数理解析研究所講究録
第 833 巻 1993 年 120-130

A Faster Algorithm for the Minimum Capacity Cut Problem of Undirected Networks

Tadashi ONO (小野 正)   Hiroshi NAGAMOCHI (永持 仁)   Toshihide IBARAKI (茨木 俊秀)

Department of Applied Mathematics and Physics,
Faculty of Engineering,
Kyoto University

1 Introduction

Let $\mathcal{N} = (G = (V, E), c)$ be a simple undirected network with a set V of vertices and a set E of edges, where $c : E \rightarrow \mathcal{R}^+$ (the set of nonnegative reals) gives the capacities of edges. The minimum capacity cut problem is to find a nonempty subset $U \subset V$, $U \neq V$ that minimizes the total capacity of the edges between U and $V - U$.

Gomory and Hu [3] show that the optimal solution of this problem can be obtained by solving $|V| - 1$ max-flow problems. The currently best bound for solving the max-flow problem is $O(mn \log(n^2/m))$ due to Goldberg and Tarjan [2], where $n = |V|$ and $m = |E|$. Thus the time complexity of Gomory and Hu's method is $O(mn^2 \log(n^2/m))$.

Padberg and Rinaldi [8] present an improved approach such that two vertices are shrunk into a single vertex to obtain a smaller network after examining the minimum capacity cut separating these vertices. Although their method requires $n - 1$ max-flow computations in the worst case as Gomory and Hu's does, they also provide a simple test to shrink a pair of vertices, by which the number of max-flow computations which need to be conducted is considerably reduced. Their algorithm is considered to be one of the fastest one in the practical sense.

In this paper, we present an efficient implementation of a faster $O(mn+n^2 \log n)$ algorithm proposed by Nagamochi and Ibaraki [6], and add various mechanisms so that as many edges as possible can be shrunk in each iteration. From numerical experiments, we find out that our implementation is much faster and more stable than Padberg and Rinaldi's for a wide variety of networks.

2 Preliminaries

Let $\mathcal{N} = (G = (V, E), c)$ stand for a simple undirected network with a vertex set V, an edge set E, and nonnegative real capacities $c(e)$, $e \in E$. We denote $e = (u, v)$ if the end vertices of $e \in E$ are u and v. A nonempty subset $U \subset V$, $U \neq V$ is called a *cut*, and the capacity for a cut U is given by $c(U) = \sum_{u \in U, v \in V-U} c(u, v)$. For notational convenience, a singleton set $\{u\}$ may also be written as u. The *minimum capacity cut problem* is the problem of finding a cut U that minimizes $c(U)$ in $\mathcal{N}$. We simply 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 capacity of a cut with the minimum capacity among the cuts that separate x and y is denoted $\lambda(x, y)$.

Shrinking edge $e = (u, v)$ of $\mathcal{N}$ is defined as follows: merge u and v into a single vertex by deleting edge $e = (u, v)$, and combine the resulting multiple edges with the same pair of end vertices into a single edge having the sum of the capacities of such edges. We denote the network obtained by shrinking e as $\mathcal{N}' = (G/e, c')$ (we may write $G/\{u, v\}$ instead of G/e), where c' is the capacities of the resulting network.

3 Shrinking Edges by CAPFOREST

3.1 Computing the Minimum Cut Based on Shrinking

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

Lemma 1 For a network $\mathcal{N} = (G, c)$ and a shrunk network $\mathcal{N}' = (G/e, c')$ where $e = (x, y) \in E$, we have

$$\lambda(\mathcal{N}) = \min\{\lambda(\mathcal{N}'), \lambda(x, y)\}. \qquad \square \tag{1}$$

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

$$\lambda(x,y) \geq \overline{\lambda}, \tag{2}$$

then there are no cuts separating x and y whose capacities are less than $\overline{\lambda}$.

Proof. It is immediate from the definition of $\lambda(x,y)$. □

If condition (2) holds for x and y, we have $\lambda(\mathcal{N}) = \min\{\lambda(\mathcal{N}'), \overline{\lambda}\}$ for network $\mathcal{N}'$ obtained by shrinking x and y. For an edge $e = (x,y)$ which satisfies the condition (2) for a $\overline{\lambda}$, we call such an edge *shrinkable*.

