ELSEVIER

Discrete Applied Mathematics 65 (1996) 167–190

DISCRETE APPLIED MATHEMATICS

A new unifying heuristic algorithm for the undirected minimum cut problems using minimum range cut algorithms

Yang Dai[a], Hiroshi Imai[b], Kazuo Iwano[c], Naoki Katoh[a,*], Keiji Ohtsuka[a], Nobuhiko Yoshimura[a]

[a]*Department of Management Science, Kobe University of Commerce, 8-2-1 Gakuen-Nishimachi, Kobe 651-21, Japan*

[b]*Department of Information Science, Faculty of Science, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan*

[c]*Tokyo Research Laboratory, IBM Japan, 1623-14 Shimotsuruma, Yamato, Kanagawa 242, Japan*

Received 4 January 1993; revised 4 November 1993

Abstract

Given a connected undirected multigraph with n vertices and m edges, we first propose a new unifying heuristic approach to approximately solving the minimum cut and the s–t minimum cut problems by using efficient algorithms for the corresponding minimum range cut problems. Our method is based on the association of the range value of a cut and its cut value when each edge weight is chosen uniformly randomly from the fixed interval. Our computational experiments demonstrate that this approach produces very good approximate solutions. We shall also propose an $O(\log^2 n)$ time parallel algorithm using $O(n^2)$ processors on an arbitrary CRCW PRAM model for the minimum range cut problems, by which we can efficiently obtain approximate minimum cuts in poly-log time using a polynomial number of processors.

1. Introduction

This paper proposes a new unifying heuristic for the minimum cut and the minimum s–t cut problems by using fast minimum range cut algorithms. We demonstrate its usefulness by performing computational experiments.

Let $G = (V, E)$ be a connected undirected multigraph with n vertices and m edges. A *cut* C associated with a partition $(X, V - X)$ of the vertex set V with $X \neq \emptyset, V$ is defined as $C = \{(u, v) \in E \mid u \in X, v \in V - X\}$, and $|C|$ is called a *cut value* of C. A cut which separates two vertices s and t is called an s–t *cut*. The cut (resp. s–t cut) C with minimum $|C|$ is called a *minimum cut* (resp. s–t minimum cut) and its cut value is called

*Corresponding author. E-mail: naoki@kobeuc.ac.jp.

the *edge connectivity* (resp. *s–t connectivity*). Given a real-valued edge weight function $w(\cdot)$, the *range* of a cut C is defined as the maximum difference of weights of edges used in C, i.e., $range(C) = \max_{e \in C} w(e) - \min_{e \in C} w(e)$. The cut (resp. s–t cut) with minimum range is called the *minimum range cut* (resp. *minimum range s–t cut*).

The minimum cut problem, one of the most fundamental network problems, has been extensively studied and has many applications, including network reliability and circuit partitioning [1, 11–12, 13, 18, 27–29, 31]. For the undirected minimum cut problem, the currently fastest algorithm requires $O(m\lambda(G)\log(n^2/m))$ time [12, 13], where $\lambda(G)$ is the edge connectivity of G. For the undirected s–t minimum cut problem, the currently fastest one requires $O(\min\{n^{2/3}m, m^{3/2}\})$ time [11], which is based on flow computation. On the other hand, the minimum range cut and the minimum range s–t cut problems can be solved in $\Theta(m + n\log n)$ time [23].

Making use of the association between the range of a cut and its cut value, our heuristic method provides a different way from the conventional approaches based on flow computation. That is, when we independently assign each edge a uniform random weight in [0, 1], the cut C with the smallest range may be expected to have a small number of edges in C. If this process is iterated, it is highly probable that the cut C with the fewest edges among those generated is very close to the minimum cut. Our minimum cut heuristic algorithm repeatedly runs a $\Theta(m + n\log n)$ time minimum range cut algorithm [23] l times by assigning a new edge weight function at each iteration while maintaining a cut with the fewest edges among those generated. We have performed our experiments by typically choosing $l = \log_2 n$ or $l = \sqrt{n}$, so that the time complexity of our algorithm becomes smaller than those of exact ones. Our heuristic for the s–t minimum cut problem works almost in the same fashion as above by using a $\Theta(m + n\log n)$ time minimum range s–t cut algorithm [23].

The above approach can also be extended to the capacitated network with slight modifications as discussed in Section 2. Let $\mathcal{N} = (G = (V, E), c)$ be a capacitated undirected network, where G is an undirected simple graph with n vertices and m edges, and c is a capacity function from E to R^+. For a cut C of $\mathcal{N}$, $\sum_{e \in C} c(e)$ is called a *cut value* of C. The minimum cut and s–t minimum cut problems for $\mathcal{N}$ are then defined in a manner similar to the above case for graphs with unit capacities. The minimum cut problems for capacitated networks have also been extensively studied [1, 11, 18, 25, 29, 31], and the currently fastest minimum cut (resp. s–t minimum cut) algorithm requires $O(mn + n^2\log n)$ time [29] (resp. $O(mn\log(n^2/m))$ time [15] or $O(mn + n^{2+\varepsilon})$ time [25]).

We carry out computational experiments by using the following three types of networks: (1) NETGEN [26]: We use this widely available network generator [20] to produce 180 networks with various sizes ($n = 150 \sim 1000$) and densities up to 60%. (2) RANDOM-CAPACITATED: We generate 180 capacitated random networks in which each edge appears with a fixed probability p such that ($n = 150, 300$, $p = 0.1 \sim 0.6$), ($n = 500$, $p = 0.05 \sim 0.15$), and ($n = 1000$, $p = 0.01 \sim 0.10$). (3) TWO-CLUSTERS: We generate 120 networks with two clusters so that a cut separating two clusters is a unique minimum cut with some probability [10]. We compare our method with exact algorithms, the one by Nagamochi and Ibaraki [29] (resp. [15]) for the (resp. s–t) minimum cut problem, with measurements including relative errors to the optimal solutions, the number of trials which successfully finds the optimal, and cpu times.

Our computational experiments for the minimum cut problem demonstrate that for the NETGEN (resp. RANDOM-CAPACITATED, TWO-CLUSTERS) family our method finds exact minimum cuts with high success ratio for graphs generated, 286 cases out of 300 cases (resp. 95/180, 105/130), with much smaller cpu times than those of the NI-algorithm. It is also observed that the overall average number of iterations necessary to obtain an exact solution firstly is only 2.5 (resp. 8.5, 5.0). For the NETGEN family, within 6 iterations our algorithm computed exact minimum cuts for more than 90% of generated graphs.

For the s–t minimum cut problem, our heuristic method shows better quality than for the minimum cut problem. For example, for the NETGEN family, within $l = \log_2 n$ iterations our method finds exact s–t minimum cuts for all graphs tested with both unit capacities and general capacities and that our algorithm finds exact s–t minimum cuts for 93% of generated graphs within only 3 iterations. It is also observed that the average number of iterations necessary to obtain an exact solution firstly is only 1.7 (resp. 3.3, 4.8) for the NETGEN (resp. RANDOM-CAPACITIES, TWO-CLUSTERS) family.

From experiments, our method we think can be useful in practice, although there is no theoretical guarantee about ratio of the size of the cut found by our heuristic algorithm to the size of a minimum cut. As will be shortly remarked in Section 5, it is shown that comparing with the randomized minimum cut algorithm recently developed by Karger and Stein [21, 22], our method computes a particular minimum cut with probability higher than or equal to the one that Karger's algorithm [21, 22] does. There is, however, no theoretical guarantee that our heuristic method finds a minimum cut with strictly higher probability.

Second, as we discuss in Section 4, our heuristic methods can be effectively parallelized. Specifically, we shall propose $O(\log^2 n)$ time parallel algorithms using $O(n^2)$ processors on an arbitrary CRCW PRAM model for the minimum range cut and the minimum range s–t cut problems. Thus, if our heuristic method repeatedly uses the minimum range cut algorithm l times, it can be done in $O(\log^2 n)$ time using $O(n^2 l)$ processors. This result should be contrasted with the fact that the directed maxflow problem is P-complete [16] (i.e., hence it is not likely to have an NC algorithm).

This paper is organized as follows: In Section 2, we introduce our new heuristic approach for finding a minimum cut and an s–t minimum cut. Section 3 reports our experimental results. Section 4 presents $O(\log^2 n)$ time parallel algorithms using $O(n^2)$ processors for the minimum range cut and minimum range s–t cut problems. Finally, we conclude this paper with discussion on probabilistic analysis of our heuristic method by comparing with Karger's algorithm [21, 22].

2 New heuristic approach to minimum cut and *s-t* minimum cut problems

As mentioned in the previous section, we propose a new heuristic method to find an approximate solution for the minimum cut problem (resp. the s–t minimum cut problem) by using a fast minimum range cut (resp. minimum range s–t cut) algorithm.

