Journal of Algorithms **38,** 51–83 (2001)
doi:10.1006/jagm.2000.1136, available online at http://www.idealibrary.com on IDEAL®

Cut Tree Algorithms: An Experimental Study

Andrew V. Goldberg[1]

STAR Laboratory, InterTrust Technologies Corp.,
4750 Patrick Henry Drive, Santa Clara, California 94054
E-mail: goldberg@intertrust.com

and

Kostas Tsioutsiouliklis[2]

Computer Science Department, Princeton University, Princeton, New Jersey 08540
E-email: kt@cs.princeton.edu

Received June 2, 1999

This is an experimental study of algorithms for the cut tree problem. We study the Gomory–Hu and Gusfield algorithms as well as heuristics aimed to make the former algorithm faster. We develop an efficient implementation of the Gomory–Hu algorithm. We also develop problem families for testing cut tree algorithms. In our tests, the Gomory–Hu algorithm with a right combination of heuristics was significantly more robust than Gusfield's algorithm. © 2001 Academic Press

1. INTRODUCTION

Cut trees, introduced by Gomory and Hu [GH61] and also known as *Gomory–Hu trees*, represent the structure of all *s*-*t* cuts of undirected graphs in a compact way. Cut trees have many applications.

All known algorithms for building cut trees use a minimum *s*-*t* cut subroutine. The most efficient currently known way to find a minimum *s*-*t* cut is using a maximum flow algorithm. See [GR98] for the currently known

[1]Part of this work was done while the first author was at NEC Research Institute, Princeton, NJ 08540.

[2]Supported in part by NSF-DIMACS Graduate Fellowship; grants CCR-8809648 and CCR-919999.

0196-6774/01 $35.00

maximum flow bounds. Gomory and Hu [GH61] showed how to solve the tree problem using $n - 1$ minimum cut computations and graph contractions. (In this paper n and m denote the number of vertices and edges in the input graph, respectively.) As we shall see, an efficient implementation of this algorithm is nontrivial. Gusfield [Gus90] proposed an algorithm that does not use graph contraction; all $n - 1$ minimum s-t cut computations are performed on the input graph. Gusfield's algorithm is very simple and can be implemented by adding a few lines to a maximum flow code.

Computational performance of algorithms for closely related problems, the maximum flow problem, and the (global, e.g., over all s,t pairs) minimum cut problem has been studied extensively; see, e.g., [AS93, CG97, DM89, Gol87, NV93] for computational studies of the former problem and [CGK+97, RT97, Lev97, NOI94, PR90] for the latter. Both problems can be solved well in practice: most instances that fit in RAM of a modern computer can be solved in a few minutes. The cut tree problem appears more difficult, for one needs to solve $n - 1$ minimum s-t cut problems.

Therefore, computational performance of cut tree algorithms is of great interest. Implementations of cut tree algorithms exist—for example, as subroutines of TSP codes [AC97, Gro97]. However, we are not aware of any published computational studies of cut tree algorithms. In this paper we undertake such a study.

We describe how to implement the Gomory–Hu and Gusfield algorithms efficiently. We also introduce and study heuristics aimed at improved computational performance of these algorithms. Our computational experiments lead to a good understanding of practical performance of the cut tree algorithms.

2. DEFINITIONS AND NOTATION

The input to the cut tree problem is an undirected graph $G = (V, E)$ and a capacity function $c : E \Rightarrow \mathbf{R}^+$. We denote $|V| = n$ and $|E| = m$. A *cut* (X, Y) in G is a partitioning of V into two nonempty sets. We say that an edge *crosses* the cut if its two endpoints are on different sides of the cut. *Capacity of a cut* is the sum of capacities of edges crossing the cut.

We distinguish between *vertices* and *nodes*. We refer to the elements of V as vertices. Nodes correspond to subsets of vertices. (A node can be a single-element subset.) We need the distinction because we use contraction operations.

For $s, t \in V$, an s-t cut is a cut such that s and t are on different sides of it. A *minimum s-t cut* is an s-t cut of minimum capacity. A *(global) minimum cut* is a minimum s-t cut over all s, t pairs.

A *cut tree* is a weighted tree T on V with the following property. For every pair of distinct vertices s and t, let e be a minimum weight edge on the unique path from s to t in T. Deleting e from T separates T into two connected components, X and Y. Then (X, Y) is a minimum s-t cut. Note that T is not a subgraph of G, i.e., edges of T do not need to be in E.

