those between frames are randomly chosen integers from the range $[c_1, c_2]$. In our case we set $c_1 = 1$ and $c_2 = 100$. The source vertex is in a corner of the first frame, and the sink is in a corner of the last frame. We used GENRMF to produce two kinds of networks as follows:

- *Genrmf-long.* The number of vertices of a generated graph is $n = 2^x$. The parameters are $a = 2^{x/4}$ and $b = 2^{x/2}$.
- *Genrmf-wide.* The number of vertices of a generated network is $n = 2^x$. The parameters are $a = 2^{2x/5}$ and $b = 2^{x/5}$.

5.3. Experiments

As shown in Figures 1 and 2, we compared different versions of our algorithm on networks of the *Genrmf-long* and *Genrmf-wide* families.

Running time (sec)						
n	m	FMA	FS	FS1/2	FS4/5	FS1
4096	18368	0.374	0.164	0.164	0.130	0.140
7371	33498	0.866	0.400	0.418	0.338	0.332
15488	71687	2.970	1.280	1.468	1.034	1.122
30589	143364	7.848	3.684	3.852	2.654	2.784
65536	311040	27.830	11.162	12.636	9.296	8.958
130682	625537	76.736	32.108	37.136	25.394	22.798
270848	1306607	239.912	104.212	122.786	78.240	71.144

Figure 1: Behaviors of our algorithms on Genrmf-long family data.

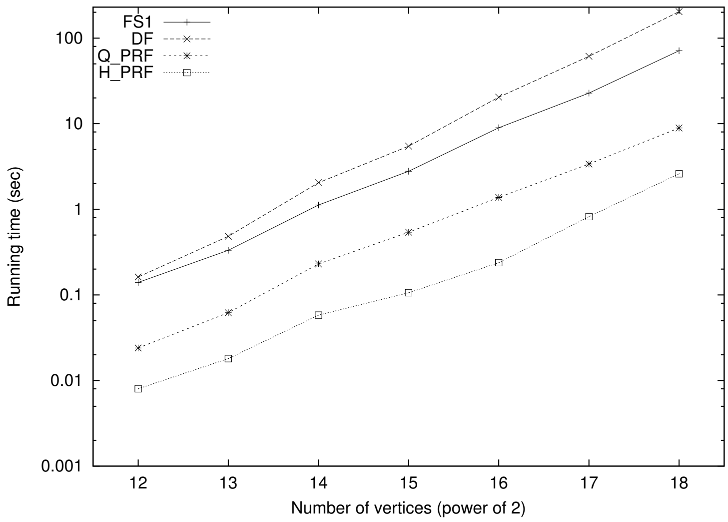
We see from Figures 1 and 2 that the modified implicit scaling version FS1 runs faster than other variants of our algorithm. Our MA-ordering algorithm shows rather poor performance but there is not very substantial difference in efficiency among the variants.

Next, we compared our modified implicit scaling version FS1 with Dinitz's and Goldberg and Tarjan's algorithms on networks of the *Genrmf-long* and *Genrmf-wide* families. Here, we made computational experiments for two variants of the Goldberg-Tarjan push-relabel method. One, denoted by H_PRF, uses the highest label first criterion and the other, denoted by Q_PRF, uses a queue to select active vertices. Except for our algorithms, all implementations were done by the programs due to Cherkassky/Goldberg that could be downloaded from Goldberg's site. Additional information can be found in Cherkassky and Goldberg's

Running time (sec)						
n	m	FMA	FS	FS1/2	FS4/5	FS1
3920	18256	4.654	1.500	1.790	1.182	0.910
8214	38813	18.656	7.068	8.658	5.526	4.262
16807	80262	72.266	29.948	32.366	23.062	17.528
32768	157696	252.964	104.222	112.100	79.328	61.194
65025	314840	929.442	363.894	402.184	285.014	224.758
123210	599289	3184.632	1242.924	1361.766	997.506	745.964
259308	1267875	12430.902	4833.230	4800.926	3628.004	2844.520