We shall first explain a key idea behind our algorithm for graphs with unit edge capacities. The idea is that, if we assign each edge a weight according to independent, uniform distributions over [0, 1], we may expect that a cut with a small range has a small cut value. This observation comes from the following lemma:

Lemma 1. *For a fixed cut C, its range, range(C), is a random variable satisfying*

$$\Pr\{range(C) \leqslant x\} = |C|x^{|C|-1} - (|C|-1)x^{|C|}$$

for x with $0 \leqslant x \leqslant 1$, *and its expectation is*

$$E(range(C)) = (|C|-1)/(|C|+1).$$

Proof. Let $|C| = c$. Then, we have

$$\Pr\{range(C) \leqslant x\} = x^c + \int_x^1 c \cdot x^{c-1} \mathrm{d}t = x^c + c \cdot x^{c-1} - c \cdot x^c.$$

The first term indicates the probability that all edges have weights less than or equal to x, while the second term is the probability that all weights are between $t - x$ and t for the range $x \leqslant t \leqslant 1$. Then, we have

$$\begin{aligned} E(range(C)) &= \int_0^1 \{1 - \Pr\{range(C) \leqslant x\}\} \, \mathrm{d}x \\ &= \int_0^1 (1 - (x^c + c \cdot x^{c-1} - c \cdot x^c)) \, \mathrm{d}x \\ &= (c-1)/(c+1). \qquad \square \end{aligned}$$

From the above observation, we can naturally obtain the following heuristic algorithm for the minimum cut problem. That is, we repeatedly run a $\Theta(m + n \log n)$ time minimum range cut algorithm [23] l times by assigning a new edge weight function at each iteration, and maintain a cut with the minimum number of edges among those obtained. Here l is a prespecified parameter. The details of our algorithm are shown in Fig. 1. A heuristic s–t minimum cut algorithm can be also obtained by simply replacing the minimum range cut algorithm by a $\Theta(m + n \log n)$ time minimum range s–t cut algorithm of [23].

The above approach can be extended to the capacitated network with slight modifications. When capacities are all positive integers, we can apply our algorithms to an equivalent multigraph, which can be obtained by introducing multiple edges corresponding to the original edge capacity. However, it is costly for practical

Procedure *Approximate-Minimum-Cut*

```
(1)     $l$: = the number of iterations; $v^*$: = $+\infty$;
(2)     do $k = 1$ to $l$
(2.1)       For each $e \in E$, assign $w(e)$ a uniform random value in $[0, 1]$;
(2.2)       Compute a minimum range cut $C$;
(2.3)       if $|C| < v^*$ then $\hat{C}$: = $C$; $v^*$: = $|C|$;
        end
(3)     return($\hat{C}$);
```

Fig. 1. Algorithm Approximate-Minimum-Cut.

purposes, since each computation of a minimum range cut takes $O(mW + n \log n)$ time where W is the maximum edge weight. Instead of doing so, we devise the following practically efficient method.

As shown in [23], the edges of minimum and maximum spanning trees are sufficient for computing minimum range (s–t) cuts. Thus, instead of assigning uniform random weights to $c(e)$ multiple edges, it is sufficient to determine two random variables y and y' that correspond to minimum and maximum weights among $c(e)$ uniform random weights. This is done as follows: Let x and x' be two random numbers chosen from an independent uniform distribution over $[0, 1]$. Assume $x < x'$ for simplicity. Then, we determine y and y' from x and x' by the following equations:

$$x = 1 - (1 - y)^{c(e)/2}, \qquad x' = (y')^{c(e)/2}. \tag{1}$$

The reason why such y and y' are the desired two weights is as follows. We shall explain it only for y' (the case of y can be explained in an analogous manner). First observe that $\Pr\{x' \leqslant t\} = t^2$ for $0 \leqslant t \leqslant 1$ since x' is the maximum of two independent uniform random variables over $[0, 1]$. Thus, the distribution function of y', denoted by $F(u)$, is computed as follows:

$$F(u) = \Pr\{y' \leqslant u\} = u^{c(e)} = \Pr\{x' \leqslant u^{c(e)/2}\}.$$

Since $\Pr\{y' \leqslant u\} = \Pr\{(y')^{c(e)/2} \leqslant u^{c(e)/2}\} = \Pr\{x' \leqslant u^{c(e)/2}\}$, we have the right equation of (1). After computing such weights y and y', we introduce two multiple edges with these weights instead of generating $c(e)$ multiple edges with uniform random weights.

For the case of real capacities, we execute the same process as above after scaling all capacities so that the minimum capacity is equal to one. The justification of this process is made by using the Diophantine approximation of real numbers as was done in [29]. Notice that the above computation of y and y' can be efficiently computed, and so does for our heuristic algorithm for capacitated networks.

3. Computational experiments

3.1. Network types

We have conducted computational experiments by typically choosing the iteration number, l, as $l = \log_2 n$ and $l = \sqrt{n}$ so that our heuristic algorithms theoretically run faster than the corresponding algorithms for optimal solutions. (Exactly speaking, this statement is not correct for the minimum cut problem for graphs with unit capacities because the algorithm of [12, 13] runs in $O(m\lambda(G)\log(n^2/m))$ time, and when $\lambda(G)$ is constant, it is faster than our $O((m + n\log n)\log n)$ heuristic algorithm with $l = \log_2 n$ iterations.) For an exact algorithm of the minimum cut problem, we chose and implemented the Nagamochi and Ibaraki (NI) algorithm [29]. Although the algorithm of [12, 13] is theoretically faster than that of [29], we adopted the one of [29] because it is easy to implement and the one of [12, 13] is not applicable to capacitated networks. For the s–t minimum cut problem, we adopted the Goldberg and Tarjan (GT) algorithm [15], and used Rothberg's implementation which we obtained at DIMACS Implementation Challenge Workshop [20][1]. We adopted the GT-algorithm for the comparison purpose because as discussed in [20] it is regarded as a champion algorithm in practice among many existing maxflow algorithms, and it can be used for both of graphs with unit capacities and capacitated networks. By using cpu times of GT-algorithm as a standard measure, we think that we can evaluate how expensive our algorithm is comparing to the maximum flow computation.

We conducted our experiments for graphs generated by the following three different generators: *NETGEN*, *RANDOM-CAPACITATED*, and *TWO-CLUSTERS*.

(1) **NETGEN family** [26]: Since NETGEN is a widely available network generator and has been used as one of benchmark sets for testing network algorithms [26, 20], we use graphs generated by NETGEN as one of our benchmark sets. We generate 180 graphs, 10 graphs each for the following (n, d)(= (vertices, density)) pairs: $d = 10\%, 20\%, 30\%, 40\%, 50\%$ and 60% for $n = 150, 300$, $d = 5\%, 10\%$, and 15% for $n = 500$, and $d = 1\%, 3\%$, and 5% for $n = 1000$. In order to generate capacitated networks, we determined edge capacities by a uniform random distribution over $[1, 100]$.

(2) **RANDOM-CAPACITATED:** The second type of networks is a family of random capacitated graphs obtained as follows: First prepare n vertices and construct a connected graph by adding $n - 1$ edges forming a chain. Other edges are then selected independently randomly with probability p. The edge capacities are determined by a uniform random distribution over $[1, 100]$. We generate 120 graphs, 10 graphs each for the following (n, p)(= (vertices, probability)) pairs: $p = 0.1, 0.2, 0.3, 0.4, 0.5$, and

[1]His implementation uses FIFO queues and periodical distance label updates, and has $O(n^3)$ theoretical time complexity.

0.6 for $n = 150, 300$, $p = 0.05, 0.10$, and 0.15 for $n = 500$, and $p = 0.01, 0.05$, and 0.10 for $n = 1000$.

(3) **TWO-CLUSTERS:** The third type of graphs is a family of random graphs with two clusters defined as follows like:

(3.1) **TWO-CLUSTERS-CAPACITATED:** First prepare n vertices $\{v_i | i = 1, 2, \ldots, n\}$, and randomly partition the vertex set into two sets A and B of equal size. Then for each set choose $n/2 - 1$ edges to construct a connected component. More precisely, select an edge joining vertex $v_i \in A$ (resp. B) with vertex $v_j \in A$ (resp. B), where $j = \min\{k | k > i, v_k \in A$ (resp. B)$\}$. The other edges joining two vertices are selected with probability p. The capacities of edges joining vertices in different sets are determined by a uniform distribution over $[1, 10^4/n]$, and the capacities of edges joining vertices in the same set are determined by a uniform distribution over $[1, 10^4]$. We generate 60 graphs, 10 graphs each for the following (vertices, probability) = (n, p) pairs: for $n = 300$ $p = 0.1$, 0.2, 0.3, 0.4, 0.5, and 0.6. Notice that in this type of networks, a cut $[A, B]$ separating A and B with high probability becomes a unique minimum cut whose size is about a half of a cut separating a single vertex.

