アルゴリズム 33－1
(1993. 5. 28)

最小容量カットアルゴリズムのプログラムによる効率の良い実現法

永持 仁　小野 正　茨木 俊秀
京都大学 工学部 数理工学教室

本報告では, Nagamochi と Ibaraki (SIAM J. on Discrete Mathematics (1992) pp.54–66) によって提案された無向ネットワークの最小容量カットを計算する $O(mn+n^2\log n)$ 時間アルゴリズムのプログラムによる効率の良い実現法を示す. プログラムには各反復においてできるだけ多くの辺が縮約されるようにさらに工夫が加えられている. 得られたプログラムの実現法の性能を評価するために計算機実験を行い, この問題に対する最速のプログラムの 1 つとして知られている Padberg と Rinaldi のアルゴリズム (Math. Prog. (1990) pp.19–36) と計算時間を比較した. この結果, 提案するプログラムは Padberg と Rinaldi のものに比べかなり速く, 計算時間は解かれるネットワークの形態にはほとんど左右されないことが分かった.

An Efficient Implementation for Minimum Capacity Cut Algorithm

Hiroshi NAGAMOCHI　Tadashi ONO　Toshihide IBARAKI
Dept. of Applied Mathematics and Physics
Kyoto University
Sakyo-ku, Kyoto 606-01

In this paper, we present an efficient implementation of the $O(mn+n^2\log n)$ time algorithm originally proposed by Nagamochi and Ibaraki (SIAM J. on Disc. Math. (1992) pp.54–66) for computing the minimum capacity cut of an undirected network. To enhance computation, various mechanisms are added so that it can contract as many edges as possible at each iteration. To evaluate the performance of the resulting implementation, we conducted extensive computational experiments, and compared the results with that of Padberg and Rinaldi's algorithm (Math. Prog. (1990) pp.19–36), 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 affected much by the types of the networks being solved.

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 capacity function $c : E \rightarrow \mathcal{R}^+$ (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 or communication networks, VLSI designs, facet identification for the traveling salesman problem, and so on.

Gomory and Hu [3] show that a minimum cut of $\mathcal{N}$ can be obtained by solving $|V| - 1$ maximum flow problems. One of the currently best time bound for solving the maximum flow problem is $O(mn \log(n^2/m))$ due to Goldberg and Tarjan [2], where $n = |V|$ and $m = |E|$, and hence the time complexity of Gomory and Hu's method is $O(mn^2 \log(n^2/m))$.

Padberg and Rinaldi [5] present an improved algorithm for this problem. In their method, two vertices are contracted into a single vertex to obtain a smaller network after examining the minimum capacity cut separating these vertices. Although their method could require $n - 1$ maximum flow computations in the worst case as Gomory and Hu's does (and hence its worst time complexity is still $O(mn^2 \log(n^2/m))$), they also provide a simple test to contract a pair of vertices, by which the number of maximum flow computations may be 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 an $O(mn + n^2 \log n)$ algorithm for the minimum capacity cut problem recently proposed by Nagamochi and Ibaraki [4]. This algorithm does not rely on the maximum flow computation, but uses some other 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 extensive numerical experiments to evaluate the performance of the proposed program, and compared it with Padberg and Rinaldi's program. The results indicate that our program is much faster and more stable than Padberg and Rinaldi's program 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 and proper subset U of V is called a *cut*, and the capacity for a cut U is given 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. 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 capacity of a cut with the minimum capacity among the cuts that separate 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 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 contracting 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 to be contracted.

3 Contracting Edges by CAPFOREST

3.1 Computing the Minimum Cut Based on Edge Contraction

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 $e \in E$, the contracted network $\mathcal{N}' = (G/e, c')$ satisfies

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

□

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 $\overline{\lambda}$. □

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 $\lambda(\mathcal{N})$ can be 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 $c(v)$, $v \in V$ 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 $\overline{\lambda}$, and contract (x, y) into a vertex x', setting $\overline{\lambda} := \min\{\overline{\lambda}, c'(x')\}$. When we repeat this contraction operation until the number of vertices becomes two, the final $\overline{\lambda}$ provides us the capacity $\lambda(\mathcal{N})$ of a minimum cut in the original network $\mathcal{N}$. This procedure is stated as follows.

