Technical Report #94011

Implementing an Efficient Minimum Capacity Cut Algorithm [1]

Hiroshi NAGAMOCHI, Tadashi ONO
and
Toshihide IBARAKI

June 1994

Kyoto University, Kyoto, Japan 606-01
Netaddress: naga@kuamp.kyoto-u.ac.jp

Abstract In this paper, we present an efficient implementation of the $O(mn + n^2 \log n)$ time algorithm originally proposed by Nagamochi and Ibaraki (1992) for computing the minimum capacity cut of an undirected network. To enhance computation, various ideas are added so that it can contract as many edges as possible in each iteration. To evaluate the performance of the resulting implementation, we conducted extensive computational experiment, and compared the results with that of Padberg and Rinaldi's algorithm (1990), which is currently known as one of the practically fastest programs for this problem. The results indicate that our program is considerably faster than Padberg and Rinaldi's program, and its running time is not significantly affected by the types of the networks being solved.

Key words: minimum capacity cut, network flow, polynomial algorithm

[1] appeared in Mathematical Programming, vol 67, 1994, pp. 325-341.

1 Introduction

Let network $\mathcal{N} = (G = (V, E), c)$ be a simple undirected graph G with a set V of vertices and a set E of edges, weighted by a capacity function $c : E \to \mathcal{R}^+$ (the set of nonnegative reals). We assume without loss of generality that G is connected. The minimum capacity cut problem is to find a nonempty and proper subset U of V that minimizes the total capacity of the edges between U and $V - U$. The minimum capacity cut problem is a fundamental problem in network optimization, and has a wide variety of applications such as analysis of road and communication networks, VLSI designs, facet identification for the traveling salesman problem, and so on.

Gomory and Hu [4] show that a minimum cut of $\mathcal{N}$ can be obtained by solving $|V| - 1$ maximum flow problems. One of the currently best time bounds for solving the maximum flow problem is $O(mn \log(n^2/m))$ due to Goldberg and Tarjan [3], where $n = |V|$ and $m = |E|$, and hence the time complexity of Gomory and Hu's method is $O(mn^2 \log(n^2/m))$. Recently, Hao and Orlin [5] showed that $|V|-1$ maximum flow problems to compute a minimum cut can be solved in amortized $O(mn \log(n^2/m))$ time by implementing Goldberg and Tarjan's maximum flow algorithm efficiently. An $O(n^2 \log^3 n)$ time randomized algorithm that computes a minimum cut with high probability is also proposed [6].

Padberg and Rinaldi [9] present an improvement of Gomory and Hu's method. Their method repeats the contraction of two vertices into a single vertex to obtain a smaller network. It may require $n - 1$ maximum flow computations in the worst case (and hence its worst time complexity is still $O(mn^2 \log(n^2/m))$), since a contraction may require the knowledge of the minimum capacity cut between the contracted vertices (which is obtained by solving the maximum flow problem between them). However, they also provide simple alternative tests to contract a pair of vertices, by which the number of maximum flow computations may be considerably reduced. Their algorithm is currently considered to be one of the fastest one in the practical sense.

In this paper, we present an efficient implementation of an $O(mn + n^2 \log n)$ algorithm for the minimum capacity cut problem recently proposed by Nagamochi and Ibaraki [8]. This algorithm does not rely on the maximum flow computation, but uses a different test to contract vertices. We add various mechanisms to this algorithm so that it can contract as many edges as possible in each iteration. We conducted an extensive numerical experiment to evaluate the performance of the proposed implementation, and compared it with the Padberg and Rinaldi's. The results indicate that our program is much faster and more stable than the Padberg and Rinaldi's for a wide variety of networks.

2 Preliminaries

Let $\mathcal{N}$ be defined as above. We denote $e = (u, v)$ if the end vertices of edge $e \in E$ are u and v. A nonempty and proper subset U of V is called a *cut*, and the capacity of a cut U is defined by

$$c(U) = \sum_{(u,v) \in E, u \in U, v \in V-U} c(u, v). \tag{1}$$

A cut U is said to separate vertices u and v if $u \in U$ and $v \in V - U$, or $u \in V - U$ and $v \in U$. For notational convenience, a singleton set $\{u\}$ may also be written as u. For example, $c(v)$ denotes the sum of capacities of those edges which are incident to v. The *minimum capacity cut problem* is the problem of finding a cut U that minimizes $c(U)$ in $\mathcal{N}$. We call such U a *minimum cut* and denote by $\lambda(\mathcal{N})$ the capacity of a minimum cut of $\mathcal{N}$.

For $x, y \in V$, the minimum capacity among the cuts separating x and y is denoted $\lambda(x, y)$. From the maximum flow minimum cut theorem by Ford and Fulkerson [1], we can obtain $\lambda(x, y)$ by computing the maximum flow between x and y.

The contraction of an edge $e = (u, v)$ of $\mathcal{N}$ is defined as follows: merge u and v into a single vertex after deleting edge $e = (u, v)$, and merge the resulting multiple edges with the same pair of end vertices into a single edge whose capacity is equal to the sum of capacities of such edges. We denote the network obtained by contracting all edges in a subset $F \subseteq E$ as $\mathcal{N}' = (G/F, c')$, where G/F is the graph obtained from G by this contraction and c' is the capacity function of the resulting network. It is easy to see that $\mathcal{N}'$ is independent of the order of edges in F to be contracted.

3 Contracting Edges by CONTRACT

3.1 Computing the Minimum Cut by Edge Contractions

In this section, we describe a general framework to compute $\lambda(\mathcal{N})$ by using edge contractions.

Lemma 1 For a network $\mathcal{N} = (G, c)$ and an edge $e = (x, y) \in E$, the contracted network $\mathcal{N}' = (G/e, c')$ satisfies

$$\lambda(\mathcal{N}) = \min\{\lambda(\mathcal{N}'), \lambda(x, y)\}. \tag{2}$$

Proof. Generally we have $\lambda(\mathcal{N}) \leq \lambda(\mathcal{N}')$. If a minimum cut of $\mathcal{N}$ does not separate x and y, it is easily seen that this minimum cut is still present in $\mathcal{N}'$, and hence $\lambda(\mathcal{N}) = \lambda(\mathcal{N}')$. If a minimum cut separates x and y, its capacity is $\lambda(x, y)$, and thus $\lambda(\mathcal{N}) = \lambda(x, y)$. Hence (2) holds. □

Lemma 2 Let $\overline{\lambda}$ be the capacity of a cut $X \subset V$ in $\mathcal{N}$. If there exist two vertices x and y satisfying

$$\lambda(x, y) \geq \overline{\lambda}, \tag{3}$$

then $\mathcal{N}$ has no cut that separates x and y, and has capacity less than $\overline{\lambda}$.