3. GOMORY–HU ALGORITHM

In this section we outline the Gomory–Hu algorithm and its efficient implementation. We also discuss heuristics that may improve the algorithm's performance in practice. We provide only the details of the algorithm needed to describe the implementation and the heuristics. For a complete description, see, e.g., [LFF62, GH61, Hu82].

The Gomory–Hu algorithm is recursive. It distinguishes between two kinds of nodes: original and contracted. A vertex of the input graph is an *original node*. If there is more than one original node, the algorithm picks two, s and t, finds a minimum s-t cut (S, T), and forms two graphs, G_s by contracting S into a contracted node, and G_t by contracting T. Then it recursively builds cut trees in G_s and G_t and puts these trees together. One can see that the algorithm maintains the following invariant: a trivial cut around a contracted node is a minimum cut between this node and some other node in the graph. If there is only one original node, the algorithm has enough information to construct a cut tree; in this case the recursion bottoms out.

Because of the contraction operations, efficient implementation of the Gomory–Hu algorithm is nontrivial. (This was the main motivation behind Gusfield's algorithm.) A naive implementation of contractions allocates new memory for a contracted node and its edges. This may result in $\Omega(n^2)$ memory allocation even for a sparse graph. We describe an implementation that uses $O(\log n)$ extra node records and $O(n)$ extra edge records. These records are allocated as a block at the beginning of the computation, avoiding expensive run-time memory allocation and improving locality of reference.

We maintain the following information at every step of the computation. Recall that the computation is recursive. For each recursive call currently in progress, we maintain information about the cut computed at this level and the graphs obtained by contracting one of the cut sides. When the first recursive call returns, we mark node and edge records of the corresponding subgraph as free. We also maintain information about the contracted nodes (on which side of the cut they are each time), since this determines the structure of the final cut tree. Finally, we maintain a data structure that builds the cut tree according to the cuts found so far.

Our implementation first recurses on the subgraph with a smaller number of nodes and uses fewer additional nodes and edges because of this. We analyze these numbers next.

For the second recursive call, we can reuse the nodes and the edges of the already processed subgraph and use no additional storage. The number of extra nodes we need is determined by the longest sequence of left branches in a root-to-leaf path in the recursion tree (corresponding to the first recursive calls), which is $\lceil \log_2 n \rceil$ because we recurse on the smaller subgraph first. Similarly, the number of extra edges needed is determined by the maximum, over all root-to-leaf paths, of the sum of sizes of subproblems corresponding to left branches. The maximum is bounded by n.

Note that our implementation destroys the input graph. If this is not desirable, one can make a copy of the graph before running the algorithm. All our codes are superlinear, and the time to make the copy would be negligible except for small graphs.

When implemented as described above, direct overhead of contraction operations is small; contraction usually costs less than the corresponding minimum cut computation. However, there is also indirect cost: locality of the input graph representation suffers because of the contractions, reducing the number of cache hits somewhat.

At high level, two major factors determine the computational performance of the algorithm. The first one is the balance (e.g., the ratio of the number of nodes) of the cuts found by the algorithm. In the worst case, one side of every such cut contains one node. In the best case, the cuts are balanced. In the latter case, assuming that minimum cut computations are superlinear, the first one dominates the total running time. The second factor is the hardness of the minimum cut subproblems. Heuristics that lead to more balanced cuts or simpler subproblems improve the algorithm performance.

The *balance* heuristic aims at keeping the cuts balanced. Assume we have at least four original nodes. First we compute all minimum cuts between two such nodes, a and b, and take the most balanced cut. If the cut is sufficiently balanced (e.g., the ratio of the number of nodes of the larger and the smaller parts does not exceed a threshold), we proceed. Otherwise, we pick two nodes, c and d, on the bigger side of the cut. We compute all minimum cuts between c and d, take the most balanced one, compare it to the most balanced minimum cut between a and b, and choose the best. We may have to compute twice as many cuts, so the worst-case loss is about a factor of two. The best-case gain can be much larger.

We can also use both cuts, since according to [GH61] if the cuts are crossing, we can always find noncrossing cuts. We use this technique in one of our codes (GHs, see below), where it often leads to a speedup, but the speedup is small: most often one of the cuts is quite unbalanced, and the computation to find noncrossing cuts is relatively expensive. This heuristic does not seem to improve our other codes, and we do not use it in these codes.