(3.2) **TWO-CLUSTERS-UNIT-CAPACITIES:** We use the model of Dyer and Frieze [10] and generate random graphs of having two clusters with unit capacities as follows. Randomly partition the vertex set into two sets A and B with $|A| = |B|$, and make both sets connected using the method for TWO-CLUSTERS-CAPACITATED above. We randomly select edges joining vertices of A with those of B with probability p_{AB}. Edges joining two vertices of A (resp. B) are selected with probabilities p_A (resp. p_B). In our experiments, we choose $p_A = p_B = 0.5$ and $p_{AB} = p_A{*}\gamma/n$ with $0.2 \leqslant \gamma \leqslant 4$. We generate 70 graphs, 10 graphs for each of $n = 300$, and $p_A = p_B = 0.5$ and $\gamma = 0.2, 0.4, 0.6, 0.8, 1.0, 2.0$, and 4.0. Notice that in this type of networks, when $\gamma \leqslant 1.0$, a cut $[A, B]$ separating A and B with high probability becomes a unique minimum cut.

We have measured the closeness of our heuristic solutions to exact ones and compared the running times of our heuristic algorithms with NI- and GT-algorithms. The results are summarized in Tables 2–9 and Figs. 2–8. Legends used in tables are summarized in Table 1. We run our heuristic algorithms, NI- and GT-algorithms on an IBM RS/6000 Model 320 workstation. We shall explain the details of our results in the following subsections. We first see results for networks generated by NETGEN, and discuss results for graphs generated by other generators.

3.2. *Results for the minimum cut problem (NETGEN family)*

Table 2 (resp. Table 3) indicates computational results for the NETGEN family. For the unit capacity case (resp. the general capacitated case), Table 2 (resp. Table 3) shows that our heuristic algorithm computes exact minimum cuts within $\sqrt{n}$ iterations for 116 cases out of 120 (resp. 170 cases out of 180). As seen from these tables, the closeness of our heuristic solutions does not seem to depend on either the number of vertices and edge densities, or whether edges have unit or general capacities.

Table 1
Legends used in Tables 2–9

Notation	Description
A.R. error	Average relative error
l_1	Number of iterations $= \log_2 n$
l_2	Number of iterations $= \sqrt{n}$
FST(l)	Average number of iterations until an exact solution is first obtained (cases for which our algorithm did not obtain exact solutions within l_2 iterations are excluded)
#	Number of cases out of 10 that our approximate minimum cut (resp. s–t minimum cut) algorithm succeeded in computing exact solutions within l_2 (resp. l_1) iterations
MR(cpu)	Average cpu time that our algorithm spends to compute approximate solutions for l_1 and l_2 iterations
NI(cpu)	Average cpu time that NI-algorithm spends
GT(cpu)	Average cpu time that GT-algorithm spends
p	Probability of selecting edge joining two vertices in graphs of RANDOM-CAPACITATED and TWO-CLUSTERS-CAPACITATED
p_A (p_B)	Probability of selecting edge joining vertices of set A (B) in TWO-CLUSTERS-UNIT-CAPACITIES graphs
p_{AB}	Probability of selecting edge joining vertices of sets A and B in TWO-CLUSTERS-UNIT-CAPACITIES graphs
Avg.	Average of values in the same category. Notice that MR(cpu) is normalized with respect to NI(cpu) or GT(cpu)
Grand avg.	Average of values in the table. Notice that MR(cpu) is normalized with respect to NI(cpu) or GT(cpu)

Regarding the computational time, we can see from these tables that our heuristic algorithm even with $\sqrt{n}$ iterations runs much faster than the NI-algorithm especially for larger graphs. This coincides with the difference of the theoretical time complexities of our algorithm and the NI-algorithm.

We can observe from Tables 2 and 3 that the cpu time of our algorithm for capacitated networks is roughly twice of that for graphs with unit capacities. This is because the overhead required for computing biased weights y and y' according to Eq. (1) is significantly large. We are currently investigating how to reduce this overhead. Notice that we can see the same behavior for our heuristic s–t minimum cut algorithm as seen from Tables 4 and 5.

Figs. 2 and 3 show the relationship between the number of iterations l and the average relative error of our best solutions obtained within l iterations. We see that the overall average number of iterations necessary to obtain an exact solution firstly is 2.5. It is surprising that this value does not change much depending upon the number of vertices or graph density. Closely examining our experimental results, it was

Table 2
Summary of results for minimum cuts for NETGEN graphs with unit capacities

Node	Density (%)	A.R. error (%) l_1	l_2	FST(l)	#	MR(cpu)(s) l_1	l_2	NI(cpu) (s)
	10	3.2	0.0	3.2	10	2.1	3.4	6.1
	20	2.9	0.9	2.9	9	2.7	4.3	7.4
150	30	3.5	1.1	3.0	9	3.4	5.4	8.9
	40	3.0	0.6	2.1	9	4.0	6.5	10.2
	50	1.8	0.5	1.4	9	4.6	7.4	11.6
	60	1.3	0	1.3	10	5.3	8.6	13.0
	Avg.	2.6	0.5	2.3	9.3	0.4	0.6	1.0
	10	2.9	0.0	4.7	10	6.3	12.7	47.8
	20	3.3	0.0	2.8	10	9.3	18.5	58.4
300	30	0.0	0.0	1.4	10	12.5	25.0	70.1
	40	0.0	0.0	2.5	10	15.1	30.2	80.0
	50	1.7	0.0	2.7	10	18.8	37.6	91.9
	60	0.0	0.0	1.6	10	21.5	42.9	103.3
	Avg.	1.3	0.0	2.6	10.0	0.2	0.4	1.0
Grand avg.		2.0	0.3	2.5	9.7	0.3	0.5	1.0

revealed that within 6 iterations, our algorithm computed exact minimum cuts for more than 90% of generated graphs.

3.3. Results for the s–t minimum cut problem (NETGEN family)

We performed similar experiments for s–t minimum cut problems. The results are summarized in Tables 4 and 5. For the unit capacity case, Table 4 shows that our algorithm produced optimal solutions for all cases with only $l = \log_2 n$ iterations except the case when $n = 150$ and $d = 10\%$. For the general capacity case, Table 5 shows that our algorithm produced optimal solutions for all 180 cases with only $l = \log_2 n$ iterations.

Thus, the closeness of our heuristic solutions is even better than the minimum cut problem. In addition, as in the minimum cut problem, the closeness of our heuristic solutions does not seem to depend on either the number of vertices and edge densities or whether edges have unit or general capacities.

Figs. 4 and 5 indicate the relationship between the number of iterations l and the average relative error of our best solutions obtained within l iterations. We see that the overall average number of iterations necessary to obtain an exact solution firstly is 1.5. As in the case of minimum cuts, it is surprising that this value does not change much depending upon the number of vertices or graph density. From details of our results, it was revealed that within 3 iterations, our algorithm computed exact s–t minimum cuts for 93% of generated graphs. Regarding the cpu time, our algorithm

Table 3
Summary of results for minimum cuts for NETGEN capacitated graphs

Node	Density (%)	A.R. error (%) l_1	l_2	FST(l)	#	MR(cpu)(s) l_1	l_2	NI(cpu) (s)
	10	0.0	0.0	1.9	10	3.4	5.5	6.1
	20	2.8	1.9	3.1	9	4.8	7.7	7.4
150	30	6.0	6.0	2.3	8	6.3	10.2	8.9
	40	8.4	0.4	2.8	9	7.8	12.7	10.2
	50	2.4	0.5	3.6	8	9.3	15.1	11.6
	60	1.3	1.3	1.3	9	10.8	17.4	12.9
	Avg.	3.5	1.7	2.5	8.8	0.7	1.2	1.0
	10	3.7	0.0	3.8	10	10.5	20.9	47.2
	20	0.0	0.0	1.3	10	17.3	34.3	57.9
300	30	0.0	0.0	1.7	10	24.1	48.1	69.2
	40	0.0	0.0	2.0	10	30.0	59.7	78.8
	50	7.5	0.2	2.4	8	38.1	75.8	91.7
	60	0.0	0.0	2.2	10	43.9	87.5	102.8
	Avg.	1.9	0.0	2.2	9.7	0.3	0.7	1.0
	5	7.7	0.0	5.6	10	15.2	38.8	191.8
500	10	0.0	0.0	1.2	10	24.7	63.0	216.6
	15	0.0	0.0	1.6	9	34.3	87.5	242.7
	Avg.	2.6	0.0	2.8	9.7	0.1	0.3	1.0
	1	0.0	0.0	2.2	10	25.3	81.2	1370.7
1000	3	0.0	0.0	3.3	10	44.3	141.4	1443.0
	5	0.0	0.0	1.9	10	63.8	203.8	1519.1
	Avg.	0.0	0.0	2.5	10.0	0.0	0.1	1.0
Grand avg.		2.2	0.6	2.5	9.4	0.4	0.7	1.0

with $\log_2 n$ iterations is slower than the GT-algorithm when the graph density is low. However, its cpu time becomes comparable when the graph density becomes high. In particular, taking into consideration the fact that our algorithm probably computes exact solutions within three iterations, we can see that our algorithm is faster than the GT-algorithm for graphs with high density.

