Journal of the Operations Research
Society of Japan
2005, Vol. 48, No. 4, 297-307

# PRACTICAL EFFICIENCY OF MAXIMUM FLOW ALGORITHMS USING MA ORDERINGS AND PREFLOWS

Yuji Matsuoka
*Tokyo University*

Satoru Fujishige
*Kyoto University*

(Received January 17, 2005; Revised April 28, 2005)

*Abstract* Fujishige proposed a polynomial-time maximum flow algorithm using maximum adjacency (MA) orderings. Computational results by Fujishige and Isotani showed that the algorithm was slower in practice than Goldberg and Tarjan's algorithm. In this paper we propose an improved version of Fujishige's algorithm using preflows. Our computational results show that the improved version is much faster than the original one in practice.

**Keywords**: Algorithm, maximum flow, MA ordering, preflow

## 1. Introduction

Maximum adjacency (MA) ordering has effectively been applied to graph connectivitiy problems by Nagamochi and Ibaraki [7, 8]. Fujishige [3] presented an application of MA ordering to the maximum flow problem to devise a new polynomial-time algorithm. For a capacitated network with $n$ vertices, $m$ arcs, and the maximum capacity $U$, Fujishige's algorithm finds a maximum flow in $O(n(m+n\log n)\log nU)$ time. Even under the similarity assumption, this complexity is not the best ruuning time bound for the maximum flow problem. In addition, Shioura [9] proved that the time complexity of Fujishige's algorithm is not strongly polynomial by giving an instance with a real-valued capacity function for which it does not terminate. In practice, computational results in [4] show that Fujishige's algorithm is slower than Goldberg and Tarjan's algorithm [5].

In this paper, we present a new variant of Fujishige's algorithm using preflows. We prove that its complexity is $O(n(m + n\log n)\log nU)$, which is the same as the original one. We compare it with the original version of Fujishige's algorithm and Goldberg and Tarjan's algorithm. Our computational experiments on six problem families reveal that the new version is faster than the original one for all the problem families. In comparison with two codes of Goldberg and Tarjan's algorithm, our algorithm is not so slower than them. We may conclude that the new version of Fujishige's algorithm is practically efficient.

The present paper is organized as follows. Section 2 gives definitions concerning flows and networks. In Section 3 we give a full description of the new version of Fujishige's algorithm. In Section 4 we show computational results comparing the new version with the original version and Goldberg and Tarjan's algorithm. Section 5 provides our conclusion.

## 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. We assume $|V| = n$, the cardinality of $V$.

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

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

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

$$\begin{aligned} A_\varphi &= A_\varphi^+ \cup A_\varphi^-, \\ A_\varphi^+ &= \{a \mid a \in A,\ \varphi(a) < c(a)\}, \\ A_\varphi^- &= \{\bar{a} \mid a \in A,\ 0 < \varphi(a)\} \quad (\bar{a} : \text{a reorientation of } a), \\ c_\varphi(a) &= \begin{cases} c(a) - \varphi(a) & (a \in A_\varphi^+) \\ \varphi(\bar{a}) & (a \in A_\varphi^-). \end{cases} \end{aligned}$$

Suppose that we are given a flow $\varphi$ in $\mathcal{N}$. For any flow $\psi$ 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

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

for each $a \in A$. The value $\hat{v}(\varphi \oplus \psi)$ of the new flow $\varphi \oplus \psi$ in $\mathcal{N}$ is greater than that of $\varphi$ by $\hat{v}(\psi)$.

Preflows will be used in our new version of Fujishige's algorithm. A function $\varphi : A \to \mathbf{Z}_+$ is called a *preflow* in $\mathcal{N}$ if it satisfies

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

(2) (Relaxed flow conservation) $\forall v \in V \setminus \{s^+\} : \partial\varphi(v) \leq 0$,

An *excess* of a preflow $\varphi$ at $v$ is defined by $-\partial\varphi(v)$. We say that a vertex $v$ is *active* if $-\partial\varphi(v) > 0$. For a preflow $\varphi$ in $\mathcal{N}$ we define $\hat{v}(\varphi) = -\partial\varphi(s^-)$. The residual network $\mathcal{N}_\varphi$ for a preflow $\varphi$ is defined in the same way as above.