The *mincut heuristic* makes use of the Hao–Orlin algorithm [HO94] for finding global mincuts. This algorithm uses the push–relabel method to find a minimum cut between the source and the sink. Then it contracts the source and the sink, and selects a new sink. Hao and Orlin show that with a careful implementation of many push–relabel algorithms, the asymptotic worst-case time bound for these $n - 1$ minimum s-t cut computations is the same as that for one minimum s-t cut computation of the underlying algorithm.

Note that the first cut found by the Hao–Orlin algorithm is a minimum s-t cut in the input graph. Also, the algorithm finds a minimum cut, which is a minimum s-t cut for any s, t on the opposite sides of it. We prove a lemma that allows us to use several cuts found by the algorithm in the cut tree construction.

The Hao–Orlin algorithm has the following property. Let s be the initial source and let S be the set of vertices contracted into the source at some point of an execution of the algorithm. Let λ be the capacity of the smallest cut found up to this point (initially $\lambda = \infty$). Then for any $x \in S$, the capacity of a minimum s-x cut is at least λ.

LEMMA 3.1. *Suppose that t is the next sink and the minimum S-t cut has value $\lambda' \leq \lambda$. Then this cut is also a minimum cut between s and t in G.*

Proof. Suppose that there is a smaller cut between s and t. This cut cannot separate s from a vertex $x \in S$ because s and x are λ-connected. Thus the cut separates S and t. This contradicts the definition of λ'. ∎

The mincut heuristic uses the above lemma and finds several minimum s-t cuts with one Hao–Orlin computation. This number is usually small, so we use this heuristic together with the balance heuristic to obtain one or two cuts—the most balanced ones.

The *source selection heuristic* is aimed at making balanced cuts more likely. This heuristic uses the fact that any original node can be chosen as the source for the next minimum cut computation. After choosing a sink for the computation, we choose an original node that is furthest away from the sink as the source. (All distances are with respect to a unit length function.) Note that we use an implementation of the push–relabel method [GT88] based on that of [CG97]. This implementation computes distances to the sink during the initialization, so the source selection heuristic adds essentially no overhead.

As part of the source selection heuristic, we choose the source–sink to be the heaviest nodes of the graph (e.g., nodes with the highest total capacity of adjacent edges), since this sometimes leads to more balanced cuts.

Padberg–Rinaldi heuristics [PR90] proved very useful for certain classes of global minimum cut problems. One can use these heuristics (in a somewhat restricted form) in the Gomory–Hu algorithm. However, on the problems these heuristics are effective, their use tends to lead to less balanced cuts and worse running times. This is because the heuristics tend to contract together large subsets of nodes. Although we invested substantial effort, we could not use the heuristics to consistently speed up our codes.

3.1. *Our Implementations*

After studying different ways of incorporating heuristics into the Gomory–Hu algorithm, we report on three implementations. The GH code uses no heuristics and picks the next source–sink pair at random. The GHs code uses the source selection heuristic. The GHg code uses the mincut heuristic in the following way: Initially, it picks the two heaviest nodes as the source and the sink of the Hao–Orlin algorithm.[3] As soon as it finds a cut in the decreasing sequence which is more balanced than the first cut found, it splits the graph according to both this cut and the first one. Our experience shows that using the mincut heuristic is the best way to find balanced cuts at low expense. Note that since some minimum-cut computations produce two minimum cuts, the number of minimum-cut computations can be less than $n - 1$.

The GH and GHs implementations run in $O(nS(n, m))$ time, where $S(n, m)$ is the running time of the maximum flow subroutine. The GHg implementation runs in $O(nH(n, m))$, where $H(n, m)$ is the running time of the Hao–Orlin subroutine.

4. GUSFIELD'S ALGORITHM

Like the Gomory–Hu algorithm, Gusfield's algorithm [Gus90] consists of $n - 1$ iterations of a minimum cut subroutine and bookkeeping that puts the resulting cuts together. Gusfield's algorithm, however, does not contract vertices and works with the original graph, making it easy to implement. At each of the $n - 1$ iterations of Gusfield's algorithm, a different vertex is chosen as the source. This choice determines the sink. Since Gusfield's algorithm always works with the original graph, source selection can affect only the difficulty of the subproblems to be solved. We do not know if one can have a selection strategy that leads to simpler subproblems. We choose the next source at random. We refer to the resulting implementation as Gus. The implementation runs in $O(nS(n, m))$ time.