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

The test in line 5 is introduced since a smaller $\overline{\lambda}$ strengthens the test of line 6, and hence reduces the required number of iterations. Although the details of lines 5 and 6 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 if at least one edge is contracted at each iteration. The existence of a contractible edge in line 6 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 the minimum cut separating x and y is smaller than $\overline{\lambda}$, then execute line 5 by considering the minimum cut as X. Otherwise skip line 5. In any case, $\lambda(x, y) \geq \overline{\lambda}$ always holds in line 6, 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 [5] tries to find an edge $e = (x, y)$ with $\lambda(x, y) \geq \overline{\lambda}$ using a simple heuristic test, and uses a maximum flow algorithm only when this heuristic test fails. The algorithm by Nagamochi and Ibaraki never resort 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.

Before proceeding to the next subsection, we note that, by maintaining the cut that attains the current $\overline{\lambda}$, CONTRACT can be modified so that it finds not only the value $\lambda(\mathcal{N})$ of a minimum cut but also a minimum cut in $\mathcal{N}$. For notational simplicity, however, algorithms in this paper will be described only for computing $\lambda(\mathcal{N})$.

3.2 Algorithm CAPFOREST

Nagamochi and Ibaraki [4] presented an algorithm CAPFORST that computes a lower bound $q(e)$ on $\lambda(x,y)$ for every edge $e=(x,y)\in E$. It is a graph traversing procedure that, starting with an arbitrarily given vertex, visits the vertices in V in the order that the next vertex v to visit maximizes the sum of capacities of the edges between v and the vertices already visited. Each edge $e=(x,y)$ is scanned and given a real $q(e)$ when one of its end nodes, say x is visited. The $q(e)$ is set to be the sum of capacities of the edges between y and the vertices already visited. CAPFOREST can be implemented in $O(m+n\log n)$ time [4], and the next property for $q(e)$ is essential for our purposes.

Lemma 3 [4] (i) The $q(e)$ obtained by CAPFOREST satisfies

$$\lambda(x,y)\geq q(e),\qquad e=(x,y)\in E. \tag{5}$$

(ii) The edge e^* satisfying 5 can be found as follows [4]. Let y^* be the vertex visited last in CAPFOREST and let $e^*=(x^*,y^*)$ be the edge scanned last among those incident to y^*. Then this satisfies

$$q(e^*)=\lambda(x^*,y^*). \tag{6}$$

Furthermore, the cut $X=\{y^*\}$ satisfies $c(X)=\lambda(x^*,y^*)$. □

Therefore, the X in (ii) may be used in line 5 of CONTRACT (i.e., $\overline{\lambda}$ is set to $\overline{\lambda}:=\min\{\overline{\lambda},\lambda(x^*,y^*)\}$) to find a contractible edge (x^*,y^*) in line 6. 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 [4].

```
   Algorithm CONTRACT/CAP
   Input: a simple undirected network $\mathcal{N}=(G=(V,E),c)$
   Output: $\lambda(\mathcal{N})$
 1 begin
 2   $\overline{\lambda}:=\min\{c(v)\mid v\in V\}$; $\mathcal{N}'(=(G'=(V',E'),c')):=\mathcal{N}$;
 3   while $|V'|\geq 3$ do
 4     begin
 5       Execute CAPFOREST $(G',c';q)$;
 6       $\overline{\lambda}:=\min\{\overline{\lambda},\lambda(x^*,y^*)\}$ for the edge $(x^*,y^*)$ of (6);
 7       while there exist edges $e$ with $q(e)\geq\overline{\lambda}$ do
 8         begin
 9           Contract $e$ into a vertex $x'$ and let $\mathcal{N}'=(G'=(V',E'),c')$ be the resulting
             network; {when two edges $e',e''$ are to be merged into $\overline{e}$ by contracting $e$,
             set $q(\overline{e}):=\max\{q(e'),q(e'')\}$.}
10           $\overline{\lambda}:=\min\{c(x'),\overline{\lambda}\}$;
11         end
12     end
13   Conclude that $\lambda(\mathcal{N})=\overline{\lambda}$;
14 end.
```

At every execution of CAPFOREST in CONTRACT/CAP, the edge $e=e^*$ always satisfies the condition of line 7 and is contracted. Of course there may also be other edges that are contractible. Therefore, CONTRACT/CAP terminates after executing CAPFOREST at most $n-2$ times, and thus its running time is $O(mn+n^2\log n)$.

3.3 Modification of CAPFOREST

We modify CAPFOREST as follows to facilitate the computation of CONTRACT/CAP.