## 3. A New Version of Fujishige's Algorithm

An MA ordering from an arbitrary node $s \in V$ in $\mathcal{N}_\varphi$ is obtained as follows. Note that here we proceed through each arc backward.

**Procedure** MA-Ordering($\mathcal{N}_\varphi, s$)

**Step 0**: For each $u \in V$, put $b(u) \leftarrow 0$ and let $L_u$ be an empty list.
Put $i \leftarrow 0$, $v_0 \leftarrow s$, $b(v_0) \leftarrow \infty$ and $W \leftarrow \{s\}$.

**Step 1**: For each $w \in V \setminus W$ with $(w, v_i) \in A_\varphi$,
put $b(w) \leftarrow b(w) + c_\varphi(w, v_i)$ and add arc $(w, v_i)$ to list $L_w$.

**Step 2**: Let $v_{i+1}$ be a vertex that attains the maximum of $b(w)$ $(w \in V \setminus W)$.

If $b(v_{i+1}) = 0$ (there is no vertex in $V \setminus W$ which is reachable to $s$)
or $W = V$ (there is no vertex to choose),
then return $(v_0(= s), v_1, \cdots, v_i)$, $b$, and $L_u(u \in W)$.
Otherwise, put $W \leftarrow W \cup \{v_{i+1}\}$, $i \leftarrow i + 1$, and go to Step 1.

The complexity of Procedure MA-Ordering is $O(m+n \log n)$ if we use a Fibonacci heap. Let $W$ be the vertex set $\{v_0(= s), v_1, \cdots, v_k\}$ obtained by this procedure, then $W$ corresponds to the set of vertices that are reachable to $s$ along directed paths in $G_\varphi$. It should also be noted here that vertex set $W$ and lists $L_w(w \in W \setminus \{s\})$ of out-going arcs form an acyclic subgraph, denoted by $H_\varphi$, of $G_\varphi$ and that obtained ordering $(v_0(= s), v_1, \cdots, v_k)$ gives a topological ordering of vertices in $H_\varphi$.

Goldberg and Tarjan's push-relabel algorithm using preflows [5] shows great efficiency in practice [2]. Their algorithm keeps a preflow and a valid distance label and repeatedly performs local push operations on the current preflow and relabel operations to update the distance label. On the other hand, in our algorithm we keep a preflow and perform push operations according to a linear ordering of vertices computed by Procedure MA-ordering for a current residual network. We repeat this process until we obtain a preflow of maximum value. The obtained maximum preflow is transformed into a maximum flow by pushing excess flows back to source $s^+$.

Now we describe the new MA-ordering maximum-flow algorithm using preflows as follows.

## A New Version of the MA-Ordering Maximum-Flow Algorithm

We compute a preflow of maximum value in Step 1 (the cycle of Steps 1-1 and 1-2), and convert it into a flow of maximum value in Step 2 (the cycle of Steps 2-1 and 2-2). It should be noted here that the maximum flow value in $\mathcal{N}$ is equal to the maximum preflow value in $\mathcal{N}$. Therefore, we can get the maximum flow value and a minimum cut by performing only Step 1.

### Procedure FMAP

**Step 0**: (Preflow Initialization)
For each $a = (s^+, u) \in A$, put $\varphi(a) \leftarrow c(a)$.
For each arc $a = (v, w) \in A$ with $v \neq s^+$, put $\varphi(a) \leftarrow 0$.

**Step 1-1**: (Obtaining MA-Ordering from $s^-$)
Perform MA-Ordering($\mathcal{N}_\varphi, s^-$) and get $(v_0(= s^-), v_1, \cdots, v_k)$.
If $\partial\varphi(v_i) = 0$ for all $i = 1, \cdots, k$, then go to Step 2-1 (the current $\varphi$ is a preflow of maximum value).