Proof. Immediate from the definition of $\lambda(x, y)$. □

If condition (3) holds for $\overline{\lambda}$ and $e = (x, y)$, we have

$$\lambda(\mathcal{N}) = \min\{\lambda(\mathcal{N}'), \overline{\lambda}\} \tag{4}$$

for the network $\mathcal{N}'$ obtained from $\mathcal{N}$ by contracting (x, y). This indicates that computing $\lambda(\mathcal{N})$ is reduced to computing $\lambda(\mathcal{N}')$ of a smaller network $\mathcal{N}'$ once such $\overline{\lambda}$ and e are found.

Definition 1 If an edge $e = (x, y)$ satisfies condition (3) for the capacity $\overline{\lambda}$ of a cut, we call e *contractible* (with respect to $\overline{\lambda}$). □

Based on this observation, we can compute $\lambda(\mathcal{N})$ in the following manner. First examine all $c(v) = \sum_{(v,u)\in E,\ u\neq v} c(v,u),\ v \in V$ in the given network $\mathcal{N}$, and store the smallest $c(v)$ as $\overline{\lambda}$ (recall that a singleton set $\{v\}$ is one of the cuts). This can be done in linear time $O(m)$. Second, find an edge $e = (x, y)$ satisfying (3) for the current $\overline{\lambda}$, contract (x, y) into a vertex x', and then set $\overline{\lambda} := \min\{\overline{\lambda}, c'(x')\}$. When we repeat this contraction operation until the number of vertices becomes two, the final $\overline{\lambda}$ provides us with the minimum capacity $\lambda(\mathcal{N})$ of the original network $\mathcal{N}$.

This procedure is described as follows.

Procedure CONTRACT
Input: a simple connected undirected network $\mathcal{N} = (G, c)$;
Output: $\lambda(\mathcal{N})$;
1 **begin**
2 $\overline{\lambda} := \min\{c(v) \mid v \in V\}$;
3 $\mathcal{N}' := \mathcal{N}$ and denote $\mathcal{N}'$ by $(G' = (V', E'), c')$;
4 **while** $|V'| \geq 3$ **do**
5 **begin**
6 **if** a cut $X \subset V$ with $c'(X) < \overline{\lambda}$ is found **then** $\overline{\lambda} := c'(X)$;
7 Find an edge $e = (x, y)$ such that $\lambda(x, y) \geq \overline{\lambda}$;
8 Contract (x, y) into x' and let $\mathcal{N}' = (G' = (V', E'), c')$ be the resulting network;
9 $\overline{\lambda} := \min\{\overline{\lambda}, c'(x')\}$
10 **end**;
11 Conclude that $\lambda(\mathcal{N}) = \overline{\lambda}$
12 **end.**

The test in line 6 is introduced since a smaller $\overline{\lambda}$ strengthens the operation of line 7, and hence reduces the computation time. Note that the property

$$\overline{\lambda} \leq \min_{v\in V'} c'(v) \tag{5}$$

is maintained in the resulting network $\mathcal{N}'$ after each contraction.

Although the details of lines 6 and 7 are not specified at this moment, we note that the minimum cut capacity can be obtained after performing at most $n - 1$ operations of edge contraction. The existence of a contractible edge in line 7 is for example guaranteed if we compute $\lambda(x, y)$ for a pair of vertices x and y (by the maximum flow algorithm). If the capacity of a minimum cut X separating x and y is smaller than $\overline{\lambda}$, then execute line 6 by using this cut. Otherwise skip line 6. In any case, $\lambda(x, y) \geq \overline{\lambda}$ always holds in line 7, and (x, y) is contractible.

However, it may not be necessary to rely on the maximum flow algorithm to find a contractible edge. Padberg and Rinaldi's algorithm [9] tries to find an edge $e = (x, y)$ with $\lambda(x, y) \geq \overline{\lambda}$ by simple alternative tests, which we call PR tests (see Appendix, for details), and uses the maximum flow algorithm only when the PR tests fail. The algorithm by Nagamochi

and Ibaraki [8] never resorts to the maximum flow algorithm, but applies a different kind of test to find at least one contractible edge, as will be described in the next subsection.

We also note that, by maintaining the cut that attains the current $\overline{\lambda}$, CONTRACT can be modified so that it finds not only the minimum capacity $\lambda(\mathcal{N})$ but also the minimum cut that realizes $\lambda(\mathcal{N})$. For notational simplicity, however, algorithms in this paper will be described only for computing $\lambda(\mathcal{N})$.

3.2 Algorithm CAPFOREST

Nagamochi and Ibaraki [8] presented an algorithm CAPFOREST that computes a lower bound $q(e)$ on $\lambda(x,y)$ for every edge $e=(x,y)\in E$. It is a graph traversing algorithm that, starting from an arbitrarily given vertex, visits all the vertices in V in the order such that the next vertex v to visit maximizes the value $r(v)$, which is the sum of capacities of the edges between v and the vertices already visited. Each edge $e=(x,y)$ is said to be scanned and given a real $q(e)$ when one of its end vertices, say x, is visited. The $q(e)$ is set to be the sum of capacities of the edges between the other end vertex y and the vertices already visited. The entire procedure is described as follows.

Algorithm CAPFOREST $(\mathcal{N};q)$
Input: a simple connected undirected network $\mathcal{N}=(G=(V,E),c)$;
Output: the lower bounds $q(e)$ on $\lambda(x,y)$, $e=(x,y)\in E$;
begin
  Label all vertices $v\in V$ "unvisited" and all edges $e\in E$ "unscanned";
  $r(v):=0$ for all $v\in V$;
  while there exists an "unvisited" vertex **do**
    begin
      Choose an "unvisited" vertex x with the largest r;
      for each vertex y adjacent to x by an "unscanned" edge $e=(x,y)$ **do**
        begin
          $r(y):=r(y)+c(e)$;
          $q(e):=r(y)$;
          Mark e "scanned"
        end;
      Mark x "visited"
    end;
end.

CAPFOREST can be executed in $O(m+n\log n)$ time [8]. The next property for $q(e)$ is essential for our purposes (see [2, 7] for other properties related to graph connectivity).