We can observe, by comparing Tables 2 and 3 with Tables 4 and 5, that our heuristic algorithm for the s–t minimum cut problem is slightly slower than that for the minimum cut problem. As shown in [23] (also see Section 4), the minimum range s–t cut is the one with minimum range separating s and t among $n-1$ upper critical cuts (see [23] or Section 4 for the definition) that the minimum range cut algorithm generates. Thus, we need to test whether each generated upper critical cut separates s and t. This is the extra work that the minimum range s–t cut algorithm [23] requires over the minimum range cut algorithm. The time for this work can be regarded as the difference between the cpu times of approximate minimum and s–t minimum cut algorithms. We can see from columns MR(cpu) of Tables 2–5 that it is relatively

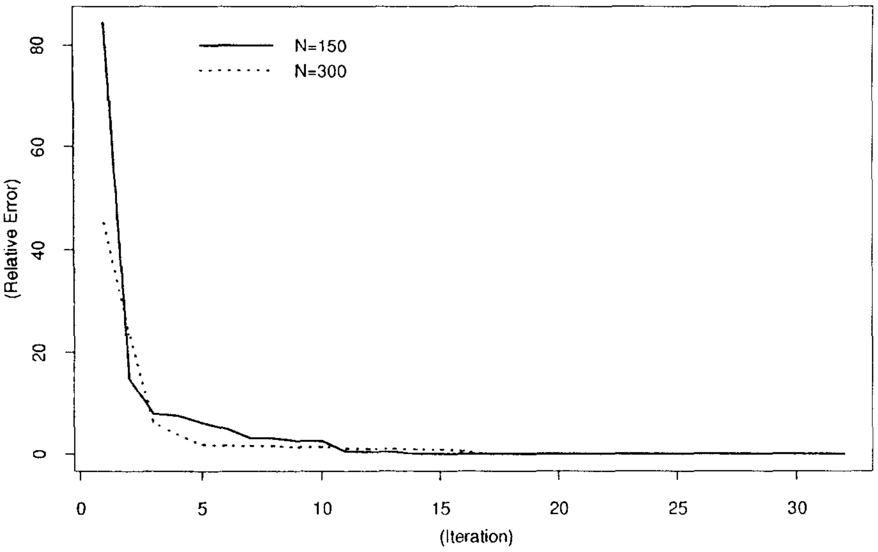

Fig. 2. Relative errors vs. the number of iterations for NETGEN graphs with unit capacities for the minimum cut problem.

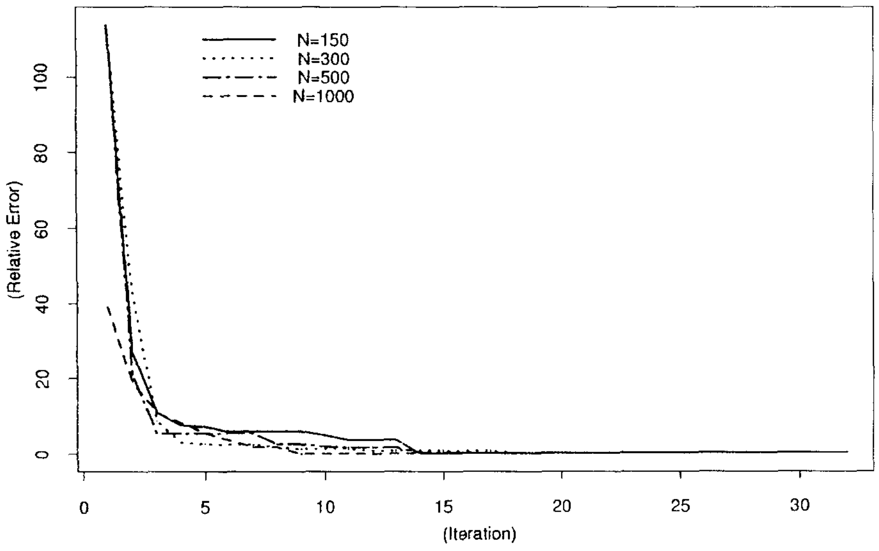

Fig. 3. Relative errors vs. the number of iterations for capacitated NETGEN graphs for the minimum cut problem.

small compared with the overall cpu time. Thus, if we use our heuristic s–t minimum cut algorithm for multiple pairs of s and t, the extra time per pair seems to be very small. On the other hand, if we use the GT-algorithm for this case, we need to apply it from scratch for each pair. Therefore, in this case, it is expected that our heuristic

Table 4
Summary of results for s–t minimum cuts for graphs with NETGEN unit capacities

Node	Density (%)	A.R. error (%) l_1	l_2	FST(l)	#	MR(cpu)(s) l_1	l_2	GT(cpu) (s)
	10	2.0	0.0	2.4	10	2.7	4.3	0.1
	20	0.0	0.0	1.3	10	3.2	5.2	0.3
150	30	0.0	0.0	1.2	10	3.9	6.3	0.5
	40	0.0	0.0	1.9	10	4.6	7.4	0.9
	50	0.0	0.0	1.5	10	5.2	8.3	1.9
	60	0.0	0.0	1.4	10	5.9	9.4	2.6
	Avg.	0.3	0.0	1.6	10.0	9.3	14.9	1.0
	10	0.0	0.0	1.6	10	7.6	15.2	0.5
	20	0.0	0.0	1.6	10	10.5	21.0	1.7
300	30	0.0	0.0	1.4	10	15.6	31.0	4.9
	40	0.0	0.0	2.1	10	16.3	32.5	7.1
	50	0.0	0.0	1.5	10	19.6	39.3	12.4
	60	0.0	0.0	2.1	10	22.4	44.8	22.4
	Avg.	0.0	0.0	1.7	10.0	4.9	9.8	1.0
Grand avg.		0.2	0.0	1.7	10.0	7.1	12.3	1.0

algorithm will become faster than using the GT-algorithm as the number of s–t pairs increases. Furthermore, since there exists an $O(n^2)$ time all s–t minimum range cuts algorithm [23], we can apply that algorithm as our heuristic method for finding approximate all s–t minimum cuts. Performing computational experiments for this problem is left for the future research.

3.4. *Results for RANDOM-CAPACITATED and TWO-CLUSTERS families*

RANDOM-CAPACITATED family: Table 6 (resp. Table 7) indicates experimental results for the minimum cut (resp. s–t minimum cut) problem for the RANDOM-CAPACITATED family.

As observed from the tables and figures, we can see phenomena similar to those for the NETGEN family. Although the number of cases in which an optimal solution was obtained is less than that for the NETGEN family, our method still finds optimal solutions in 95 cases out of 180 for the minimum cut problem and in 107 cases out of 180 for the s–t minimum cut problem.

As for the cpu time, we observed that our method runs much faster than the NI-algorithm for the minimum cut problem and runs much slower than GT for the single s–t minimum cut problem. For the minimum cut problem, Table 6 shows that as a graph becomes larger and denser, the average relative error (A.R. error) becomes bigger. However, notice that the average relative error with $\sqrt{n}$ iterations even for

Table 5
Summary of results for s-t minimum cuts for NETGEN capacitated networks

Node	Density (%)	A.R. error (%) l_1	l_2	FST(l)	#	MR(cpu)(s) l_1	l_2	GT(cpu) (s)
	10	0.0	0.0	2.0	10	3.7	6.0	0.1
	20	0.0	0.0	1.4	10	5.0	8.1	0.5
150	30	0.0	0.0	1.7	10	6.4	10.5	0.8
	40	0.0	0.0	1.9	10	7.9	12.8	1.4
	50	0.0	0.0	1.4	10	9.4	15.2	3.0
	60	0.0	0.0	1.7	10	10.8	17.4	4.0
	Avg.	0.0	0.0	1.7	10.0	11.1	18.0	1.0
	10	0.0	0.0	2.2	10	11.1	22.1	1.0
	20	0.0	0.0	1.5	10	17.4	34.7	2.7
300	30	0.0	0.0	1.8	10	23.8	47.6	6.6
	40	0.0	0.0	1.8	10	30.0	59.0	9.6
	50	0.0	0.0	2.2	10	37.8	75.3	18.2
	60	0.0	0.0	1.4	10	44.2	88.0	32.2
	Avg.	0.0	0.0	1.8	10.0	4.6	9.2	1.0
	5	0.0	0.0	1.6	10	16.8	43.0	1.5
500	10	0.0	0.0	1.6	10	26.5	67.8	3.7
	15	0.0	0.0	2.1	10	36.2	91.7	10.3
	Avg.	0.0	0.0	1.8	10.0	7.3	18.6	1.0
	1	0.0	0.0	1.3	10	30.0	95.2	0.8
1000	3	0.0	0.0	1.3	10	47.2	150.2	3.4
	5	0.0	0.0	2.3	10	65.9	210.6	9.5
	Avg.	0.0	0.0	1.6	10.0	19.4	61.8	1.0
Grand avg.		0.0	0.0	1.7	10.0	9.7	22.5	1.0

largest and densest graphs tested ($n = 1000$ and $p = 0.10$) remains 3.85%, while the cpu time of our method is 5.8 times smaller than that of the NI-algorithm. For the s–t minimum cut problem, Table 7 shows that the average relative errors remain at a small level within 0.6%. However, considering much smaller cpu times of the GT-algorithm, the GT-algorithm would be the choice for finding an s–t minimum cut for a single pair. Notice that for multiple s–t minimum cuts problem our method can be used as described in the previous subsection.