[3]A random choice was much less robust in our tests.

TABLE 1
Problem families reported on in this paper. We experimented with more families, but do not report on some where the results were similar to the ones we include.

Problem family	# nodes	# edges	Other parameters
BIKEWHE	32,64,...,1024	$2n - 3$	
CYC1	64,...,4096	n	
DBLCYC	64,...,1024	$2n$	
IRREG	1000	4500-5000	$k = 9, W \in [0 \ldots 1000]$
NOI1	100-800	density: 50%	$P = n, k = 1$
NOI2	100-800	density: 50%	$P = n, k = 2$
NOI3	500	6000-124000	$P = 1000, k = 1$
NOI4	500	6000-124000	$P = 1000, k = 2$
NOI5	500	62000	$P = 1000$
			$k = 1, 2, 3, 5, ..., 300, 500$
NOI6	500	62000	$P = 5000, 2000, ..., 10, 1$
			$k = 2$
PATH	2000	20000	$P = 1,000$
			$k \in [1 \ldots 2000]$
PR1	200,400,...,1000	density: 2%	$k = 1$
PR5	200,400,...,1000	density: 2%	$k = 2$
PR6	200,400,...,1000	density: 10%	$k = 2$
PR7	200,400,...,600	density: 50%	$k = 2$
PR8	200,400,...,600	density: 100%	$k = 2$
REG1	301	301,...,90300	
REG2	50,100,...,800	$50n$	
TREE	800	density: 50%	$k \in [1 \ldots 800]$
TSP	500-13000	$\approx n$	
WHE	64,128,...,1024	$2n - 2$	

Low-level operations of this algorithm are efficient because of its simplicity and the fact that the algorithm takes advantage of locality of the input graph representation.

5. EXPERIMENTAL SETUP

For our experiments, we used a 300 MHz SUN Ultra-10 workstation with 256 Mbyte memory running SOLARIS-7. All the code is written in C and compiled with gcc and optimization option -O4. Our implementations are written in the same style and are derived from the Hao–Orlin algorithm implementation of [CGK+97]. We attempted to make all implementations as efficient as possible.

For our tests we use problem families from the previous minimum cut studies [CGK+97, Lev97, NOI94, PR90], but instead of finding a minimum cut of a graph, we build a cut tree. We omit the description of the problem families. Detailed descriptions appear in [Lev97]. We do not report on PR2–PR4 problem families because the results are very close to those for the PR1 family, and on REG3–REG4 families because the results are very close to those for the REG1 and REG2 families. We also use two new problem families produced by two generators, PATHGEN and TREEGEN, described below. A summary of the problem families we use appears in Table 1.

The PATHGEN generator works as follows. Given a parameter k, it builds a path of $k-1$ "heavy" edges and connects the remaining $n-k$ vertices to the path vertices by heavy edges, at random. Then it adds "light" edges at random to achieve the desired number of edges and to make the minimum cut problems more difficult. This generator takes the following parameters:

- n, the number of vertices;
- d, the density of the graph as a percentage;
- k, the path length;
- P, the path arc capacity parameter;
- S, the seed.

Heavy edge capacities are chosen uniformly at random from the interval $[1, \ldots, 100 \cdot P]$ and light edge capacities from $[1, \ldots, 100]$.

The value of k determines the path shape. For example, if $k = n$ then we get one heavy path through all the nodes; if $k = 1$, then the graph is a star. We use PATHGEN to produce the PATH problem family. We use $n = 2000$, $d = 10$, $P = 1000$, and k changing from 1 to 2000.

The TREEGEN generator works as follows. Given a parameter k, it builds a tree by connecting vertex i, $2 \leq i \leq n$, to a randomly chosen vertex in $[1, \min(i-1, k)]$. The tree edges are heavy. Then it adds "light" edges at random to achieve the desired number of edges and to make the minimum cut problems more difficult. This generator takes the following parameters:

- n, the number of vertices;
- d, the density of the graph as a percentage;
- k, the shape parameter mentioned above;
- P, the path arc capacity parameter;
- S, the seed.

The generator chooses heavy edge capacities uniformly at random from the interval $[1, \ldots, 100 \cdot P]$ and light edge capacities from $[1, \ldots, 100]$.