(i) In CONTRACT/CAP, **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

$$G_{\overline{\lambda}}=(V,E_{\overline{\lambda}}), \tag{7}$$

it is easy to see that $G/T = G/E_{\overline{\lambda}}$ holds. By property $|T| \leq |E_{\overline{\lambda}}|$, it is advantageous to contract T instead of $E_{\overline{\lambda}}$. For this purpose, we modify CAPFOREST so that such T is obtained during its execution.

(ii) In the early stage of computation, CAPFOREST tends to visit those vertices that are mutually connected each other by edges with relatively large capacities. Therefore, when it selects a vertex y with 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 relatively small cut. To make use of this property, we modify CAPFOREST and keep track of the minimum one among such cuts X.

The revised CAPFOREST, called MCAP, computes a forest T of contractible edges and outputs the minimum cut capacity α among those cuts X as explained in (ii), in addition to the original computation of CAPFOREST.

```
   Algorithm MCAP $(G, c, \overline{\lambda}; q, T, \alpha)$
   Input: a simple undirected network $\mathcal{N} = (G = (V, E), c)$, and capacity $\overline{\lambda} > 0$
   of the currently known minimum cut;
   Output: lower bound $q(e)$, a forest $T$ of contractible edges, and
   capacity $\alpha$ of the cut found as explained above;
 1 begin
 2    Label all vertices $v \in V$ "unvisited" and all edges $e \in E$ "unscanned";
 3    $r(v) := 0$ for all $v \in V$;
 4    $\alpha(v) := 0$ for all $v \in V$; $T := \emptyset$; $\alpha' := 0$;
 5    while there exist "unvisited" vertices do
 6       begin
 7          Choose an "unvisited" vertex $x$ with the largest $r$;
 8          $\alpha' := \alpha' + c(x) - 2r(x)$;
 9          $\alpha(x) := \alpha'$;
10          for each vertex $y$ adjacent to $x$ by an "unscanned" edge $e = (x, y)$ do
11             begin
12                if $r(y) < \overline{\lambda} \leq r(y) + c(e)$ then $T := T \cup \{e\}$;
13                $q(e) := r(y) + c(e)$;
14                $r(y) := r(y) + c(e)$;
15                Mark $e$ "scanned";
16             end
17          Mark $x$ "visited";
18       end
19    $\alpha := \min\{\alpha(v) \mid v \in V\}$;
20 end.
```

The modifications from CAPFOREST consist in the above 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 vertex x as the cycle for x. Let x_i $(1 \leq i \leq n-1)$ be the i-th vertex visited by MCAP, and let $X^i = \{x_1, \ldots, x_i\}$.

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

Lemma 5 The edge set T obtained by MCAP is a maximal spanning forest of $G_{\overline{\lambda}}$ of (7). □

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    $\overline{\lambda} := \min\{c(v) \mid v \in V\}$; $\mathcal{N}'(= (G' = (V', E'), c')) := \mathcal{N}$;
 3    while $|V'| \geq 3$ do
 4       begin
 5          Execute MCAP $(G', c', \overline{\lambda}; q, T, \alpha)$;
```