TWO-CLUSTERS family: Table 8 (resp. Table 9) indicates experimental results for the minimum cut problem for the TWO-CLUSTERS-CAPACITATED (resp. TWO-CLUSTERS-UNIT-CAPACITY) family. Notice that in a graph of TWO-CLUSTERS-CAPACITATED, there is a unique minimum cut and at a first glance it seems to be hard for our heuristic to identify such a unique minimum cut. This is because there exist exponentially many non-minimum cuts, and hence the minimum range among those of non-minimum cuts may be smaller than that of a unique minimum cut. However, as Table 8 shows, it finds the minimum cut in 49 cases out of 60 with

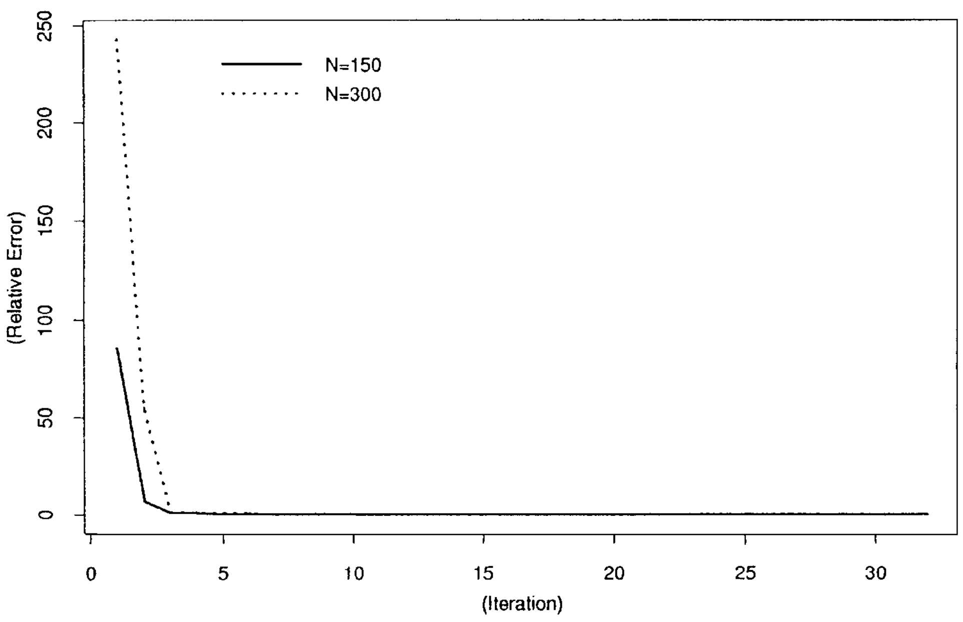

Fig. 4. Relative errors vs. the number of iterations for NETGEN graphs with unit capacities for the s–t minimum cut problem.

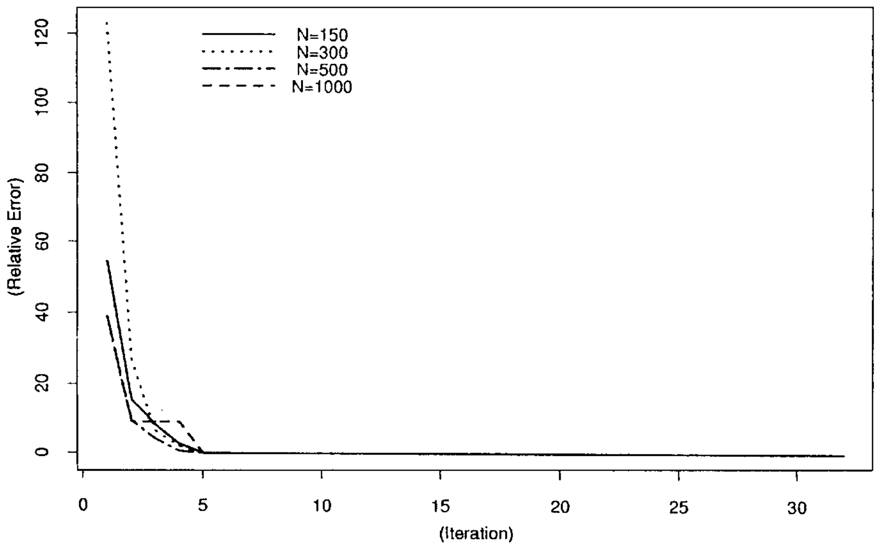

Fig. 5. Relative errors vs. the number of iterations for capacitated NETGEN graphs for the s–t minimum cut problem.

Table 6
Summary of results for minimum cuts for RANDOM-CAPACITATED networks

Node	p	A.R. error (%)		FST(l)	#	MR(cpu)(s)		NI(cpu)
		l_1	l_2			l_1	l_2	(s)
	0.1	0.00	0.00	4.0	10	3.25	5.21	6.33
	0.2	1.27	0.00	5.4	10	4.49	7.22	7.92
150	0.3	4.95	0.00	7.4	10	5.76	9.29	9.43
	0.4	4.82	0.00	4.8	10	7.15	11.54	10.66
	0.5	4.90	0.59	6.7	6	8.42	13.58	12.04
	0.6	5.55	2.21	8.8	4	9.69	15.73	13.41
	Avg.	3.6	0.5	6.2	8.3	0.6	1.0	1.0
	0.1	6.84	1.81	12.3	7	10.28	20.43	49.96
	0.2	7.35	3.02	7.7	3	16.35	32.53	62.37
300	0.3	5.41	2.17	11.0	3	22.45	44.69	74.67
	0.4	7.93	6.49	1.5	2	28.62	57.09	86.28
	0.5	7.59	6.70	11.0	3	34.24	68.38	98.58
	0.6	10.83	6.56	14.0	1	40.50	80.87	110.66
	Avg.	7.7	4.5	9.6	3.2	0.3	0.6	1.0
	0.05	7.93	2.13	11.0	4	15.76	40.23	199.20
500	0.10	7.91	6.92	8.7	3	24.16	61.50	229.91
	0.15	5.22	3.04	8.2		32.57	82.89	259.75
	Avg.	7.0	4.0	9.3	3.7	0.1	0.3	1.0
	0.01	0.00	0.00	3.0	10	27.64	87.87	1391.54
1000	0.05	8.12	3.72	14.7	3	58.62	187.26	1563.78
	0.10	6.90	3.85	13.0	2	98.22	313.48	1809.50
	Avg.	5.0	2.5	10.2	5.0	0.0	0.1	1.0
Grand avg.		5.8	2.7	8.5	5.3	0.3	0.6	1.0

smaller CPU time than the NI-algorithm. As the table shows the average relative error of $\log_2 n$ iterations becomes prohibitively large more than 15% when a graph becomes denser ($p \geqslant 0.5$). However, the average relative error of $\sqrt{n}$ iterations remains within a satisfiable level (less than 7%). For unit capacity cases (TWO-CLUSTERS-UNIT-CAPACITIES family), we can observe such good performance in Table 9, in which our heuristic finds the optimal solution in 56 cases out of 70 cases. From Table 9, where $\gamma \leqslant 1.0$ (that is, a cut $[A, B]$ separating two sets A and B is probably a unique cut of size less than a half of a cut separating a single vertex), our algorithm finds optimal cuts in all cases. Fig. 8 shows that when γ becomes smaller, the number of iterations for firstly finding an optimal solution becomes also smaller. Notice also that the case of $\gamma = 1.0$ corresponds to the construction of TWO-CLUSTERS-CAPACITATED and results in Table 8 seem to be similar to those in Table 9 with $\gamma \leqslant 1.0$.