Based on this observation, we can compute $\lambda(\mathcal{N})$ in the following manner: we first examine all $c(v)$, $v \in V$ in the given network, and store the smallest $c(v)$ as $\overline{\lambda}$. Second, find two vertices x and y satisfying (2) for the current $\overline{\lambda}$, and shrink x and y into a vertex x', setting $\overline{\lambda} = \min\{\overline{\lambda}, c'(x')\}$. When we repeat this shrinking until the number of vertices becomes two, the current $\overline{\lambda}$ is equal to $\lambda(\mathcal{N})$.

This procedure is stated as follows.

```
Procedure SHRINK
Input: a simple undirected network $\mathcal{N} = (G, c)$
Output: $\lambda(\mathcal{N})$
begin
    $\overline{\lambda} := \min\{c(v) \mid v \in V\}$; $\mathcal{N}' := (G' = (V', E'), c') := \mathcal{N}$;
    while $|V'| \geq 3$ do
        begin
            if a cut $X \subset V$ with $c'(X) < \overline{\lambda}$ is found then $\overline{\lambda} := c'(X)$;
            Find two vertices $x, y$ such that $\lambda(x,y) \geq \overline{\lambda}$;
            Shrink $x$ and $y$ into $x'$ and let $\mathcal{N}' = (G' = (V', E'), c')$ be the resulting network;
            $\overline{\lambda} := \min\{c'(x'), \overline{\lambda}\}$;
        end
    Conclude that $\lambda(\mathcal{N}) = \overline{\lambda}$;
end.
```

We can get the minimum cut capacity after performing $n-1$ operations of vertex shrinking. In order to find two vertices x and y such that $\lambda(x,y) \geq \overline{\lambda}$, we may compute $\lambda(x,y)$ by the max-flow algorithm. In this case, after setting $\overline{\lambda} := \min\{\overline{\lambda}, \lambda(x,y)\}$, the above condition is satisfied. Padberg and Rinaldi's algorithm [8] is the first one which is based on procedure SHRINK. Specifically, they try to find an edge $e = (x,y)$ with $\lambda(x,y) \geq \overline{\lambda}$ using a simple test, and rely on a max-flow algorithm only when this heuristic test fails to find such edge $e = (x,y)$ (see [8], for detail).

3.2 Algorithm CAPFOREST

Algorithm CAPFOREST presented by Nagamochi and Ibaraki [6] computes the following lower bound $q(e)$ on $\lambda(x,y)$ for all edges $e=(x,y)$ in $O(m+n\log n)$ time.

Lemma 3 [6] The $q(e)$ obtained by CAPFOREST satisfies

$$\lambda(x,y) \geq q(e), \qquad e=(x,y)\in E, \tag{3}$$

and at least one edge $e^*=(x^*,y^*)\in E$ satisfies

$$q(e^*)=\lambda(x^*,y^*). \qquad \square \tag{4}$$

To compute $\lambda(\mathcal{N})$ based on procedure SHRINK, we exploit these $q(e)$ obtained by CAPFOREST for finding shrinkable edges.

Lemma 4 Let $q(e)$ be the lower bound obtained by CAPFOREST, and $\overline{\lambda}$ be the capacity of a cut which has already been detected. If

$$q(e)\geq \overline{\lambda}, \tag{5}$$

then e is shrinkable.

Proof. It is obvious from Lemma 2 and condition (3). $\square$

By employing CAPFOREST to find shrinkable edges in algorithm SHRINK, we have the following algorithm to compute $\lambda(\mathcal{N})$, which is a modification of the original algorithm by Nagamochi and Ibaraki [6].

```
Algorithm CF_SHRINK
Input: a simple undirected network $\mathcal{N}=(G=(V,E),c)$
Output: $\lambda(\mathcal{N})$
begin
  $\overline{\lambda}:=\min\{c(v)\mid v\in V\}$; $\mathcal{N}':=(G'=(V',E'),c'):=\mathcal{N}$;
  while $|V'|\geq 3$ do
    begin
      Execute CAPFOREST $(G',c';q)$;
      $\overline{\lambda}:=\min\{\overline{\lambda},\lambda(x^*,y^*)\}$ for the edge $(x^*,y^*)$ of (4);
      while there exist edges $e=(x,y)$ with $q(e)\geq\overline{\lambda}$ do
```

```
            begin
                Shrink $x, y$ into $x'$ and let $\mathcal{N}' = (G' = (V', E'), c')$ be the resulting network;
                $\overline{\lambda} := \min\{c(x'), \overline{\lambda}\}$;
            end
        end
    Conclude that $\lambda(\mathcal{N}) = \overline{\lambda}$;
end.
```