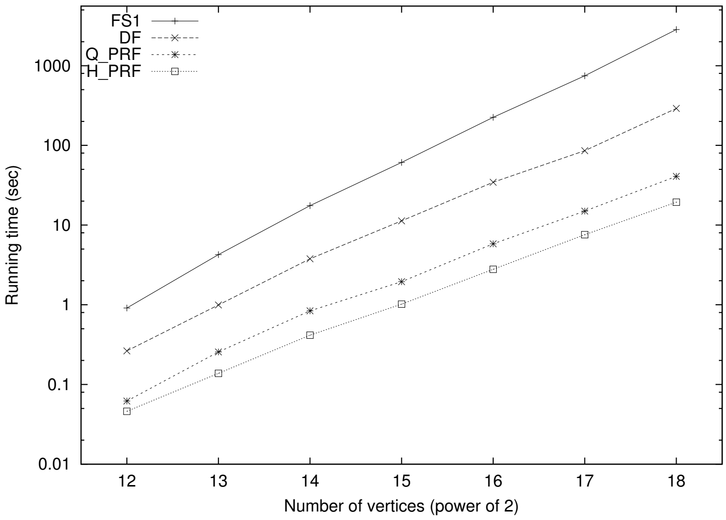
Figure 2: Behaviors of our algorithms on Genrmf-wide family data.

paper [2]. Figure 3 shows results for the *Genrmf-long* family. Our algorithm FS1 was faster than Dinitz's algorithm but was slower than Goldberg and Tarjan's. The results for the *Genrmf-wide* family are shown in Figure 4. Our algorithm showed poor performance on this family data.



Running time (sec)						
n	m	$\log_2 U$	FS1	DF	Q_PRF	H_PRF
4096	18368	12	0.140	0.162	0.024	0.008
7371	33498	12	0.332	0.484	0.062	0.018
15488	71687	13	1.122	2.046	0.230	0.058
30589	143364	14	2.784	5.460	0.540	0.106
65536	311040	14	8.958	20.432	1.376	0.238
130682	625537	15	22.798	61.246	3.378	0.820
270848	1306607	15	71.144	203.572	8.902	2.596

Figure 3: Computational results on Genrmf-long family data.



Running time (sec)						
n	m	$\log_2 U$	FS1	DF	Q_PRF	H_PRF
3920	18256	16	0.910	0.264	0.062	0.046
8214	38813	17	4.262	0.996	0.256	0.138
16807	80262	17	17.528	3.768	0.840	0.416
32768	157696	18	61.194	11.304	1.948	1.018
65025	314840	19	224.758	34.418	5.826	2.786
123210	599289	20	745.964	85.790	14.974	7.586
259308	1267875	21	2844.520	291.006	40.858	19.380

Figure 4: Computational results on Genrmf-wide family data.

6. Concluding Remarks

We have proposed new polynomial-time maximum flow algorithms by MA orderings and scaling and examined behaviors of the proposed algorithms by computational experiments. In our proposed algorithms each flow augmentation is carried out by finding a flow (or a multiple-path) in a residual graph, whereas standard flow algorithms except for push-relabel methods adopt the augmenting single-path approach. This feature is interesting in its own right. Furthermore, our computational results showed that ours ran faster than Dinitz's for the generated Genrmf-long family data. Our proposed algorithms thus seem to be worth further investigation.

Acknowledgments

The present research was carried out while the authors were with Division of Systems Science, Graduate School of Engineering Science, Osaka University and was supported by a Grant-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan and by Japan International Cooperation Agency.

References

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

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

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

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

[5] S. Fujishige: A maximum flow algorithm using MA orderings. *Operations Research Letters* (to appear).

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

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

[8] V. M. Malhotra, M. P. Kumar, and S. N. Maheshwari: An $\mathrm{O}(|V|^3)$ algorithm for finding maximum flows in networks. *Information Processing Letters*, **7** (1978) 277–278.

[9] H. Nagamochi and T. Ibaraki: Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, **5** (1992) 54–66.

[10] H. Nagamochi and T. Ibaraki: Graph connectivity and its augmentation: applications of MA orderings. *Discrete Applied Mathematics*, **123** (2002) 447–472.

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

[12] The First DIMACS international algorithm implementation challenge: The core experiments, 1990. Available at ftp://dimacs.rutgers.edu/pub/netflow/general-info/core.tex .

Satoru Fujishige and Shigueo Isotani
Research Institute for Mathematical Sciences
Kyoto University
Kyoto 606-8502, Japan
E-mail: {fujishig, shigueo}@kurims.kyoto-u.ac.jp