Table 7
Summary of results for s–t minimum cuts for RANDOM-CAPACITATED networks

Node	p	A.R. error (%)		FST(l)	#	MR(cpu)(s)		GT(cpu)
		l_1	l_2			l_1	l_2	(s)
	0.1	0.00	0.00	3.1	10	3.85	6.18	0.19
	0.2	0.00	0.00	2.6	10	5.07	8.18	0.52
150	0.3	0.00	0.00	1.8	10	6.37	10.29	1.04
	0.4	9.12	1.41	3.8	4	7.77	12.57	2.36
	0.5	0.99	0.99	3.8	4	9.02	14.60	3.69
	0.6	0.53	0.53	4.8	6	10.29	16.65	4.97
	Avg.	1.8	0.5	3.3	7.3	7.3	11.8	1.0
	0.1	0.20	0.20	2.8	9	11.57	23.02	1.59
	0.2	0.42	0.42	2.3	7	17.56	35.00	5.09
300	0.3	1.49	0.09	6.4	8	23.53	46.90	10.87
	0.4	0.22	0.22	4.8	8	29.61	59.12	19.04
	0.5	2.19	0.29	6.5	6	35.99	71.69	29.17
	0.6	0.40	0.40	3.0	5	41.90	83.86	44.95
	Avg.	0.8	0.3	4.3	7.2	2.8	5.5	1.0
	0.05	0.24	0.24	2.4	9	18.09	46.00	1.84
500	0.10	0.19	0.19	3.4	8	26.35	67.05	6.19
	0.15	0.29	0.29	2.4	8	34.81	88.73	13.75
	Avg.	0.2	0.2	2.7	8.3	5.5	14.1	1.0
	0.01	0.00	0.00	1.5	10	32.05	101.92	1.21
1000	0.05	0.61	0.61	2.9	7	62.91	201.01	15.56
	0.10	0.28	0.28	1.8	8	102.87	327.75	51.74
	Avg.	0.3	0.3	2.1	8.3	10.8	34.5	1.0
Grand avg.		0.9	0.3	3.3	7.6	6.1	13.9	1.0

Summarizing the above experimental results in various types of networks for both minimum and s–t minimum cut problems, our method seems to be promising for practical purposes, because its general framework is simple and minimum range cut algorithms of [23] are also easy to implement. We believe that our approach gives a new way to approximately solve the minimum cut and the s–t minimum cut problems because it does not involve any flow algorithm. However, a challenging question seems to be how to theoretically estimate the closeness of a heuristic solution $\hat{C}$ to the optimal.

4. Parallel algorithm

In this section, we show that a minimum range cut can be computed in $O(\log^2 n)$ time using $O(n^2)$ processors on an arbitrary CRCW PRAM model. Thus, the minimum

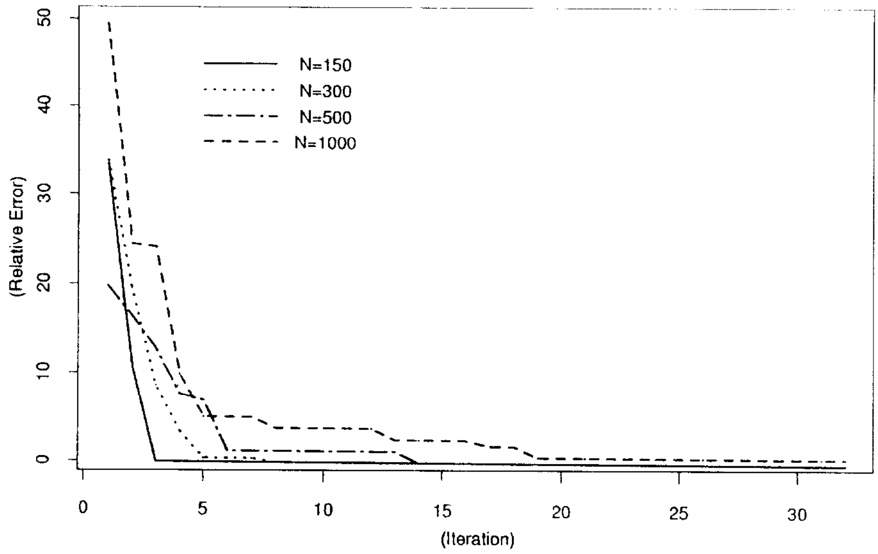

Fig. 6. Relative errors vs. the number of iterations for RANDOM-CAPACITATED graphs for the minimum cut problem ($p = 0.1$).

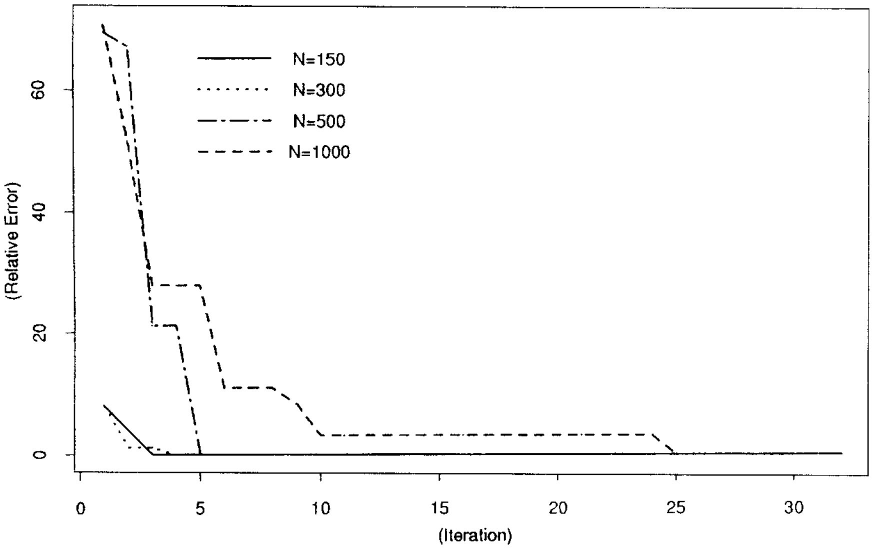

Fig. 7. Relative errors vs. the number of iterations for RANDOM-CAPACITATED graphs for the s–t minimum cut problem ($p = 0.1$).

range cut problem belongs to NC. Since our approach for finding approximate minimum cuts consists of independent (thus, parallelizable) iterations, we can find an approximate minimum cut in $O(\log^2 n)$ time using $O(n^2 l)$ processors where l is the number of iterations.

Table 8
Summary of results for TWO-CLUSTERS-CAPACITATED networks ($n = 300$).

	p	A.R. error (%)		FST(l)	#	MR(cpu)(s)		NI(cpu)
		l_1	l_2			l_1	l_2	(s)
	0.1	0.00	0.00	3.2	10	10.07	20.03	50.09
	0.2	6.79	0.00	5.7	10	16.08	32.21	61.89
Min. cut	0.3	4.72	0.38	7.4	9	22.58	44.92	72.65
	0.4	2.37	0.72	5.9	9	28.72	57.41	82.78
	0.5	6.35	3.26	4.2	6	34.95	69.72	95.39
	0.6	15.52	6.72	8.4	5	41.85	83.37	105.19
	Avg.	6.0	1.9	5.8	8.2	0.3	0.6	1.0
	p	A.R. error (%)		FST(l)	#	MR(cpu)(s)		GT(cpu)
		l_1	l_2			l_1	l_2	(s)
	0.1	0.00	0.00	1.5	10	11.43	22.76	6.94
	0.2	0.00	0.00	2.5	10	17.18	34.26	20.10
s–t min. cut	0.3	0.00	0.00	2.3	10	23.38	46.69	34.48
	0.4	12.19	0.00	4.8	10	29.58	59.03	67.32
	0.5	29.38	0.02	5.3	10	35.70	71.23	95.51
	0.6	20.84	0.02	6.9	9	42.14	84.18	118.55
	Avg.	10.4	0.0	3.9	9.8	0.7	1.4	1.0
Grand avg.		8.2	0.9	4.8	9.0	0.5	1.0	1.0

Table 9
Summary of results for minimum cuts for TWO-CLUSTERS-UNIT-CAPACITIES ($n = 300$, $p_A = p_B = 0.5$, $p_{AB} = p_A{}^*\gamma/n$)

γ	A.R. error (%)		FST(l)	#	MR(cpu)(s)		GT(cpu)
	l_1	l_2			l_1	l_2	(s)
0.2	0.00	0.00	1.0	10	10.47	20.90	67.30
0.4	0.00	0.00	1.6	10	10.37	20.70	67.43
0.6	0.00	0.00	4.0	10	10.42	20.78	67.48
0.8	0.00	0.00	2.8	10	10.42	20.78	67.46
1.0	0.00	0.00	4.1	10	10.40	20.87	67.63
2.0	6.76	5.61	6.3	3	10.52	21.02	67.58
4.0	9.09	6.61	10.7	3	10.61	21.18	67.69
Avg.	2.3	1.8	4.4	8.0	0.2	0.3	1.0

First, we define some notations. Let $w(\cdot)$ be a real-valued edge weight function. For the sake of simplicity, we assume that all edge weights are distinct. When some edge weights are not distinct, we can handle them as in [23]. For a cut C, we define $\max(C) \in C$ (resp. $\min(C)$) as the edge whose weight is maximum (resp. minimum) among edges in C. Then *range*$(C) = w(\max(C)) - w(\min(C))$. Let $T_{\min}$ (resp. $T_{\max}$) be a minimum (resp. maximum) spanning tree in G. Then we have the following lemma [23].