**Step 1-2**: (Pushing preflows to $s^-$)
For $i = k, k-1, \cdots, 1$ do the following:
For each arc $(v_i, u)$ in list $L_{v_i}$, push $(v_i, u)$:
If $(v_i, u) \in A^+_\varphi$ then $\varphi(v_i, u) \leftarrow \varphi(v_i, u) + \min\{-\partial\varphi(v_i), c_\varphi(v_i, u)\}$,
If $(v_i, u) \in A^-_\varphi$ then $\varphi(u, v_i) \leftarrow \varphi(u, v_i) - \min\{-\partial\varphi(v_i), c_\varphi(v_i, u)\}$.
Go to Step 1-1.

**Step 2-1**: (Obtaining MA-Ordering from $s^+$)
Perform MA-Ordering($\mathcal{N}_\varphi, s^+$) and get $(v_0(= s^+), v_1, \cdots, v_k)$.
If $\partial\varphi(v) = 0$ for all $v \in V \setminus \{s^+, s^-\}$, return $\varphi$ (a maximum flow).

**Step 2-2**: (Pushing excess flows back to $s^+$)
For $i = k, k-1, \cdots, 1$ do the following if $v_i \neq s^-$:

For each arc $(v_i, u)$ in list $L_{v_i}$, push $(v_i, u)$:
If $(v_i, u) \in A_\varphi^+$ then $\varphi(v_i, u) \leftarrow \varphi(v_i, u) + \min\{-\partial\varphi(v_i), c_\varphi(v_i, u)\}$,
If $(v_i, u) \in A_\varphi^-$ then $\varphi(u, v_i) \leftarrow \varphi(u, v_i) - \min\{-\partial\varphi(v_i), c_\varphi(v_i, u)\}$.
Go to Step 2-1.

We first remark that during the procedure the computed $\varphi$ remains to be a preflow. That is, in push operations we maintain the following conditions for $\varphi$:
(1) (Capacity constraints) $\forall a \in A : 0 \leq \varphi(a) \leq c(a)$.
(2) (Relaxed flow conservation) $\forall v \in V \setminus \{s^+\} : \partial\varphi(v) \leq 0$.

Step 1 repeatedly performs MA-Ordering and push operations. When there are no active vertices that are reachable to $s^-$, the iteration of Step 1 terminates. Then obtained $\varphi$ has the following property:

**Lemma 3.1.** *When we finish the iteration of* Step 1, *then preflow $\varphi$ is of maximum value.*

*Proof.* For the preflow $\varphi$, we have $\partial\varphi(v) = 0$ for any vertex $v$ that is reachable to $s^-$. Then let $(v_0(= s^-), v_1, \cdots, v_k)$ be the MA-ordering of vertices obtained at the last iteration of Step 1 and define $W = \{v_0(= s^-), v_1, \cdots, v_k\}$. For the current preflow $\varphi$ and a cut $V \setminus W$ we have

$$\hat{v}(\varphi) = -\partial\varphi(s^-) = \sum_{j=0}^{k}(-\partial\varphi(v_j)) = \kappa(W)$$

where $\kappa(W) = \sum\{c(u, v) \mid (u, v) \in A,\ u \in V \setminus W,\ v \in W\}$. The max-flow min-cut theorem implies that $\hat{v}(\varphi)$ attains the maximum value among values of all preflows in $\mathcal{N}$. □

This lemma shows that we have both a preflow $\varphi$ of maximum value and a minimum cut $V \setminus W$ in $\mathcal{N}$ when finishing Step 1.

In Step 2 we convert the preflow of maximum value into a flow of maximum value. When $\partial\varphi(v) = 0$ for all $v \in V \setminus \{s^+, s^-\}$, the iteration of Step 2 terminates. It implies that the computed $\varphi$ satisfies the flow conservation condition and is a flow of the maximum value.

Now, we examine the complexity of the algorithm. First note that Step 2 is at most the same complexity as Step 1, so we only have to examine Step 1. Since Step 1-1 requires $O(m + n \log n)$ time and Step 1-2 requires $O(m)$ time, each iteration of Step 1 requires $O(m + n \log n)$ time. The following lemma tells us how many times Step 1 is repeated.