Lemma 3 [8] (i) The $q(e)$ obtained by CAPFOREST satisfies

$$q(e)\le\lambda(x,y),\qquad e=(x,y)\in E. \tag{6}$$

(ii) Let $e^*=(x^*,y^*)$ be the last edge scanned in CAPFOREST. Then this e^* satisfies

$$q(e^*) = \lambda(x^*, y^*). \tag{7}$$

Furthermore, the last vertex y^* visited in CAPFOREST satisfies $c(y^*) = q(e^*)$. □

Note that the second statement in (ii) follows from how $q(e)$'s are determined in CAPFOREST. By property (5), the $\lambda(x^*, y^*)$ in (ii) of Lemma 3 always satisfies

$$\lambda(x^*, y^*) = q(e^*) = c(y^*) \geq \overline{\lambda},$$

and hence e^* is contractible in line 7 of CONTRACT. Based on this observation, we have the following algorithm to compute $\lambda(\mathcal{N})$, which is a slightly modified version of the original algorithm of [8].

Algorithm CONTRACT/CAP
Input: a simple connected undirected network $\mathcal{N} = (G = (V, E), c)$;
Output: $\lambda(\mathcal{N})$;

```
 1 begin
 2    λ̄ := min{c(v) | v ∈ V};
 3    N' := N and denote N' by (G' = (V', E'), c');
 4    while |V'| ≥ 3 do
 5       begin
 6          Call CAPFOREST (N'; q);
 7          while there exists an edge e with q(e) ≥ λ̄ do
 8             begin
 9                Contract e = (x, y) into a vertex x' and let N' = (G' = (V', E'), c') be the
                  resulting network; {when two edges with forms e' = (z, x), e'' = (z, y) are
                  merged into ē = (z, x') by this contraction, set q(ē) := max{q(e'), q(e'')}
                  so that q(ē) still plays a lower bound on λ(z, x').}
10                λ̄ := min{c(x'), λ̄}
11             end;
12       end;
13    Conclude that λ(N) = λ̄
14 end.
```

At every execution of CAPFOREST in CONTRACT/CAP, at least one edge $e^* = (x^*, y^*)$ is contractible, and there may also be other contractible edges. Therefore, CONTRACT/CAP terminates after executing CAPFOREST at most $n - 2$ times (usually much less), and thus its running time is $O(mn + n^2 \log n)$.

3.3 Modification of CAPFOREST

We modify CAPFOREST into the following MCAP (modified CAPFOREST) to facilitate the computation of CONTRACT/CAP. The modification consists of the following two points.

1. In CONTRACT/CAP, the **while** loop of lines 7-11 contracts all edges e with $q(e) \geq \overline{\lambda}$ to obtain $G/E(\overline{\lambda})$, where $E(\overline{\lambda}) = \{e \in E \mid q(e) \geq \overline{\lambda}\}$. However, if we take a maximal spanning forest T of the subgraph

$$G_{\overline{\lambda}} = (V, E(\overline{\lambda})), \tag{8}$$

it is easy to see that $G/T = G/E(\overline{\lambda})$ holds. By property $|T| \leq |E(\overline{\lambda})|$, it is computationally advantageous to contract T instead of $E(\overline{\lambda})$. For this purpose, we modify CAPFOREST so that such T is obtained during its execution at no extra cost.

2. In the early stage of computation, CAPFOREST tends to visit those vertices that are mutually connected by edges with relatively large capacities. Therefore, when it happens to select a vertex y with a relatively small $r(y)$, which is the sum of capacities between y and the set X of already visited vertices, X has a chance of being a small cut. To make use of this property, we keep track of the minimum cut capacity α among such cuts X. If a small cut is found and $\overline{\lambda}$ is set to a small value, it tends to enlarge the set of contractible edges, thereby facilitating the entire computation of CONTRACT.

Algorithm MCAP $(\mathcal{N}, \overline{\lambda}; q, T, \alpha)$
Input: a simple connected undirected network $\mathcal{N} = (G = (V, E), c)$, and a capacity $\overline{\lambda} > 0$ of the currently known minimum cut;
Output: lower bounds $q(e)$ for $e \in E$, a forest T of contractible edges, and a cut capacity α as explained in 2. above;

```
 1 begin
 2    Label all vertices v ∈ V "unvisited" and all edges e ∈ E "unscanned";
 3    r(v) := 0 for all v ∈ V;
 4    α(v) := 0 for all v ∈ V; T := ∅; α' := 0;
 5    while there exists an "unvisited" vertex do
 6       begin
 7          Choose an "unvisited" vertex x with the largest r;
 8          α' := α' + c(x) − 2r(x);
 9          α(x) := α';
10          for each vertex y adjacent to x by an "unscanned" edge e = (x, y) do
11             begin
12                if r(y) < λ̄ ≤ r(y) + c(e) then T := T ∪ {e};
13                r(y) := r(y) + c(e);
14                q(e) := r(y);
15                Mark e "scanned"
16             end;
17          Mark x "visited"
18       end;
19    α := min{α(v) | v ∈ V − x*}, where x* is the vertex visited lastly.
20 end.
```

The modifications from CAPFOREST consist in the addition of lines 4,8,9,12 and 19. The output $q(e)$ are obviously the same as those obtained in CAPFOREST.

We call the execution of lines 6-18 for an unvisited vertex x as the cycle for x. Let x_i $(1 \leq i \leq n)$ be the i-th vertex visited by MCAP, and let $X_i = \{x_1, \ldots, x_i\}$.

Lemma 4 $\alpha(x_i) = c(X_i)$ $(1 \leq i \leq n-1)$ holds after the cycle for $x = x_i$.

Proof. Since $r(x_1)$ remains 0 in the cycle for $x = x_1$, $\alpha(x_1) = c(x_1) - 2r(x_1) = c(x_1) = c(X_1)$ holds. Assuming that $\alpha(x_i) = c(X_i)$ holds after the cycle for $x = x_i$ $(1 \leq i < n)$, we show that $\alpha(x_{i+1}) = c(X_{i+1})$ holds after the cycle for $x = x_{i+1}$. Since $r(x_{i+1}) = \sum_{e=(x_h,x_{i+1})\in E,\ 1\leq h\leq i} c(e)$ holds in the cycle for $x = x_{i+1}$, by the definition of r, we have

$$\begin{aligned} c(X_{i+1}) &= c(X_i) - \sum_{e=(x_h,x_{i+1})\in E,\ 1\leq h\leq i} c(e) + \sum_{e=(x_{i+1},x_k)\in E,\ i+2\leq k\leq n} c(e) \\ &= c(X_i) - r(x_{i+1}) + (c(x_{i+1}) - r(x_{i+1})) \\ &= c(X_i) + c(x_{i+1}) - 2r(x_{i+1}). \end{aligned}$$

By noting that α' stores $c(X_i)$ in the beginning of the cycle for $x = x_{i+1}$, we see that $\alpha(x_{i+1}) = c(X_{i+1})$ holds after executing line 9 in the cycle for $x = x_{i+1}$. □

Lemma 5 The edge set T obtained by MCAP is a maximal spanning forest in $G_{\overline{\lambda}}$ of (8).