Once we execute CAPFOREST in CF_SHRINK, we can find at least one edge $e^* = (x^*, y^*)$ that satisfies (3) by Lemma 3, which implies that (x^*, y^*) is shrinkable after setting $\overline{\lambda} := \min\{\overline{\lambda}, \lambda(x^*, y^*)\}$. Therefore, we can continue vertex shrinking until the number of vertices becomes two. Hence, $\lambda(\mathcal{N})$ can be computed by executing CAPFOREST at most $n-2$ times, and thus the running time of CF_SHRINK is $O(mn + n^2 \log n)$.

3.3 Algorithm MODIFIED_CAPFOREST

In this section, We revise CAPFOREST in order to find shrinkable edges and small cuts more efficiently. The revised CAPFOREST, called MODIFIED_CAPFOREST, computes a set T of shrinkable edges and detects a certain cut capacity α in addition to the original computation of CAPFOREST.

```
Algorithm MODIFIED_CAPFOREST $(G, c, \overline{\lambda}; q, T, \alpha)$
Input: a simple undirected network $\mathcal{N} = (G = (V, E), c)$, and the currently obtained
minimum cut capacity $\overline{\lambda} > 0$;
Output: the lower bound $q(e)$, a shrinkable edge set $T$, and a certain cut capacity $\alpha$;
begin
    Label all vertices $v \in V$ and all edges $e \in E$ "unscanned";
    $r(v) := 0$,  $\alpha(v) := 0$, for all $v \in V$;
    $q(e) := 0$, for all $e \in E$;
    $T := \emptyset$; $\alpha' := 0$;
    while there exist "unscanned" vertices do
        begin
            Choose an "unscanned" vertex $x$ with the largest $r$;
            $\alpha' := \alpha' + c(x) - 2r(x)$; $\alpha(x) := \alpha'$;
            for each vertex $y$ adjacent to $x$ by an "unscanned" edge do
            begin
                if $r(y) < \overline{\lambda} \leq r(y) + c(e)$ then $T := T \cup \{e\}$;
                $q(e) := r(y) + c(e)$; $r(y) := r(y) + c(e)$;
                Mark $e$ "scanned";
            end
        Mark $x$ "scanned";
    end
    $\alpha := \min\{\alpha(v) \mid v \in V\}$;
end.
```

The output $q(e)$ is the same as by CAPFOREST [6], so it also satisfies conditions (3) and (4).

Theorem 5 Let $G_{q,\overline{\lambda}}$ define a subgraph of G such as $G_{q,\overline{\lambda}} = (V, \{e \in E \mid q(e) \geq \overline{\lambda}\})$. An edge set T obtained by MODIFIED_CAPFOREST is a maximal spanning forest of $G_{q,\overline{\lambda}}$.

Proof. See [7]. □

The network resulting from shrinking all edges in a maximal spanning forest T of $G_{q,\overline{\lambda}}$ is the same as the one obtained by shrinking all edges in $G_{q,\overline{\lambda}}$. The mechanism of identifying such T in MODIFIED_CAPFOREST saves the unnegligible time, if compared with the time to collect all edges in $G_{q,\overline{\lambda}}$ of in T from scratch.

Next we mention about the capacity α of a certain cut.

Lemma 6 Let x_i $(1 \leq i \leq n-1)$ be the i-th vertex visited by MODIFIED_CAPFOREST and define $X_i = \{x_1, x_2, \ldots, x_i\}$. Then $\alpha(x_i) = c(X_i)$. □

By Lemma 6, the minimum capacity $\alpha = \min_i \alpha(x_i)$ among the cuts X_i may be relatively small, because CAPFOREST tends to select a subset of some vertices connected each other by those edges with relatively large capacities in the early stage of computation. If this α satisfies $\alpha < \overline{\lambda}$, we update $\overline{\lambda} := \alpha$. So $\overline{\lambda}$ may decrease in an earlier stage, thereby enlarging the set of edges satisfying condition (5).

By replacing CAPFOREST with MODIFIED_CAPFOREST in CF_SHRINK, we have the following more efficient algorithm.