**Lemma 3.2.** *Let $\varphi$ be a preflow available immediately before the execution of* Step 1-2 *and let $\tilde{\varphi}$ be a preflow obtained after the execution of* Step 1-2, *then the increased preflow value $\hat{v}(\tilde{\varphi}) - \hat{v}(\varphi)$ is more than $(\hat{v}^* - \hat{v}(\varphi))/n$, where $\hat{v}^*$ is the maximum flow value (the maximum preflow value) in $\mathcal{N}$.*

*Proof.* If there exists no active vertex after finishing an execution of Step 1-2, $\tilde{\varphi}$ is a preflow of the maximum value and we have $\hat{v}(\tilde{\varphi}) = \hat{v}^*$.

Otherwise let $l$ be the smallest index such that $\partial\tilde{\varphi}(v_j) < 0$ $(j = 1, 2, \cdots, k)$. Define $W_l = \{v_0, v_1, \cdots, v_{l-1}\}$, and let $b(v_j) = \sum\{c_\varphi(v_j, u) \mid u \in W_l\}$ $(j = l, l+1, \cdots, k)$, the value $b(v_j)$ when $v_l$ is chosen in MA-Ordering. Since $\partial\tilde{\varphi}(v_l) < 0$, the amount of preflow pushed from $v_l$ toward $s^-$ is $b(v_l)$. Moreover, as we have $\partial\tilde{\varphi}(v_j) = 0$ $(j = 1, 2, \cdots, l-1)$, there is no preflow excess at $v_j$ $(j = 1, 2, \cdots, l-1)$. These two observations imply that the flow value

increased by the execution of Step 1-2 satisfies

$$\hat{v}(\tilde{\varphi}) - \hat{v}(\varphi) \geq b(v_l) + \sum_{j=1}^{l-1}(-\partial\varphi(v_j)).$$

It follows from the definition of an MA ordering that

$$\kappa_\varphi(W_l) = \sum_{j=l}^{k} b(v_j) \leq (k-l+1)b(v_l),$$

where $\kappa_\varphi(W_l) = \sum\{c_\varphi(u,v) \mid (u,v) \in A_\varphi,\ u \in V \setminus W_l,\ v \in W_l\}$. On the other hand, since $\varphi$ is a preflow in $\mathcal{N}$, we have

$$\hat{v}^* \leq \kappa(W_l) = \kappa_\varphi(W_l) + \sum_{j=0}^{l-1}(-\partial\varphi(v_j)),$$

using the max-flow min-cut theorem. It follows from the above three inequalities that

$$\begin{aligned}\hat{v}^* - \hat{v}(\varphi) &\leq (k-l+1)b(v_l) + \sum_{j=1}^{l-1}(-\partial\varphi(v_j)) \\ &\leq nb(v_l) + n\sum_{j=1}^{l-1}(-\partial\varphi(v_j)) \\ &\leq n(\hat{v}(\tilde{\varphi}) - \hat{v}(\varphi)).\end{aligned}$$

□

Lemma 3.2 shows that

$$\hat{v}^* - \hat{v}(\varphi^{(i+1)}) \leq (1 - \frac{1}{n})(\hat{v}^* - \hat{v}(\varphi^{(i)})),$$

where $\varphi^{(i)}$ denotes the preflow $\varphi$ computed at the end of the $i$th execution of Step 1-2. This implies that every $O(n)$ iterations of Step 1 (the cycle of Step 1-1 to Step 1-2) at least halve the difference $\hat{v}^* - \hat{v}(\varphi)$. Since initially we have $\hat{v}^* - \hat{v}(\varphi) \leq nU - 0$ where $U$ denotes the maximum arc capacity, and since $\varphi$ computed while executing our algorithm is integer-valued, our algorithm finds a maximum flow by repeating Step 1 $O(n \log nU)$ times. Hence, we have the following theorem.

**Theorem 3.3.** *Our algorithm finds a maximum flow after* $O(n \log nU)$ *iterations of* Step 1 *and* Step 2. *Hence the complexity of our algorithm is* $O(n(m + n \log n) \log nU)$.

Note that the complexity of our algorithm is the same as the original version of Fujishige's [3]. While Fujishige's algorithm does not terminate for the instance network of a real-valued capacity function shown by Shioura [9], our algorithm finds a maximum flow after five iterations for the instance. However, any better estimation of the complexity of our algorithm proposed here has not been found yet.