```
6           $\overline{\lambda} := \min\{\alpha, \overline{\lambda}\}$;
7           for $e \in T$ do
8              begin
9                 Contract $e$ into a vertex $x'$ to obtain $\mathcal{N}' = (G' = (V', E'), c')$;
10                $\overline{\lambda} := \min\{c(x'), \overline{\lambda}\}$;
11             end
12       end
13    Conclude that $\lambda(\mathcal{N}) = \overline{\lambda}$;
14 end.
```

As we update $\overline{\lambda}$ by $\overline{\lambda} := \min\{\alpha, \overline{\lambda}\}$, $\overline{\lambda}$ may decrease in an earlier stage of CONTRACT/MCAP, thereby enlarging the set of edges e that satisfy $q(e) \geq \overline{\lambda}$.

4 Computational Experiments

In this section, we report some results from the computational experiments conducted 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 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 generating edges between randomly chosen pairs of vertices, where $n-1$ out of $n(n-1)d/200$ edges are used to create a Hamilton path to make the graph connected. As the name of parameter k suggests, the generated networks tend to have k clusters since the capacities are generated in the following manner. If $k = 1$, capacity of each edge is randomly chosen from interval $[0, 100)$. If $k \geq 2$, we divide vertices into k subsets randomly (each of the subsets is called a cluster), and capacities of the edges connecting two vertices in the same cluster are selected from $[0, 100)$, while the capacities of the edges between different clusters 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.

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 (PR, for short),
- the proposed algorithm CONTRACT/MCAP, and
- a single maximum flow computation by the algorithm of Goldberg and Tarjan [2] (MAXFLOW, for short), which is included only for the reference purposes.

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

First, Figures 1 and 2 show the CPU time of the above three algorithms 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 only by the heuristic test of PR. Contrary to this, CONTRACT/MCAP could accelerate the 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. In these figures, the running time of PR fluctuates depending on the density. In PR, the probability of finding contractible edges by heuristic tests tends to become small when the density is low, and hence the number of times to rely on maximum flow computation increases (though one execution time MAXFLOW decreases). However, when density is high (i.e., d is close to 1), the heuristic tests of PR 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 $n = 400$. By the mechanism of defining capacities as described above, an edge whose both end vertices are in the same cluster is not likely to be contractible by the heuristic tests of PR. If $k \geq 50$, 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 in an early stage of the algorithm. This results in the 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 $n = 400$. Note that when $p = 1$, there is only one cluster (i.e., equivalent to the case of $k = 1$). When $p = 0.005$ (i.e., $p = 2/n$) and $k = 2$, since the expected capacity $c(x)$ of a vertex x is approximately equal to the capacity of one cluster, either a singleton cut or non-singleton cut may be a minimum cut. In this situation, the heuristic tests of PR rarely finds contractible edges. Therefore PR takes large time for $0.005 \leq p \leq 0.1$.