Algorithm MCF
Input: a simple undirected network $\mathcal{N} = (G = (V, E), c)$

```
Output: $\lambda(\mathcal{N})$
begin
  $\overline{\lambda} := \min\{c(v) \mid v \in V\}$; $\mathcal{N}' := (G' = (V', E'), c') := \mathcal{N}$;
  while $|V'| \geq 3$ do
    begin
      Execute MODIFIED_CAPFOREST $(G', c', \overline{\lambda}; q, T, \alpha)$;
      $\overline{\lambda} := \min\{\alpha, \overline{\lambda}\}$;
      for $e = (x, y) \in T$ do
        begin
          Shrink $x, y$ into $x'$ and let $\mathcal{N}' = (G' = (V', E'), c')$ be the resulting network;
          $\overline{\lambda} := \min\{c(x'), \overline{\lambda}\}$;
        end
    end
  Conclude that $\lambda(\mathcal{N}) = \overline{\lambda}$;
end.
```

4 Computational Experiments

In this section, we report the results from the computational experiments conducted to evaluate the performance of the proposed algorithm MCF. All computer programs were coded in FORTRAN 77 in double precision and run on a workstation SUN SPARC 1 IPX.

4.1 Generation of Networks for Experiments

In our numerical experiments, we specify network types by the following four parameters: the number of vertices n, the density of edges $d = \dfrac{2m}{n(n-1)} \times 100$ (%) (m is the number of edges), 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 $\dfrac{n(n-1)(d/100)}{2}$ edges by randomly generating an edge between a pair of vertices, after creating a Hamilton path to ensure the connectivity. Second, we define the capacity of each edge by the following rule. When $k = 1$, capacity of each edge is randomly chosen from the interval $[0, 100)$ uniformly. When $k \geq 2$, we divide vertices into k subsets randomly (each subset is called a cluster), and the capacities of the edges connecting two vertices in the same cluster are selected from $[0, 100)$, while the capacities of the rest of edges are randomly chosen from $[0, 100p)$. In our experiments, p is set to $1/n$ except in Figure 6 so that a set of clusters is likely to form a minimum cut in the generated network. Note that $p = 1/n$ is used because the ratio of the capacity $c(x)$ of a singleton $\{x\}$ to that of non-singleton set is at most $\dfrac{n-1}{n(n-1)/2} = \dfrac{2}{n}$.

4.2 Computational Results

By solving the networks generated in the above manner, we compare the average running times of Padberg and Rinaldi's algorithm (abbreviated to PR), the proposed algorithm (abbreviated to MCF), and a single max-flow computation by Goldberg and Tarjan [2] (abbreviated to MAXFLOW), where ten instances are tested with the same four parameter values.

Figures 1 and 2 show the CPU time of the above three algorithms for various values of n for density $d = 50$ (%), for $k = 1$ and $k = 2$ respectively. From these figures, we can see that MCF is much faster than PR, and the larger the number of vertices is, the greater the difference of the computational time becomes. When $k = 2$, PR takes longer time compared with case of $k = 1$, because the network loses uniformity of capacity distribution and it becomes hard to find shrinkable edges only by heuristic test. Contrary to this, MCF could accelerate the shrinking process, with the help of the value α of a small capacity cut. As a result, the difference between MCF and PR becomes greater when k is changed from 1 to 2.

Figures 3 and 4 illustrate the CPU time of the above algorithms for different densities. In Figure 3, running time of PR fluctuates depending on the density. In PR, the probability of finding shrinkable edges tends to become small when the density is low, and hence the number of times to rely on max-flows computations increases. Conversely, when density is high, the probability that the heuristic test of PR for a single edge succeeds becomes very close to 1. The time required to find one shrinkable edge by PR is shorter than that by MCF. For this reason, PR is slightly faster than MCF if $d = 100$ (%) and $k = 1$ (Figure 3). However, for $k = 2$ (Figure 4), computing α helps MCF run faster even for highly dense networks.

Figure 5 shows the results when the decomposition number k changes. As we discussed above for case of $k = 2$ previously, an edge whose both end vertices are in the same cluster is not likely to be shrinkable by the heuristic test in PR. It still has the same difficulty for $3 \le k \le 20$. When $k \ge 50$, the capacities distribute nearly uniformly in the sense that a handful of edges with relatively large capacities are scattered over the entire network, and such edges can be shrunk in an early stage of the algorithm, resulting in the high performance of PR. On the contrary, MCF 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 $[0.0005, 1]$. Note that when $p = 1$, there is one cluster. When $p = 0.005 = 2/n$, since the expectation of the capacity of a singleton is approximately equal to that of one cluster produced by $k = 2$, both a singleton and non-singleton cuts may be a minimum cut. Generally by the heuristic test in PR, it is difficult to find a minimum cut if the capacities of both types of cuts are nearly equal. Therefore PR takes large time for $0.005 \leq p \leq 0.1$.