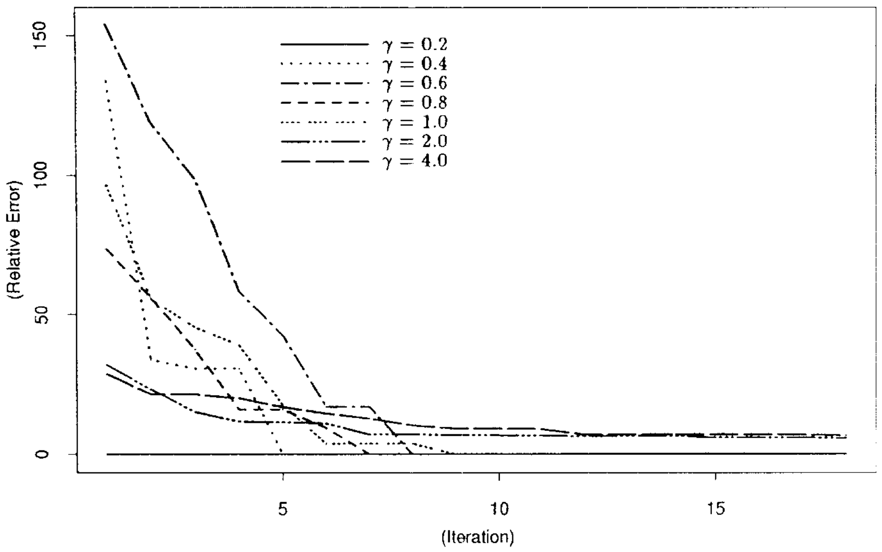

Fig. 8. Relative errors vs. the number of iterations for TWO-CLUSTERS-UNIT-CAPACITIES graphs for the minimum cut problem.

Lemma 2. *For any cut C, we have* $\max(C) \in T_{\max}$ *and* $\min(C) \in T_{\min}$.

From the above lemma, without loss of generality, we can assume that $E = T_{\min} \cup T_{\max}$ and $m \leqslant 2(n-1)$. Let $w_1, w_2, \ldots, w_{n-1}$ be the edge weights of $T_{\max}$ in their increasing order. Let $E[w, w'] = \{e \in E \mid w \leqslant w(e) \leqslant w'\}$. For an edge $e \in T_{\min}$, we define an *upper-critical value*, *upper*$(w(e))$, *associated with* $w(e)$ by the weight $w_i (= w(f))$ such that $f \in T_{\max}$ and $E[w(e), w_i]$ forms a cut, but $E[w(e), w_{i-1}]$ does not. The cut $E[w(e), w_i]$ is called the *upper-critical cut with respect to e*.

When such $f \in T_{\max}$ does not exist, *upper*$(w(e))$ is not defined. From Lemma 2, the minimum range can be given as the minimum of $w(\mathit{upper}(w(e))) - w(e)$ over all $e \in T_{\min}$. Thus, we can devise the parallel algorithm as shown in Fig. 9.

For (1), we have an algorithm on CREW PRAM which runs in $O(\log^2 n)$ with total work of $O(n^2)$ [32, 3, 19].

For (2), Cole's merge-sort algorithm [4] runs in $O(\log n)$ time with $O(n)$ processors on a CREW PRAM model.

For (3), we find *upper*$(w(e))$ for each $e \in T_{\min}$ in parallel. Given $e \in T_{\min}$, we can find *upper*$(w(e))$ by using a binary search of $O(\log n)$ iterations, in each of which step checks whether $E[w(e), w(f)]$ forms a cut or not for $f \in T_{\max}$ by using a connectivity algorithm. For the connectivity problem, it takes $O(\log n)$ time using $O(n+m)$ processors (which is $O(n)$ processors since $m \leqslant 2(n-1)$ in our case) on an arbitrary CRCW PRAM model [33, 2, 19]. Thus, Step (3) requires $O(\log^2 n)$ time using $O(n^2)$ processors on an arbitrary CRCW PRAM model.

Procedure *Parallel-Minimum-Range-Cut*

```
(1)    Find $T_{\min}$ and $T_{\max}$. Let $E = T_{\min} \cup T_{\max}$.
(2)    Sort edges in $T_{\max}$ with respect to their edge weights.
(3)    For each $e \in T_{\min}$, find an $upper(w(e))$.
(4)    Determine the minimum range from $\{w(upper(w(e))) - w(e) | e \in T_{\min}\}$.
(5)    Determine a set of edges which forms a minimum range cut.
```

Fig. 9. Algorithm Parallel-Minimum-Range-Cut.

For (4), there is an $O(\log n)$ time EREW algorithm using $O(n)$ processors which uses a simple tournament method. For (5), suppose $E[w(e), w(f)]$ forms a minimum range cut such that $e \in T_{\min}$ and $f \in T_{\max}$ which are found in Step (4) : that is, f is the edge in $T_{\max}$ satisfying $w(f) = upper(w(e))$. This can be done in $O(\log n)$ time using $O(n)$ processors on an arbitrary CRCW PRAM model [33, 2]. We first find connected components after deleting $E[w(e), w(f)]$ from G, which takes $O(\log n)$ time using $O(n)$ processors. Let X be a connected component found. Mark each edge which belongs to the cut of $(X, V - X)$, which can be done by assigning a processor to each edge, and thus which takes $O(1)$ time using $O(n)$ processors.

Therefore, our parallel algorithm for finding a minimum range cut takes $O(\log^2 n)$ time using $O(n^2)$ processors on an arbitrary CRCW PRAM model.

The modification of this algorithm to deal with the minimum range s–t cut problem can be done in a straightforward manner. For this, we shall cite the following fact [23]: Let $E[w(e), w(f)]$ be an upper-critical cut with respect to $e \in T_{\min}$. Then the minimum range s–t cut is the one with minimum range among all upper-critical cuts that separate s and t. Using this fact, we only need to add the routine in Step (3) to test whether s and t belong to different connected components associated with each upper-critical cut. This does not affect the overall time complexity.

Theorem 3. *We can find a minimum range cut and a minimum range s–t cut in* $O(\log^2 n)$ *time using* $O(n^2)$ *processors on an arbitrary CRCW PRAM model.*

5. Conclusion

We proposed a unifying heuristic method for approximately solving the minimum (s–t minimum) cut problem by using a fast minimum range cut (s–t cut) algorithm. This heuristic method is based on the observation that a minimum range cut with uniform random edge weights in [0, 1] can be a good estimator of a minimum cut.

According to our experiments, our method found exact minimum cuts with high success ratio (480 cases out of 610) much faster than the NI-algorithm. For the s–t minimum cut problem, our method exhibited even better performance on relative errors with larger cpu times for the NETGEN and RANDOM-CAPACITATED families and with comparable cpu time for the TWO-CLUSTERS family. However, as discussed in Section 3, if we are interested in finding multiple (or all) pairs of s–t minimum cuts, our heuristic method runs faster than applying the GT-algorithm to each pair from scratch (or Gomory-Hu cut tree algorithm).

From these results, we see that our methods are quite promising for practical purposes. In particular, as far as the ratio that our methods find exact solutions is concerned, our methods for both of minimum cut and s–t minimum cut problems are the most successful for graphs of small size or low density. For graphs of large size and/or high density, our method for minimum cut problems produces solutions close to the optimal by spending cpu time significantly less than the NI-algorithm. Thus, for minimum cut problems, if taking cpu times into account, our method is effective for graphs of large size and/or high density. However, for s–t minimum cut problems, our method seems to be less effective since the GT-algorithm runs faster in general than our method.

The future research includes the further extensive experiments verifying the current results and the refinement of the current methods such as new heuristic which makes use of the information obtained from previous iterations for the next iteration.

In summary, our methods have been developed based on the following key observations:

(1) In general, minimum range problems can be solved more efficiently than the corresponding minimum-sum problems.

(2) When we assign each edge a uniform random weight in $[0, 1]$ the edge set C with the smallest range may be expected to have a small number of edges in C.