## 4. Computational Results

In this section we describe computational results on our new version of Fujishige's algorithm, comparing it with the original version and Goldberg and Tarjan's algorithm.

### 4.1. Computational setup

Our experiments were conducted on TOSHIBA WXPHESP1 JP001 with an Intel Pentium M, CPU 1.30 GHz, 512 megabytes of memory and running Microsoft Windows XP Home Edition verson 2002. All programs are written in C language and compiled with the gcc using the -O3 optimization option. Program *FMA* implements the original version of Fujishige's algorithm. While the program in [4] used a Fibonacci heap as the data structure, we use an ordinary (non-Fibonacci) heap for practical efficiency. Program *Q_PRF* is Goldberg and Tarjan's algorithm using a queue to select active vertices, which is the same as used by Cherkassky and Goldberg in [2]. Program *HI_PR* is a new and more robust version of Goldberg and Tarjan's algorithm using the highest-label-first criterion, and is derived from H_PRF which was also used in [2]. We used HI_PR of version 3.5 in the experiments.

All the running times reported here are in seconds, and we only report the user CPU time, excluding the input and output time. Except for the AK family explained below, we generated five instances for each problem family of specified size, using different random seeds. Each number in the figures is the averaged time over five runs.

### 4.2. Problem instances

We used six problem families, which were produced by three generators: GENRMF, WASHINGTON, and AK. These generators are available from DIMACS [6].

**GENRMF Family** The GENRMF generator produces networks consisting of $b$ 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 $[c_1, c_2]$. 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 three kinds of networks as follows:

- GENRMF-LONG family: The number of vertices in a generated network is $n = 2^x$. We set parameters as $a = 2^{x/4}$, $b = 2^{x/2}$, $c_1 = 1$ and $c_2 = 10000$.
- GENRMF-LONGER family: The number of vertices in a generated network is $n = 2^x$. We set parameters as $a = 4$, $b = 2^{x-4}$, $c_1 = 1$ and $c_2 = 10000$.
- GENRMF-WIDE family: The number of vertices in a generated network is $n = 2^x$. We set parameters as $a = 2^{2x/5}$, $b = 2^{x/5}$, $c_1 = 1$ and $c_2 = 10000$.

**WASHINGTON Family** The WASHINGTON generator generates random level graphs with $a$ rows and $b$ columns. The number of vertices is $ab + 2$ and that of arcs is $3ab - b$. For each column except for the last one, every vertex is connected to three randomly chosen vertices in the next column. The source vertex is connected to every vertex in the first row, and the sink vertex to every vertex in the last row. Capacities of the connecting arcs are randomly chosen integers from the range $[1, c]$. Capacities of the source and sink arcs are from the range $[1, 3c]$. We used WASHINGTON to gererate two families as follows:

- WASHINGTON-RLG-LONG family: The number of vertices in a generated network is $n = 2^x$. We set parameters as $a = 64$, $b = 2^{x-6}$ and $c = 10000$.

- WASHINGTON-RLG-WIDE family: The number of vertices in a generated network is $n = 2^x$. We set parameters as $a = 2^{x-6}$, $b = 64$ and $c = 10000$.

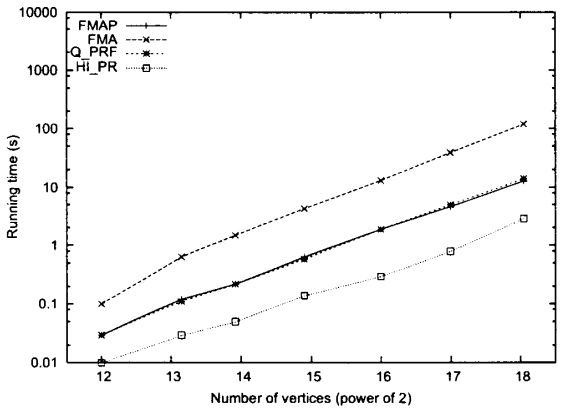
**AK Family** The AK generator produces the problem families that are hard for Goldberg and Tarjan's push-relabel algorithms. Generated networks are deterministic for each value of $n$. The details for generated networks are described in [2].