From the above observation, we conclude that proposed MCF is much faster than PR. MCF may be considered as one of the fastest practical program available for the minimum capacity cut problem. Its running time is almost comparable (i.e., always at most $O(mn + n^2 \log n)$ times) with the fastest max-flow algorithm. Another advantage of MCF is its simplicity of implementation, because MCF basically consists only of MODIFIED_CAPFOREST, while PR requires the max-flow algorithm and a sophisticated control for computing heuristic tests. Only in the case of $k = 1$ and $d \geq 95$ (%), PR runs slightly faster than MCF in above experiment.

4.3 A Hybrid Implementation: Algorithm HCF

In order to improve our algorithm MCF for the case of $k = 1$ and $d = 100$ (%), we also propose algorithm HCF, a hybrid version of MCF and PR obtained by including the heuristic test of PR into MCF. The reason why PR runs faster than MCF in case of $k = 1$ and $d \geq 95$ (%) is that the heuristic test for shrinking succeeds with probability nearly close to 1. Only in this situation, the time required by PR for finding one shrinkable edge is shorter than MCF. We want to capture this power of the heuristic test, but have to take care not to increase the running time of MCF for other cases. Concretely speaking, after MODIFIED_CAPFOREST followed by shrinking T, we execute the heuristic test only for those edges having relatively large capacities among the edges incident to the vertex that was shrunk most recently. We continue this test only when the previous one successfully finds a shrinkable edge, and once it fails, we go back to start MODIFIED_CAPFOREST (see [7], for detail).

As aimed, the above heuristics works effectively for the case of $k = 1$ and $d \geq 95$ (%) in HCF. As a result, HCF runs faster than any of PR and MCF, as shown in Figure 3. It is also observed that HCF performs almost the same as MCF in other cases. The HCF algorithm is slightly complicated than MCF to implement.

5 Conclusion

By revising some part of the algorithm proposed by Nagamochi and Ibaraki [6], we obtained a practically efficient implementation for computing the minimum capacity cut of undirected networks. From computational results, we have shown that MCF is faster than PR proposed by Padberg and Rinaldi in most cases except the case in which capacities are uniformly distributed on network with very high density. Moreover, an important advantage of MCF is in its simplicity of implementation. We also proposed a hybrid implementation obtained by including the heuristic test of PR into MCF, and showed that it is never slower than any of the PR and MCF for all types of networks we examined in this study.

References

[1] L. Ford, and D. Fulkerson: "Maximal Flow Through a Network," *Canadian Journal of Mathematics*, Vol. 8, pp.399-404, 1956.

[2] A. Goldberg and T. Tarjan: "A new approach to the maximum flow problems," *Proceedings of 18th ACM Symposium on the Theory of Computing*, pp.136-146, 1986.

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

[4] T. Ibaraki: *Algorithms and Data Structures* (in Japanese), Shokodo, 1989.

[5] T. Ibaraki and M. Fukushima: *FORTRAN 77 Optimization Programming* (in Japanese), Iwanami-Shoten, 1991.

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

[7] T. Ono: "A Faster Algorithm for the Minimum Capacity Cut Problem of Undirected Networks," *Master thesis*, Department of Applied Mathematics and Physics, Faculty of Engineering Kyoto University, 1993.

[8] M. Padberg and G. Rinaldi: "An efficient algorithm for the minimum capacity cut problem," *Mathematical Programming*, Vol. 47, pp.19–36, 1990.

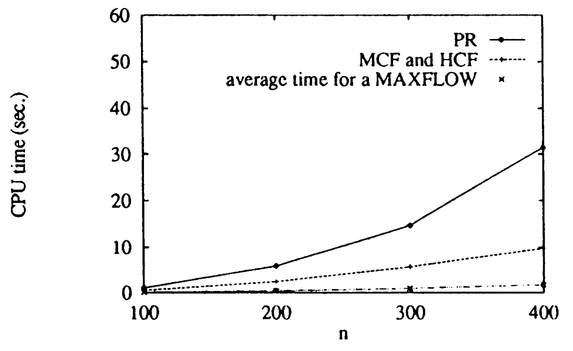

Figure 1: CPU time of algorithm PR, MCF and MAXFLOW when the number of vertices n changes ($d = 50(\%)$, $k = 1$, $p = 1/n$).