Proof. Define:

$$\begin{aligned} E_i &= \{(x_h, x_i) \in E \mid 1 \leq h < i\}, \\ r_h(v) &= \text{the value of } r(v) \text{ when the cycle for } x = x_h \text{ is completed.} \end{aligned}$$

Note that each edge $e = (x_h, x_i)$, $h < i$, is assigned value $q(e) = r_h(x_i) = r_{h-1}(x_i) + c(e)$ when the cycle for $x = x_h$ is completed. This e is added to T if $r_{h-1}(x_i) < \overline{\lambda} \leq r_h(x_i)$ holds.

(i) We first show that T obtained by MCAP is a forest in G. For this, it is sufficient to prove that

$$|E_i \cap T| \leq 1 \quad \text{for all } i = 2, \ldots, n, \tag{9}$$

because, if T contained a circuit C, then there would be a vertex x_i with $|E_i \cap C| \geq 2$. Now note that, for a given i, $r_h(x_i)$ never decreases with $h = 1, 2, \ldots, i-1$ and it increases as $r_h(x_i) = r_{h-1}(x_i) + c(e)$ only when $e = (x_h, x_i) \in E_i$ are scanned. Therefore, given a positive constant $\overline{\lambda}$ and an index i,

$$r_{h-1}(x_i) < \overline{\lambda} \leq r_h(x_i)$$

holds for at most one h among $h = 1, 2, \ldots, i-1$, and only this $(x_h, x_i) \in E_i$ is added to T. This proves (9).

(ii) We show the maximality of T in G. For any edge $e = (x_s, x_t) \in E(\overline{\lambda}) - T$, where $s < t$ is assumed without loss of generality, we show that there is a path from x_s to x_t in T. (This proves the maximality of T.) For this, assume otherwise and let $x_{s'}$ (resp. $x_{t'}$) be the vertex with the smallest index among those connected from x_s (resp. x_t) by a path $P_s \subseteq T$ (resp. $P_t \subseteq T$). We first show $t' < s$. Since e with $q(e) = r_s(x_t) \geq \overline{\lambda}$ was not added to T, it means $r_{s-1}(x_t) \geq \overline{\lambda}$ and hence there exists an edge $(x_h, x_t) \in T$ with $h \leq s-1$. From this, we have $t' \leq h < s$.

Then assume $s' < t'$, i.e., P_s must contains an edge $e' = (x_a, x_b)$ with $a < t' < b$ to skip t'. However, this is impossible because $r_{t'-1}(x_b) \geq r_a(x_b) = q(e') \geq \overline{\lambda}$ and $E_{t'} \cap T = \emptyset$ (i.e., $r_{t'-1}(x_{t'}) < \overline{\lambda}$) imply that x_b must be visited before $x_{t'}$ (i.e., $b < t'$) by the selection rule of MCAP, a contradiction.

The case of $t' < s'$ can be analogously treated. It also leads to a contradiction. □

3.4 Algorithm CONTRACT/MCAP

By replacing CAPFOREST in CONTRACT/CAP with MCAP, we obtain the following algorithm.

Algorithm CONTRACT/MCAP
Input: a simple undirected network $\mathcal{N} = (G = (V, E), c)$;
Output: $\lambda(\mathcal{N})$;

```
 1 begin
 2    λ̄ := min{c(v) | v ∈ V};
 3    N' := N and denote N' by (G' = (V', E'), c');
 4    while |V'| ≥ 3 do
 5       begin
 6          Call MCAP (N', λ̄; q, T, α);
 7          λ̄ := min{α, λ̄};
 8          for e ∈ T do
 9             begin
10                Contract e into a vertex x' to obtain N' = (G' = (V', E'), c');
11                λ̄ := min{c(x'), λ̄}
12             end;
13       end;
14    Conclude that λ(N) = λ̄
15 end.
```

During the execution of MCAP, it would also be possible to update $\overline{\lambda}$ by

$$\overline{\lambda} := \min\{\overline{\lambda}, \alpha(x_i)\} \tag{10}$$

whenever $\alpha(x_i)$ is computed at line 9 of MCAP, for the purpose of constructing a forest T' which is larger than the above T. However, it turned out by our experiment that such modification did not improve the efficiency because $|T'|$ rarely become larger than $|T|$ but operation (10) required certain amount of time.

4 Computational Experiments

In this section, we report some results from our computational experiment to evaluate the performance of CONTRACT/MCAP. All computer programs were coded in FORTRAN 77 in double precision and run on a workstation SUN SPARC 1 IPX.

4.1 Generation of Sample Networks

In our experiment, we specify network types by the following four parameters:

- the number of vertices n,

- the density of edges $d = \dfrac{2m}{n(n-1)} \times 100$ (%), where m is the number of edges, i.e., $m = n(n-1)(d/200)$,
- a decomposition number k ($1 \leq k \leq n$), and
- a constant p satisfying $0 < p \leq 1$ (for $k \geq 2$).

Based on these parameters, we generate a network as follows: First, we construct a connected graph with n vertices and $m = n(n-1)(d/200)$ edges by randomly choosing pairs of vertices as edges, where $n-1$ out of m edges are initially specified to create a Hamilton path to make the graph connected. Capacity of each edge is randomly chosen from interval $[0, 100)$.

As the name of parameter k suggests, the generated networks tend to have k tightly connected components since, in case of $k \geq 2$, the capacities of edges are determined in the following manner. We divide vertices into k subsets randomly (each of the subsets is called a cluster), and the capacities of the edges between different clusters are scaled down by factor $0 < p \leq 1$, while the capacities of the edges connecting two vertices in the same cluster remain unchanged. Obviously, if $p = 1$ (i.e., no clusters in effect), a singleton cut $X = \{x\}$ for some vertex x usually becomes a minimum cut, but if a small p is chosen, a union X of some clusters is likely to form a minimum cut in the generated network. In our experiments, p is set to $1/n$ (except in Figure 6), so that this property is usually guaranteed. (Because the expected cut capacity between clusters is at most $d\lfloor\frac{n}{2}\rfloor\lceil\frac{n}{2}\rceil p/100 \approx nd/400$ in this case, while the expected cut capacity of a singleton is $d(n-1)/100$.)

4.2 Computational Results

For the networks generated in the above manner, we compare the average running times of

- Padberg and Rinaldi's algorithm (PR algorithm, for short),
- the proposed algorithm CONTRACT/MCAP, and
- a single maximum flow computation by the algorithm of Goldberg and Tarjan [3] (MAXFLOW, for short), which is included here only for the reference purposes,

where each data point represents the average of ten problem instances.