- AK family: The number of vertices in a gererated network is $n = 2^x$.

### 4.3. Experiments

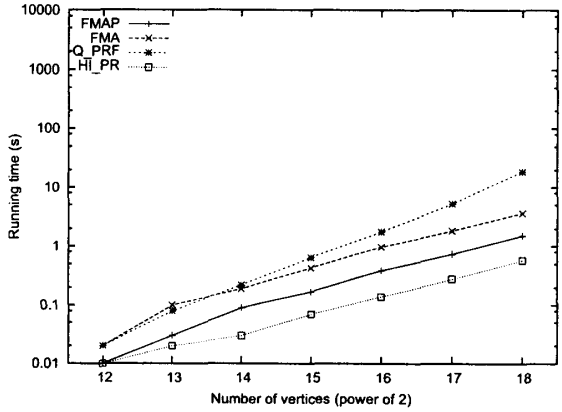
To examine practical efficiency of our proposed algorithm, we implemented it by using the adjacency list representation of input graphs. For data structures in MA orderings, we chose an ordinary heap to select vertices of maximum $b(w)$ and maintained the list $L_w$ as a queue. We denote this program by $FMAP$. We also used an ordinary heap instead of a Fibonacci heap for the original version of Fujishige's algorithm.

We made computatinal experiments for the following four programs: FMA, FMAP, Q_PRF, HI_PR. Our results are shown in Figures 1~6, one for each family.



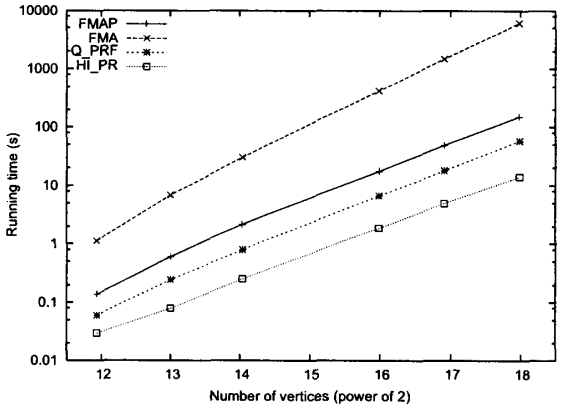

| input data | | | Running time(s) | | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $\log_2 U$ | FMA | FMAP | Q_PRF | HI_PR |
| 4096 | 18368 | 19.3 | 0.10 | 0.03 | 0.03 | 0.01 |
| 9100 | 41760 | 19.9 | 0.64 | 0.13 | 0.11 | 0.03 |
| 15488 | 71687 | 20.2 | 1.48 | 0.27 | 0.22 | 0.05 |
| 30589 | 143364 | 20.7 | 4.30 | 0.72 | 0.59 | 0.14 |
| 65536 | 311040 | 21.3 | 13.27 | 2.11 | 1.87 | 0.30 |
| 130682 | 625537 | 21.8 | 39.11 | 4.93 | 5.03 | 0.80 |
| 270848 | 1306607 | 22.3 | 119.76 | 14.60 | 14.21 | 2.87 |