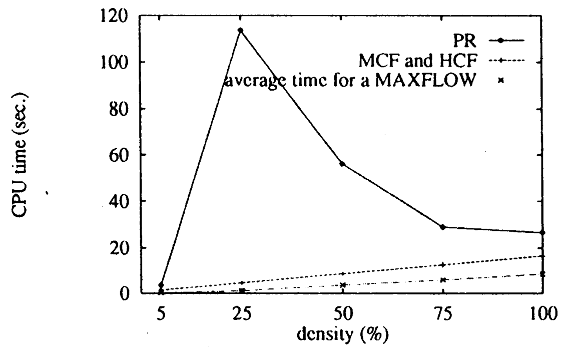

Figure 4: CPU time of algorithm PR, MCF and MAXFLOW when the edge density d changes ($n = 400$, $k = 2$, $p = 1/n$).

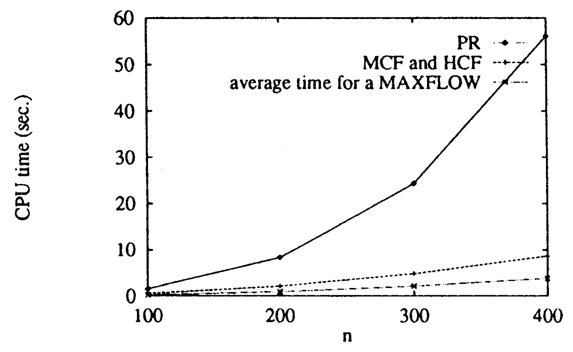

Figure 2: CPU time of algorithm PR, MCF and MAXFLOW when the number of vertices n changes ($d = 50(\%)$, $k = 2$, $p = 1/n$).

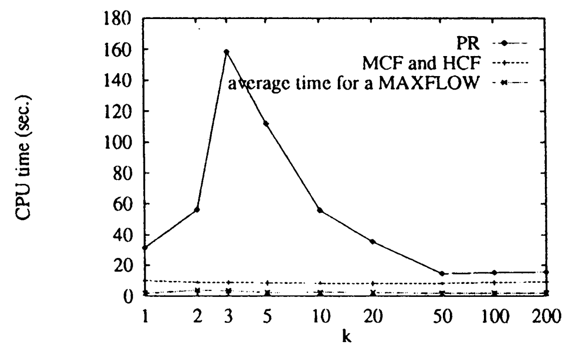

Figure 5: CPU time of algorithm PR, MCF and MAXFLOW for various decomposition number k ($n = 400$, $d = 50(\%)$, $p = 1/n$).

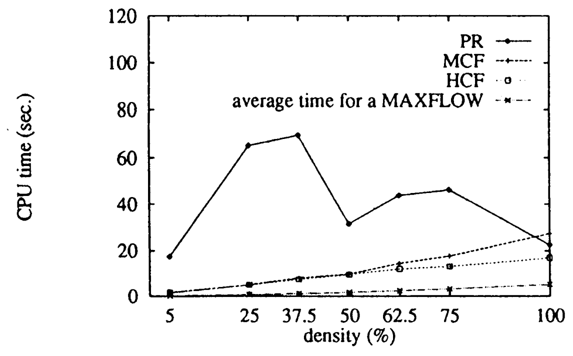

Figure 3: CPU time of algorithm PR, MCF, HCF and MAXFLOW when the edge density d changes ($n = 400$, $k = 1$, $p = 1/n$).

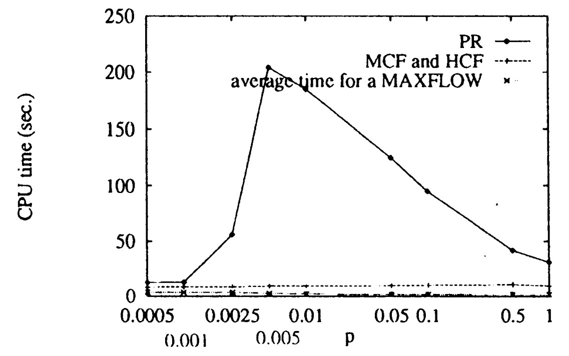

Figure 6: CPU time of algorithm PR, MCF and MAXFLOW for various constant p ($n = 400$, $d = 50(\%)$, $k = 2$).