First, Figures 1 and 2 show the CPU time for various values of n with density $d = 50$ (%), and $k = 1$ and $k = 2$ respectively. From these figures, we can see that CONTRACT/MCAP is much faster than PR, and the difference grows with n. When $k = 2$, PR takes longer time than the case of $k = 1$, because the network loses uniformity of capacity distribution and it becomes hard to find contractible edges by the PR tests of PR. Contrary to this, CONTRACT/MCAP accelerates contraction process, as a result of introducing α. The difference between CONTRACT/MCAP and PR becomes greater when k is changed from 1 to 2.

Figures 3 and 4 illustrate the CPU time for different densities d, corresponding to $k = 1$ and $k = 2$, respectively. In these figures, the CPU time of PR fluctuates with the density. In PR, the probability of finding contractible edges by the PR tests tends to become small when the density is low, and hence the number of resorting to maximum flow computation

increases. However, when density is high (i.e., d is close to 100), the simple PR tests are very effective. For this reason, PR becomes slightly faster than CONTRACT/MCAP for the case of $d = 100$ (%) and $k = 1$ (Figure 3). However, for $k = 2$ (Figure 4), CONTRACT/MCAP always run faster than PR.

Figure 5 shows the results when the decomposition number k changes for the networks with $n = 400$. For relatively small k, an edge whose both end vertices are in the same cluster is not likely contractible by the PR tests (as described in Appendix, only an edge with large capacity relative to the adjacent edges is likely to satisfy the condition of the PR tests). If $k \geq 50$, however, the capacities distribute nearly uniformly in the sense that a small number of edges with relatively large capacities are scattered over the entire network, and such edges can be contracted by the PR tests in an early stage of the algorithm. This results in a high performance of PR. On the contrary, CONTRACT/MCAP is little influenced by the decomposition number k and always remains faster than PR.

Figure 6 shows the effect of parameter p in the range $[0.0005, 1]$ for the networks with $n = 400$ and $k = 2$. Note that when $p = 1$, there is only one cluster (i.e., equivalent to the case of $k = 1$). When p takes intermediate values, e.g., 0.005 (i.e., $p = 2/n$) to 0.1, singleton cuts and non-singleton cut that separates two clusters have the same chance of being a minimum cut, since the expected capacity $c(x)$ of a vertex x is approximately equal to the capacity of one cluster. In this situation, the PR tests rarely finds contractible edges, and PR becomes rather slow. However, CONTRACT/MCAP is always faster than PR, and is almost independent of p.

From the above observation, we conclude that CONTRACT/MCAP is much faster than PR, and its performance is almost independent of the types of the networks being solved. Its running time is comparable to 2-3 executions of maximum flow algorithm. Another advantage of CONTRACT/MCAP consists in its simplicity of implementation, while PR involves the maximum flow algorithm and a sophisticated control of the PR tests.

4.3 A Hybrid Implementation: HYBRID

Since PR runs faster than CONTRACT/MCAP for a special case of $k = 1$ and $d \geq 95$ (%), we propose a modified algorithm HYBRID, a hybrid version of CONTRACT/MCAP and PR. It is obtained by incorporating a part of the PR tests into CONTRACT/MCAP.

A crucial point here is that, although we want to capture the power of the PR tests, which are particularly effective for the above types of problem instances, we also have to take care not to increase the running time of CONTRACT/MCAP for other types of problem instances. To attain this, after line 12 of CONTRACT/MCAP (i.e., after contracting all the edges in T), we apply the PR tests only for the edges having relatively large capacities [2] among those incident to the vertex x^* contracted lastly. For each T, we continue this test as long as the PR tests successfully find contractible edges, and once it fails, we move to the next T by restarting MCAP.

Algorithm HYBRID

[2] we employed $c(e) \geq 0.7c(x^*)/|V'|$ as a criterion of relatively large capacity, where $|V'|$ is the current number of vertices. Note that $c(x^*)/|V'|$ is approximately the average capacity among the edges incident to x^*, if the network is very dense, and can be computed in $O(1)$ time.

```
Input: a simple undirected network N = (G = (V, E), c);
Output: λ(N);
begin
    λ̄ := min{c(v) | v ∈ V};
    N' := N and denote N' by (G' = (V', E'), c');
    while |V'| ≥ 3 do
        begin
            Execute MCAP (N', λ̄; q, T, α);
            λ̄ := min{α, λ̄};
            for e ∈ T do
                begin
                    Contract e into x' to obtain N' = (G' = (V', E'), c');
                    λ̄ := min{c'(x'), λ̄};
                    x* := x'
                end;
            if |V'| ≥ 3 then prtest:=true;
            while prtest do
                begin
                    Find an edge e with c(e) ≥ 0.7c(x*)/|V'| among the edges incident to x*;
                    Apply the PR tests to e (see (11) – (14) of Appendix);
                    if e is contractible then
                        Contract e into x' to obtain N' = (G' = (V', E'), c');
                        λ̄ := min{c'(x'), λ̄};
                        x* := x';
                        if |V'| ≤ 2 then prtest:=false
                    else prtest:=false
                end{while prtest}
        end{while |V'| ≥ 3};
    Conclude that λ(N) = λ̄
end.
```

This modification turns out to be effective for problem instances such as $k = 1$ and $d \geq 95$ (%) (see Fig. 3). It is also observed that HYBRID performs almost the same as CONTRACT/MCAP for other types of problem instances. The results of HYBRID are also indicated in Figs. 1-6.

5 Conclusion

By revising some part of the algorithm proposed by Nagamochi and Ibaraki [8], we obtained a practically efficient program for computing the minimum cut capacity in undirected networks. From the obtained computational results, we see that CONTRACT/MCAP is faster than PR (proposed by Padberg and Rinaldi [9]) in most cases except for those networks that have very high edge density and uniform capacities. The performance of CONTRACT/MCAP is very stable and almost independent of the network types specified by parameters k and p. We

also proposed a hybrid implementation by including some PR tests into CONTRACT/MCAP, and showed that the resulting algorithm HYBRID is never slower than any of the PR and CONTRACT/MCAP for all types of networks we tested in this study. Finally, we emphasize that an important advantage of CONTRACT/MCAP consists in its simplicity.

Acknowledgement
We are grateful to Professor Giovanni Rinaldi for providing us a FORTRAN program of algorithm PR. This work was partially supported by the Grant in Aid by the Ministry of Education, Science and Culture of Japan.

References

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

[2] A. Frank, T. Ibaraki and H. Nagamochi: "On sparse subgraphs preserving connectivity properties," *J. Graph Theory* 17 (1993) 275-281.

[3] A. Goldberg and T. Tarjan: "A new approach to the maximum flow problem," *J. ACM* 35 (1988) 921-940.