Figure 1: Computational results on GENRMF-LONG family data



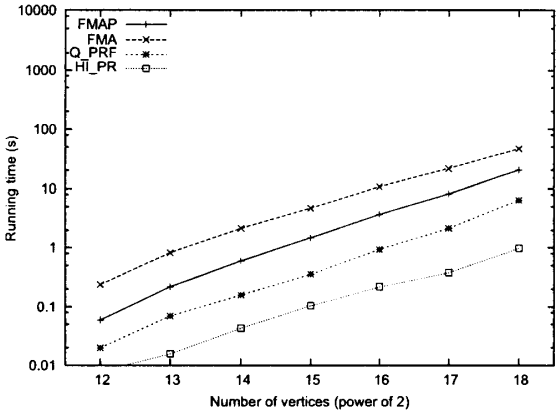

| input data | | | Running time(s) | | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $\log_2 U$ | FMA | FMAP | Q_PRF | HI_PR |
| 4096 | 16368 | 17.3 | 0.02 | 0.01 | 0.02 | 0.01 |
| 8192 | 32752 | 17.3 | 0.10 | 0.03 | 0.08 | 0.02 |
| 16384 | 65520 | 17.3 | 0.19 | 0.09 | 0.22 | 0.03 |
| 32768 | 131056 | 17.3 | 0.43 | 0.17 | 0.64 | 0.07 |
| 65536 | 262128 | 17.3 | 0.98 | 0.39 | 1.78 | 0.14 |
| 131072 | 524272 | 17.3 | 1.85 | 0.74 | 5.26 | 0.28 |
| 262144 | 1048560 | 17.3 | 3.72 | 1.56 | 19.17 | 0.59 |

Figure 2: Computational results on GENRMF-LONGER family data



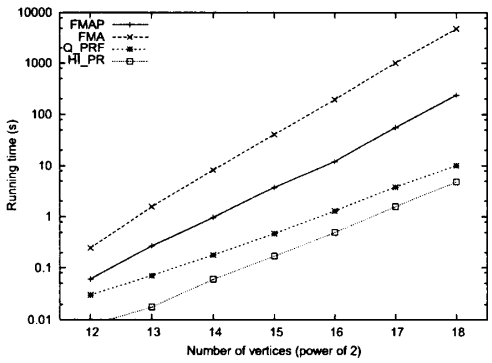

| input data | | | Running time(s) | | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $\log_2 U$ | FMA | FMAP | Q_PRF | HI_PR |
| 3920 | 18256 | 22.9 | 1.13 | 0.14 | 0.06 | 0.03 |
| 8214 | 38813 | 23.7 | 6.93 | 0.61 | 0.25 | 0.08 |
| 16807 | 80262 | 24.5 | 30.33 | 2.17 | 0.80 | 0.26 |
| 65025 | 314840 | 26.1 | 425.72 | 17.89 | 6.78 | 1.89 |
| 123210 | 599289 | 26.9 | 1503.79 | 49.21 | 18.33 | 5.03 |
| 259308 | 1267875 | 27.7 | 6118.04 | 151.70 | 58.28 | 14.33 |

Figure 3: Computational results on GENRMF-WIDE family data



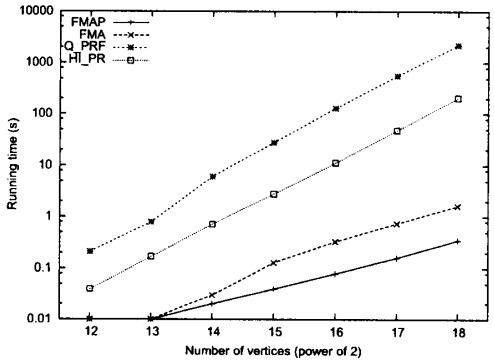

| input data | | | Running time(s) | | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $\log_2 U$ | FMA | FMAP | Q_PRF | HI_PR |
| 4098 | 12224 | 14.8 | 0.24 | 0.06 | 0.02 | 0.01 |
| 8194 | 24512 | 14.8 | 0.83 | 0.22 | 0.07 | 0.02 |
| 16386 | 49088 | 14.8 | 2.17 | 0.60 | 0.16 | 0.04 |
| 32770 | 98240 | 14.8 | 4.76 | 1.49 | 0.36 | 0.11 |
| 65538 | 196544 | 14.8 | 10.89 | 3.71 | 0.94 | 0.22 |
| 131074 | 391168 | 14.8 | 21.51 | 8.28 | 2.17 | 0.39 |
| 262146 | 786368 | 14.8 | 46.94 | 20.28 | 6.47 | 0.99 |