These observations are very general, and hence can be further applied to other problems. In fact, we have already developed heuristic algorithms for two related but much more difficult (i.e., NP-complete) problems [7–9, 30]; the *k-multiway split problem* [17] and the balanced cut problem (also called the *graph partitioning problem*) [24]. They are obtained by newly devising a $\Theta(m + n\log n)$ time algorithm for the minimum range k-multiway split problem [8] and an $O(m + n^{2.5})$ time minimum range balanced cut algorithm [7, 30]. Preliminary computational experiments have also been reported [8, 30], and our recent paper [9] shows that for small k such as $k = 3, 4$, and 6 our heuristic algorithm for the k-multiway split problem produces acceptable solutions for graphs generated by random graph generators similar to those used in our experiments. The idea can also be applied to obtain a heuristic algorithm for the *k-multiway cut problem* [5, 6].

Finally, it should be noted that the most challenging problem is how to theoretically estimate the closeness of our heuristic solution. Recently, Karger and Stein devised an $O(n^2 \log^3 n)$ time randomized minimum cut algorithm based on contraction operations like ours [21, 22]. As Karger discussed in [21], when we randomly rank edges, construct a minimum spanning tree, and then remove the heaviest edge from the minimum spanning tree, we have two components. Then, it was shown that the cut defined by these components is in fact a particular minimum cut with $\Omega(n^{-2})$ probability. Notice that as discussed in [23] and in Section 4, our minimum range cut algorithm repeatedly finds $(n - 1)$ upper-critical cuts, each of which has one edge from a maximum spanning tree and one of which can be associated with the cut described above by Karger. Thus, at each run of a minimum range cut algorithm, we can find a cut corresponding to a cut obtained by a single iteration of Karger's algorithm. Moreover, a minimum range cut algorithm reports $(n - 1)$ critical cuts at each execution, which may increase the probability of finding a particular minimum cut. However, the difficulty of probabilistic analysis arises in that these critical cuts are interrelated each other in the following sense. That is, when starting with random edge weights, $(n - 1)$ critical cuts can be obtained successively by cyclic shifts of the initial edge weights. Thus, we can not extend Karger's analysis technique directly. Another difficulty arises in estimating the probability of that a minimum cut has a smaller range than other cuts since there are exponentially many other cuts, although given two particular cuts we can estimate the probability of that a smaller cut has a smaller range.

From Karger and Stein's results [21, 22] and the above observation, it is guaranteed that applying our heuristic algorithm $O(n^2 \log n)$ iterations produces a minimum cut with high probability, while our experimental results demonstrate that our heuristic method produces solutions very close to optimal within only $\sqrt{n}$ iterations. Thus, there is a big gap between theoretical and experimental results. Closing this gap theoretically would be indeed a challenging open problem. One possible direction of future research along this line is to carry out precise probabilistic analysis of our heuristic algorithm so as to establish a theoretically significant distinction between two probabilities that Karger's approach [21] and our heuristics find a particular minimum cut. Another direction is to present a bound stronger than $\Omega(n^{-2})$ on the probability that Karger's approach [21] finds a particular minimum cut for random graphs.

Acknowledgements

The authors would like to express their gratitude to anonymous referees for carefully reading the previous version of the paper and giving useful comments and suggestions which greatly helped to improve the quality of the paper.

References

[1] R.K. Ahuja., T.L. Magnanti and J.B. Orlin, Network Flows: Theory, Algorithms, and Applications (Prentice Hall, Englewood Cliffs, NJ, 1992).

[2] B. Awerbuch and Y. Siloach, New connectivity and MSF algorithm for ultra-computer and PRAM, IEEE Trans. Comput. 36 (1987) 1258–1263.

[3] F.Y. Chin, J. Lam and I. Chen, Efficient parallel algorithms for some graph problems, Comm. ACM 25 (1982) 659–665.

[4] R. Cole, Parallel merge sort, SIAM J. Comput. 17 (1988) 770–785.

[5] W.H. Cunningham, The optimal multiterminal cut problem, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science 5 (American Mathematical Society, Providence, RI, 1991) 105–120.

[6] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour and M. Yannakakis, The complexity of multiway cuts, in: Proceedings of the Twenty Fourth Annual ACM Symposium on the Theory of Computing (1992) 241–251.

[7] Y. Dai, H. Imai, K. Iwano and N. Katoh, How to treat delete requests in semi-online problems, in: Proceedings of Fourth Annual International Symposium on Algorithms and Computation, Lecture Notes in Computer Science (Springer, Berlin, 1993) 48–59.

[8] Y. Dai, H. Imai, K. Iwano, N. Katoh, K. Ohtsuka and N. Yoshimura, A new unified approximate approach to the minimum cut problem and its variants using minimum range cut algorithms, IBM Research Report, RT0082 (1992).

[9] Y. Dai, K. Iwano and N. Katoh, A new approximate algorithm for the minimum k-cut problem by using minimum range k-cut algorithm, Publ. Electron. Inform. System Soc. Japan C-144 (1994) 438–443 (in Japanese).

[10] M.E. Dyer and A.M. Frieze, The solution of some random NP-hard problems in polynomial expected time, J. Algorithms 10 (1989) 451–489.

[11] S. Even and R.E. Tarjan, Network flow and testing graph connectivity, SIAM J. Comput. 4 (1975) 507–518.

[12] H.N. Gabow, A matroid approach to finding edge connectivity and packing arborescences, in: Proceedings of the Twenty Third Annual ACM Symposium on the Theory of Computing (1991) 112–122.

[13] H.N. Gabow, Applications of a poset representation to edge connectivity and graph rigidity, in: Proceedings of 32nd Annual Symposium on Foundations of Computer Science (1991) 812–821.

[14] M.R. Garey and D.S. Johnson, Computers and Intractability – A Guide to the Theory of NP-Completeness (Freeman, New York, 1979).

[15] A.V. Goldberg and R.E. Tarjan, A new approach to the maximum flow problem, J. ACM 35 (1988) 921–940.

[16] L.M. Goldschlager, R.A. Shaw and J. Staples, The maximum flow problem is logspace complete for P, Theoret. Comput. Sci. 21 (1982) 105–111.

[17] O. Goldschmidt and D.S. Hochbaum, Polynomial algorithm for the k-cut problem, in: Proceedings of 29nd Annual Symposium on Foundations of Computer Science (1988) 444–451.

[18] J. Hao and J.B. Orlin, A faster algorithm for finding the minimum cut in a graph, in: Proceedings of the Third Annual ACM/SIAM Symposium on Discrete Algorithms (1992) 165–174.

[19] J. JáJá, An Introduction to Parallel Algorithms (Addison-Wesley, Reading, MA, 1992).

[20] D.S. Johnson and C.C. McGeoch, eds., Network Flows and Matching, First DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 12 (American Mathematical Society, Providence, RI, 1993).

[21] D.R. Karger, Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm, in: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (1993) 21–30.

[22] D.R. Karger and C. Stein, An $\tilde{O}(n^2)$ algorithm for minimum cuts, in: Proceedings of the Twenty Fourth Annual ACM Symposium on the Theory of Computing (1993) 757–765.

[23] N. Katoh and K. Iwano, Efficient algorithms for minimum range cut problems, IBM Research Report, in: Proceedings of the Second Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, 519 (Springer, Berlin, 1991) 80–91.

[24] B.W. Kernighan and S. Lin, An effective heuristic procedure for partitioning graphs, BSTJ 49 (1970) 291–307.

[25] V. King, S. Rao and R.E. Tarjan, A faster deterministic maximum flow algorithm, in: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (1992) 157–164.

[26] D. Klingman, A. Napier and J. Stutz, NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems, Management Sci. 20 (1974) 814–821.

[27] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout (Wiley, West Sussex, 1990).

[28] D.W. Matula, Determining edge connectivity in $O(nm)$, in: Proceedings of 28th Annual Symposium on Foundations of Computer Science (1987) 249–251.

[29] H. Nagamochi and T. Ibaraki, Computing edge-connectivity in multiple and capacitated graphs, SIAM J. Discrete Math. 5 (1992) 54–66.

[30] K. Ohtsuka, Y. Dai, N. Katoh and K. Iwano, A new randomized algorithm for the minimum cut problems and its performance evaluation, IPSJ Technical Report, Algorithms, 92-AL-26-3 (1992).

[31] J.C. Picard and M. Queyranne, Selected applications of minimum cuts in networks, INFOR 20 (1982) 394–422.
[32] C. Savage and J. JáJá, Fast, efficient parallel algorithms for some graph problems, SIAM J. Comput. 10 (1981) 682–691.
[33] Y. Shiloach and U. Vishkin, An O(log n) parallel connectivity algorithm, J. Algorithms 3 (1982) 57–67.
[34] D.D. Sleator and R.E. Tarjan, A data structure for dynamic trees, J. Comput. Sci. 26 (1983) 362–391.
[35] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, IEEE Trans. Software Engrg. 3 (1977) 85–93.