[4] R. Gomory and T. Hu: "Multi-terminal network flows," *SIAM Journal on Applied Mathematics* 9 (1961) 551-570.

[5] J. Hao and J.B. Orlin: "A faster algorithm for finding the minimum cut in a graph," *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, Orlando, Florida (1992) 165-174.

[6] D.V. Karger and C. Stein: "An $\tilde{O}(n^2)$ algorithm for minimum cuts," *Proc. 25th ACM Symp. Theory of Computing*, San Diego, California (1993) 757-765.

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

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

[9] M. Padberg and G. Rinaldi: "An efficient algorithm for the minimum capacity cut problem," *Mathematical Programming* 47 (1990) 19-36.

Appendix: Simple Tests in Padberg and Rinaldi's Algorithm

Padberg and Rinaldi [9] derived the following two types of sufficient conditions for a pair of vertices x, y to be contractible.

Lemma 6 If $x, y \in V$ satisfy one of the following conditions,
(i)

$$c(x) \leq 2c(x,y) \quad \text{or} \quad c(y) \leq 2c(x,y), \tag{11}$$

(ii)

$$c(x,y) \geq \overline{\lambda}, \tag{12}$$

then either $\{x\}$ or $\{y\}$ is a minimum cut, or there exists a minimum cut X such that $x, y \in X$.
□

By maintaining $c(x)$ and $c(x,y)$ as explicit data for all $x \in V$ and $(x,y) \in E$, conditions (11) and (12) can be examined for a pair of x and y in $O(1)$ time.

Let $N(u)$, $u \in V$, be the neighbor set of u defined by,

$$N(u) = \{v \in V - \{u\} \mid (u,v) \in E\}.$$

The following lemma provides another condition for contractibility.

Lemma 7 If $x, y \in V$ satisfy one of the following conditions,
(i) There is a $z \in N(x) \cap N(y)$ such that

$$c(x) \leq 2\{c(x,y) + c(x,z)\} \quad \text{and} \quad c(y) \leq 2\{c(x,y) + c(y,z)\}, \tag{13}$$

(ii)

$$c(x,y) + \sum_{z \in N(x) \cap N(y)} \min\{c(x,z), c(y,z)\} \geq \overline{\lambda} \tag{14}$$

then either $\{x\}$ or $\{y\}$ is a minimum cut or there exists a minimum cut X such that $x, y \in X$.
□

Conditions (13) and (14) for a pair of $x, y \in V$ can be checked in $O(n)$ time.

At the beginning of PR, $\lambda(x,y)$ for an edge $(x,y) \in E$ is computed by using the maximum flow algorithm, and the two vertices x and y are contracted into one vertex x'. Then the edges incident to x' are pushed into a stack. These edges in the stack have large capacities relative to other edges, and are likely to satisfy the conditions in Lemmas 6 and 7. While the stack is nonempty, an edge $e = (x,y)$ is popped up to check the conditions (11)-(14). If the test succeeds, x, y will be contracted and the edges incident to the resulting vertex will be added to the stack. Only when the stack becomes empty[3], $\lambda(x,y)$ for an appropriately chosen pair of vertices is directly computed by using the maximum flow algorithm, in order to continue the contraction process.

[3]To be precise, not only when the stack becomes empty, but also when the number of consecutive failures of the test exceeds a specified number, the maximum flow algorithm is invoked in PR.

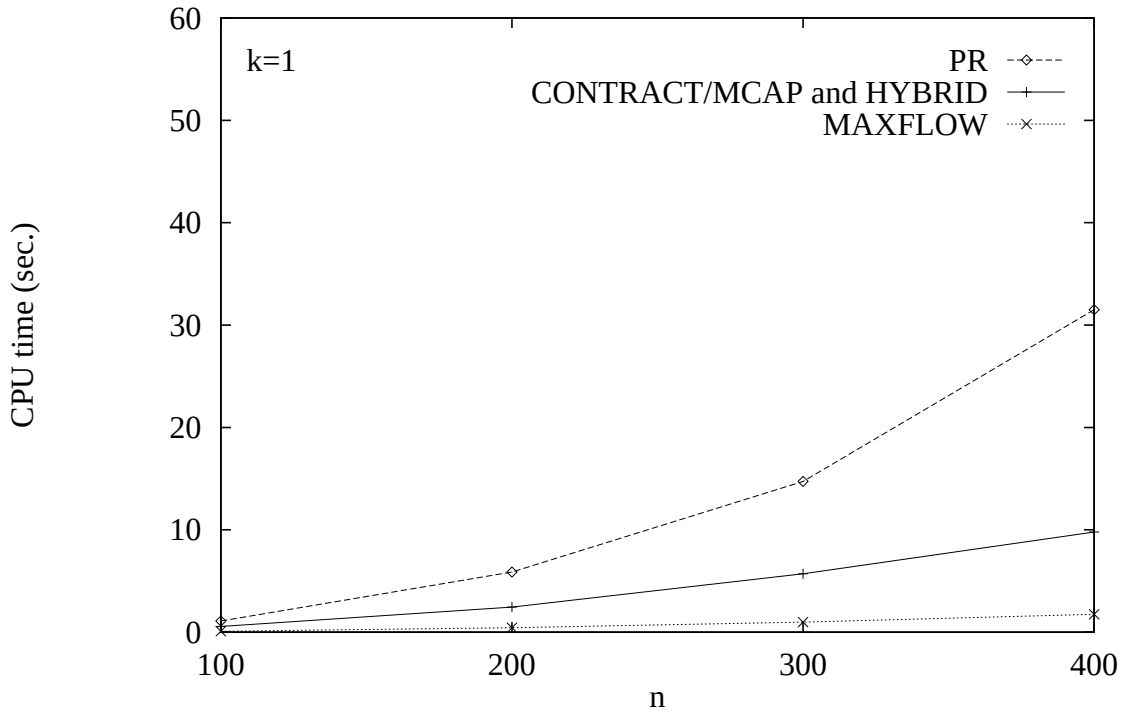

Figure 1: CPU time of algorithms PR, CONTRACT/MCAP, HYBRID and MAXFLOW when the number of vertices n changes ($d = 50(\%)$, $k = 1$, $p = 1/n$).