Figure 4: Computational results on WASHINGTON-RLG-LONG family data




| input data | | | Running time(s) | | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $\log_2 U$ | FMA | FMAP | Q_PRF | HI_PR |
| 4098 | 12224 | 14.8 | 0.25 | 0.06 | 0.03 | 0.01 |
| 8194 | 24512 | 14.8 | 1.56 | 0.27 | 0.07 | 0.02 |
| 16386 | 49088 | 14.8 | 8.34 | 0.98 | 0.18 | 0.06 |
| 32770 | 98240 | 14.8 | 40.87 | 3.74 | 0.48 | 0.17 |
| 65538 | 196544 | 14.8 | 195.73 | 12.29 | 1.30 | 0.51 |
| 131074 | 391168 | 14.8 | 1017.21 | 55.76 | 3.80 | 1.59 |
| 262146 | 786368 | 14.8 | 4916.26 | 238.85 | 10.17 | 4.88 |

Figure 5: Computational results on WASHINGTON-RLG-WIDE family data




| input data | | | Running time(s) | | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $\log_2 U$ | FMA | FMAP | Q_PRF | HI_PR |
| 4102 | 6151 | 19.9 | 0.01 | 0.01 | 0.21 | 0.04 |
| 8198 | 12295 | 19.9 | 0.01 | 0.01 | 0.79 | 0.17 |
| 16390 | 24583 | 19.9 | 0.03 | 0.01 | 6.10 | 0.72 |
| 32774 | 49159 | 19.9 | 0.13 | 0.03 | 27.94 | 2.83 |
| 65542 | 98311 | 19.9 | 0.33 | 0.07 | 129.26 | 11.32 |
| 131078 | 196615 | 19.9 | 0.74 | 0.16 | 563.24 | 48.58 |
| 262150 | 393223 | 19.9 | 1.62 | 0.35 | 2167.15 | 206.87 |

Figure 6: Computational results on AK family data

Figure 1 shows results for the GENRMF-LONG family. The new version is faster than the original version and is almost as fast as Q_PRF.

Figure 2 shows results for the GENRMF-LONGER family. Our proposed algorithm is faster than Q_PRF.

Figure 3 shows results for the GENRMF-WIDE family. The new version is much faster than the original version. However it is slower than both codes of Goldberg and Tarjan's algorithm.

Figure 4 shows results for the WASHINGTON-RLG-LONG family. The new version is slower than both codes of Goldberg and Tarjan's algorithm.

Figure 5 shows results for the WASHINGTON-RLG-WIDE family. For this family our proposed algorithm performs much better than the original version.

Figure 6 shows results for the AK family. For this special data family our proposed algorithm outperforms the others.

Our experiments show that the new version is faster than the original version for all the problem instances given above. Our proposed algorithm outperformed the two codes of Goldberg and Tarjan's algorithm for one problem families: AK family. For the other families the proposed algorithm is not so slower than Goldberg and Tarjan's. The computational results show that our algorithm is practically efficient.

## 5. Conclusion

We have presented an improved version of Fujishige's algorithm using preflows and showed its behavior by giving computational results. The improved version is faster than the original version for all problem instances of our experiments. While Goldberg and Tarjan's algorithm maintains a locally valid order of vertices and performs local push operations, our improved algorithm performs global orderings and global push operations.

It is left for future work to examine whether a better running time bound of our improved algorithm exists or whether our algorithm is strongly polynomial.

## Acknowledgements

We are very grateful to Satoru Iwata for his valuable discussions and useful comments throughout this research. The present work is partly supported by a Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan.

## References

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

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

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

[4] S. Fujishige and S. Isotani: New maximum flow algorithms by MA orderings and scaling. *Journal of the Operations Research Society of Japan,* **46** (2003), 243–250.

[5] 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.

[6] D.S. Johnson and C.C. McGeoch: *Network Flows and Matching: First DIMACS Implementation Challenge* (AMS, Rhode Island, 1993).

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

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

[9] A. Shioura: The MA-ordering max-flow algorithm is not strongly polynomial for directed networks. *Operations Research Letters,* **32** (2004), 31–35.

[10] A.V. Goldberg: Synthetic maximum flow families. Available at http://www.avglab.com/andrew/CATS/maxflow_synthetic.htm .

Yuji Matsuoka
Department of Mathematical Informatics,
Graduate School of Information Science and Technology,
University of Tokyo
Bunkyo, Tokyo 113-8656, Japan
E-mail: yuji_matsuoka@mist.i.u-tokyo.ac.jp