From the above observation, we conclude that proposed CONTRACT/MCAP is much faster than PR. CONTRACT/MCAP may be considered as one of the fastest practical program available for the minimum capacity cut problem. Its running time is almost comparable to the fastest 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 heuristic tests. Only in the case of $k = 1$ and $d \geq 95$ (%), PR runs slightly faster than CONTRACT/MCAP in the above experiment.

4.3 A Hybrid Implementation: Algorithm HC

In order to improve our algorithm CONTRACT/MCAP for problem instances with $k = 1$ and $d \geq 95$ (%), for which PR runs faster, we propose algorithm HC (abbreviation of Hybrid CONTRACT/MCAP), a hybrid version of CONTRACT/MCAP and PR. It is obtained by including a part of the heuristic test of PR into CONTRACT/MCAP. The reason why PR runs faster than CONTRACT/MCAP for the above types of problem instances is that the heuristic test for contracting is successful with high probability. We want to capture this power of heuristic test, but 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 a T), we apply the heuristic test of PR only for edges having relatively large capacities among those incident to vertex x' obtained by contracting the last edge in T (or an edge found contractible by the heuristic test). We continue this test only when all the previous tests successfully find contractible edges, and once it fails, we give up applying the heuristic test and find another T of contractible edges by restarting MCAP. This modification turns out to be effective for problem instances such as $k = 1$ and $d \geq 95$ (%). As a result, HC runs faster than any of PR and CONTRACT/MCAP, as shown in Figure 3. It is also observed that HC performs almost the same as CONTRACT/MCAP for other types of problem instances.

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 problem," *J. ACM*, 35 (1988), pp.921-940.

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

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

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

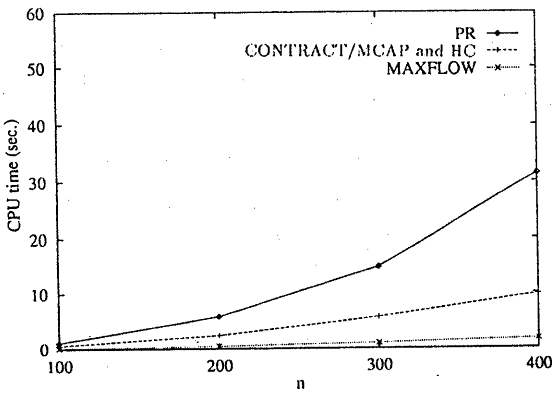

Figure 1: CPU time of algorithms PR, CONTRACT/MCAP, CH and MAXFLOW when the number of vertices n changes (d = 50(%), k = 1, $p = 1/n$).

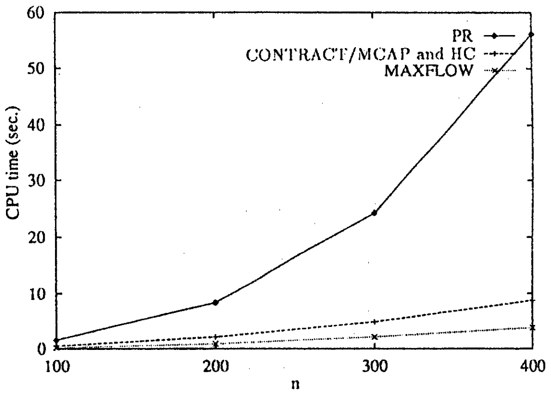

Figure 2: CPU time of algorithms PR, CONTRACT/MCAP, CH and MAXFLOW when the number of vertices n changes (d = 50(%), k = 2, $p = 1/n$).

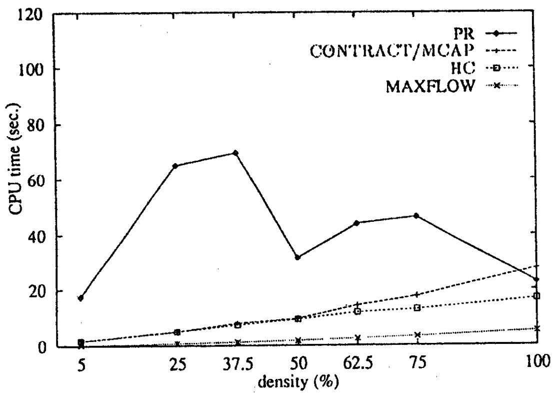

Figure 3: CPU time of algorithms PR, CONTRACT/MCAP, HC and MAXFLOW when the edge density d changes (n = 400, k = 1, $p = 1/n$).

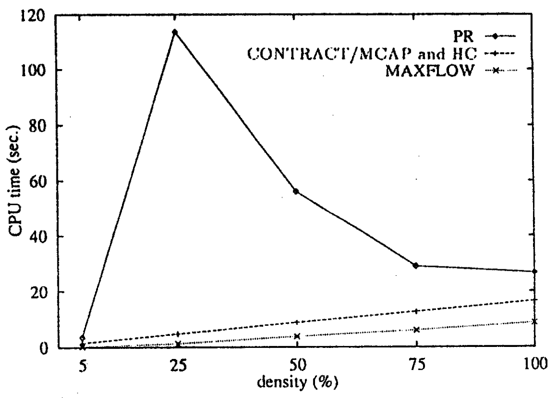

Figure 4: CPU time of algorithms PR, CONTRACT/MCAP, CH and MAXFLOW when the edge density d changes (n = 400, k = 2, $p = 1/n$).

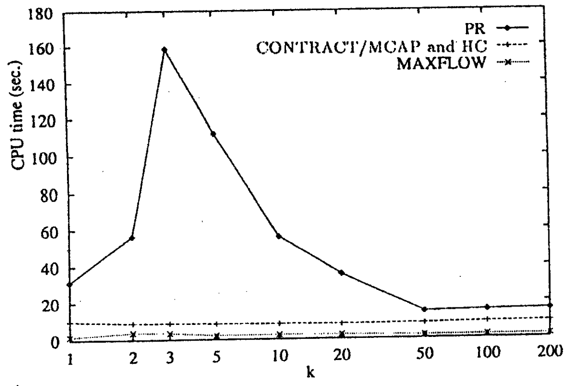

Figure 5: CPU time of algorithms PR, CONTRACT/MCAP, CH and MAXFLOW when decomposition number k changes (n = 400, d = 50(%), $p = 1/n$).

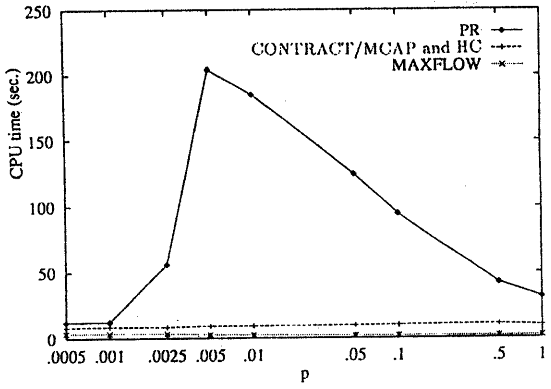

Figure 6: CPU time of algorithms PR, CONTRACT/MCAP, CH and MAXFLOW when constant p changes (n = 400, d = 50(%), k = 2).