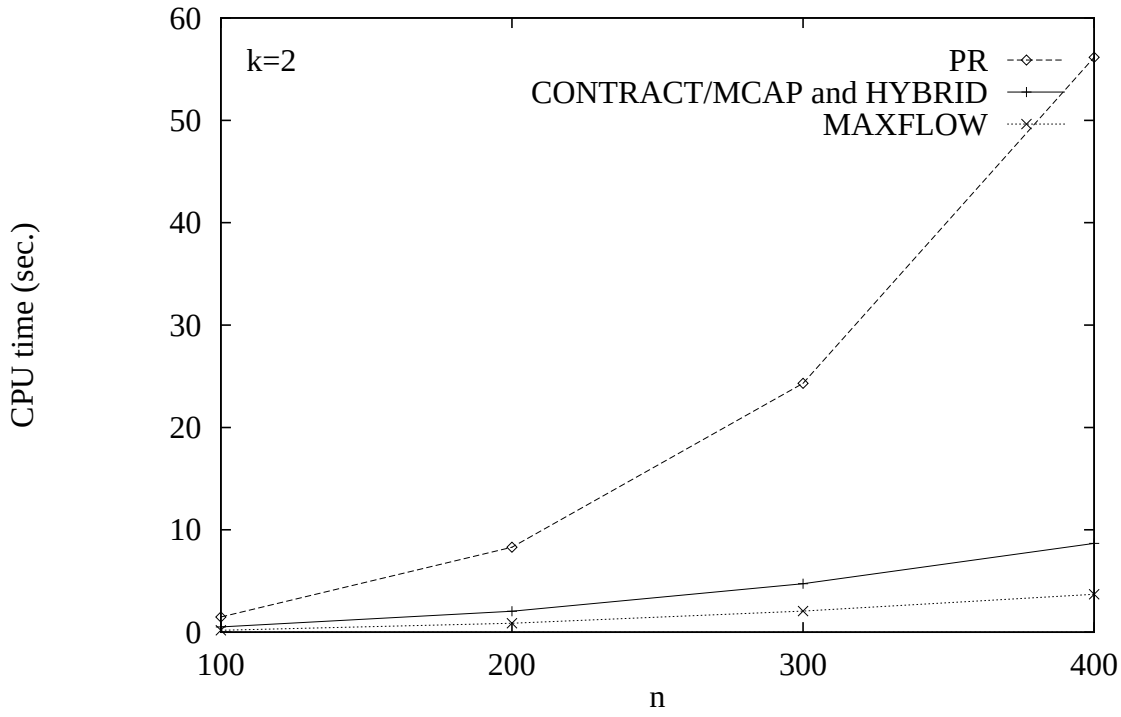

Figure 2: CPU time of algorithms PR, CONTRACT/MCAP, HYBRID and MAXFLOW when the number of vertices n changes ($d = 50(\%)$, $k = 2$, $p = 1/n$).

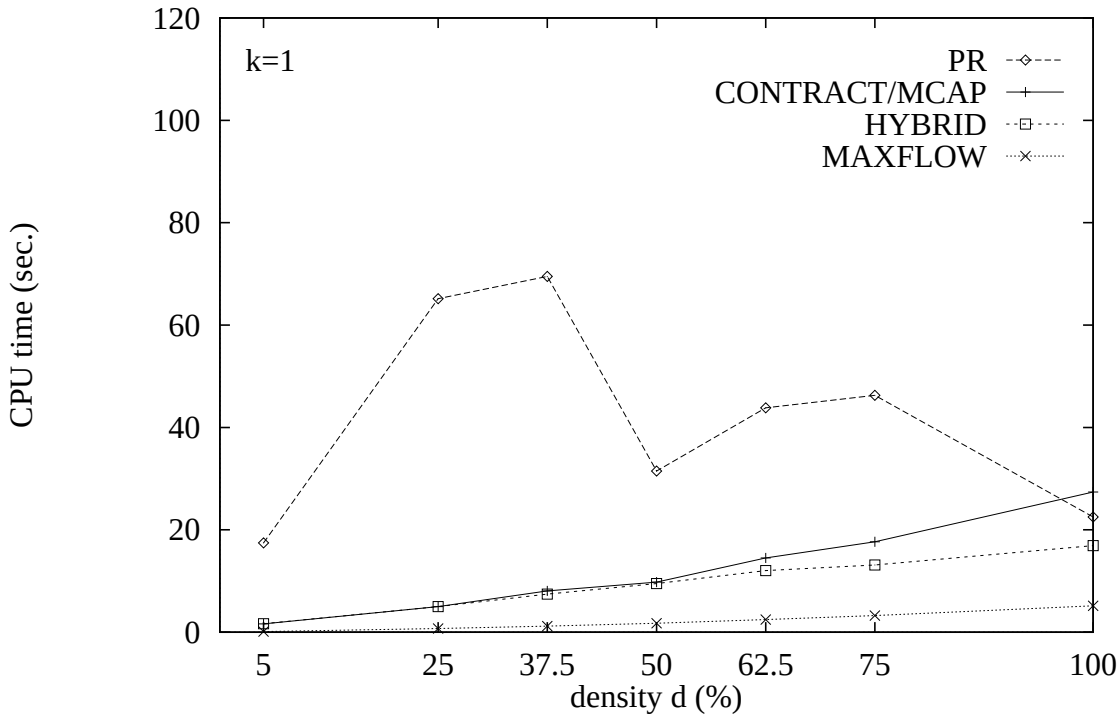

Figure 3: CPU time when the edge density d changes ($n = 400,\ k = 1,\ p = 1/n$).

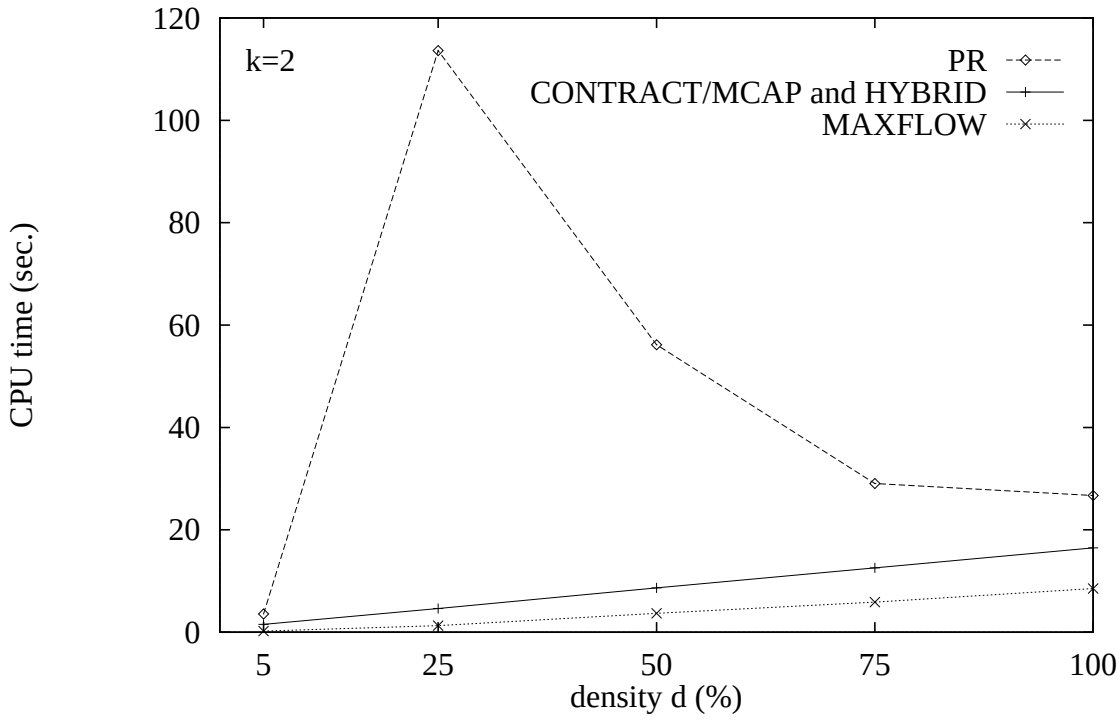

Figure 4: CPU time when the edge density d changes ($n = 400,\ k = 2,\ p = 1/n$).

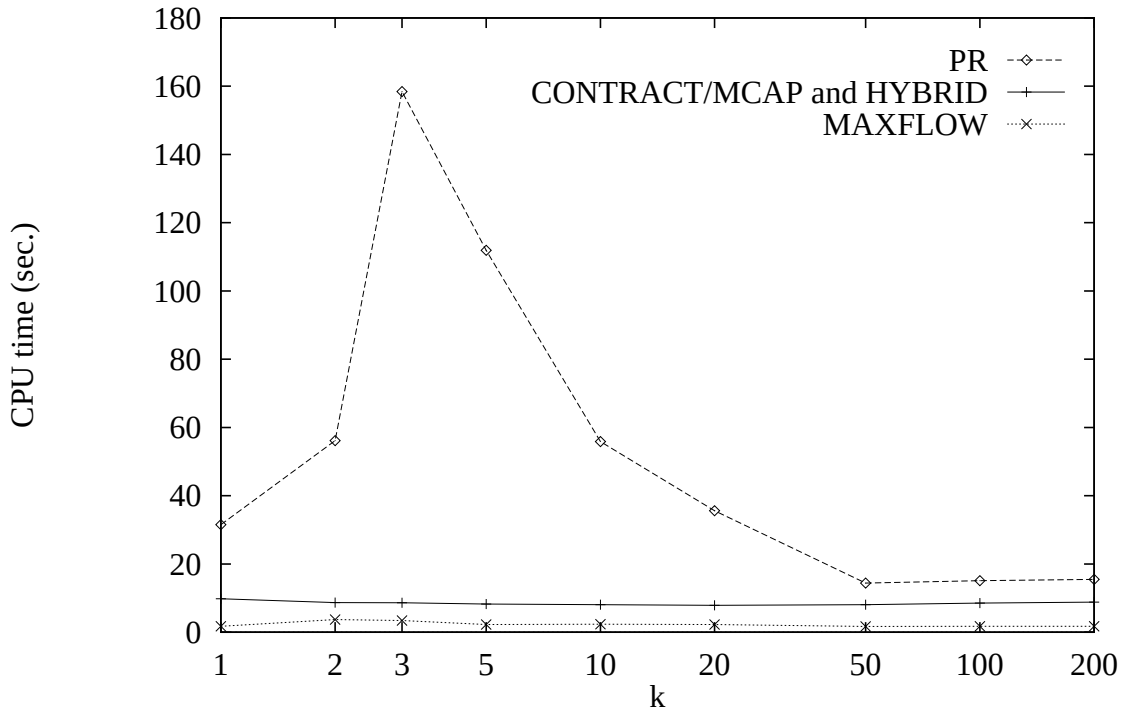

Figure 5: CPU time when decomposition number k changes ($n = 400,\ d = 50(\%),\ p = 1/n$).

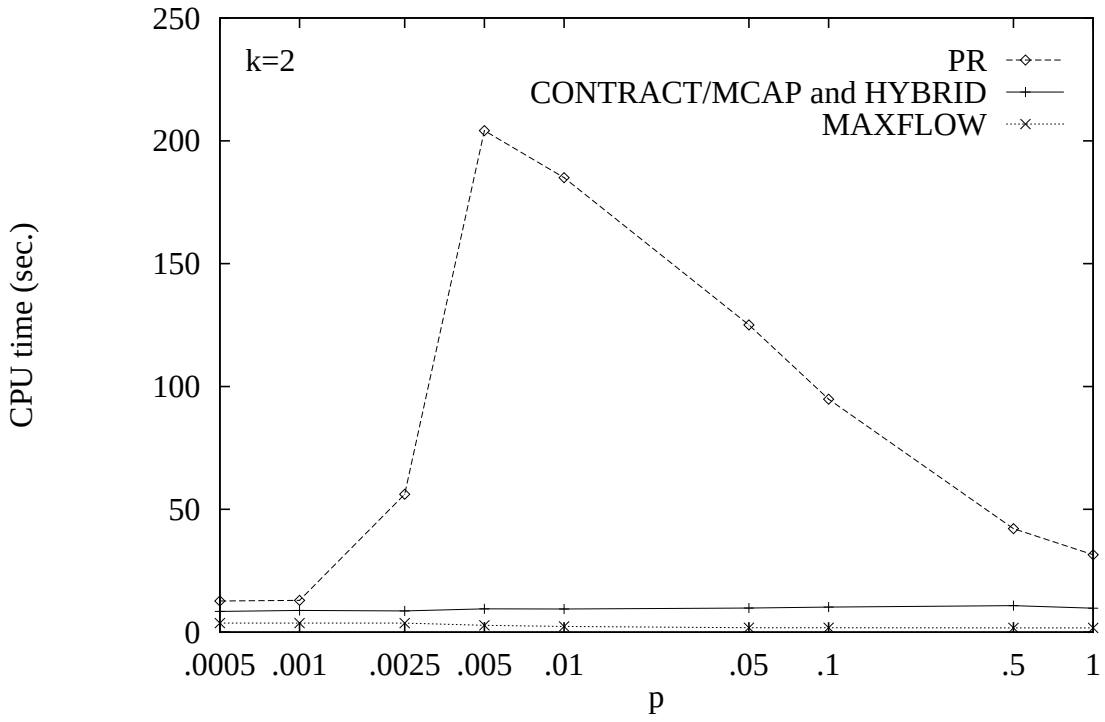

Figure 6: CPU time when constant p changes ($n = 400,\ d = 50(\%),\ k = 2$).