The value of k determines the shape of the tree. For example, if $k = 1$ then the tree is a star. If $k = n - 1$, then a tree is obtained by connecting each vertex except the first one to a randomly chosen preceding vertex.

We use TREEGEN to produce the TREE problem family.

TABLE 2
Summary of algorithm performance. $\bigcirc$ means good, $\odot$ means fair, $\otimes$ means poor, and $\bullet$ means bad. + marks the fastest code(s).

Problem family	GUS	GH	GHS	GHG
BIKEWHE	$\bigcirc$	$\bigcirc$	$\odot$	$\bigcirc$+
CYC1	$\bigcirc$+	$\bigcirc$	$\bigcirc$	$\bigcirc$
DBLCYC	$\bullet$	$\otimes$	$\otimes$	$\bigcirc$+
IRREG1	$\bigcirc$	$\bigcirc$	$\bigcirc$	$\bigcirc$
NOI1	$\bigcirc$+	$\bigcirc$	$\bigcirc$	$\bigcirc$
NOI2	$\odot$	$\bigcirc$+	$\bigcirc$	$\bigcirc$
NOI3	$\bigcirc$+	$\bigcirc$	$\bigcirc$	$\bigcirc$
NOI4	$\odot$	$\bigcirc$	$\bigcirc$	$\bigcirc$
NOI5	$\bullet$	$\odot$	$\bigcirc$+	$\bigcirc$
NOI6	$\bigcirc$	$\bigcirc$	$\bigcirc$	$\bigcirc$
PATH	$\bullet$	$\bullet$	$\bigcirc$	$\bigcirc$
PR1	$\bigcirc$+	$\bigcirc$	$\bigcirc$	$\bigcirc$
PR5	$\bigcirc$	$\bigcirc$	$\bigcirc$+	$\bigcirc$
PR6	$\odot$	$\bigcirc$	$\bigcirc$	$\bigcirc$
PR7	$\odot$	$\bigcirc$+	$\bigcirc$	$\bigcirc$
PR8	$\odot$	$\bigcirc$+	$\bigcirc$	$\bigcirc$
REG1	$\bigcirc$	$\bigcirc$	$\bigcirc$	$\bigcirc$
REG2	$\bigcirc$	$\bigcirc$	$\bigcirc$	$\bigcirc$
TREE	$\otimes$	$\bullet$	$\bigcirc$	$\odot$
TSP	$\otimes$	$\bigcirc$	$\bigcirc$	$\bigcirc$+
WHE	$\odot$	$\odot$	$\odot$	$\bigcirc$+

6. EXPERIMENTAL RESULTS

In this section we describe our experimental results. Table 2 summarizes these results. Detailed data appear in the Appendix. Our experimental results should be taken in the context of our study.

We use the following scoring system in the table. For each data point, we normalize the times by that of the fastest code and use a factor of two as the threshold between adjacent scores. For example, if the fastest code runs in x seconds, a code running in $1.5x$ s is rated good, in $3x$ s fair, in $7x$ s poor, and in $12x$ s bad. Then for each problem family, we assign each code the worst rating over all data points in this problem family. Our choice of the threshold makes it less likely that a code not rated good in our experiment would be the fastest under a different compiler and machine architecture combination. If a code is consistently faster than the other codes, we mark that code with a +. Several codes can be marked if their performance is very close, and no codes can be marked if there is no consistent winner. Note that no code will get a good score on a problem family if every code performs relatively poorly for some parameter values.

This scoring system gives a general idea of relative performance of the codes and is fairly independent of many low-level implementation details and machine architecture variations. Note that in some cases larger problem sizes may amplify performance differences and thus change the scores.

Data tables in the appendix give much more information than the above scores and can be used to explain performance differences. All our implementations are based on the push–relabel maximum flow method; we give counts of the relabel operations. This machine-independent count is usually a good measure of performance of the push–relabel algorithm we use [CG97]. One exception is very simple problems, where the number of operations is much less than the number of vertices; in such cases the algorithm is dominated by its linear-time initialization. We also give the average size (the number of nodes and the number of edges) of the s-t cut problems solved, or, for GHG, the average size of the problems to which we apply the Hao–Orlin subroutine. The average problem size is correlated with the algorithm performance. In addition to the total running time, we give the time spent computing minimum s-t cuts (CutTime) and the time spent on auxiliary operations (ManipTime) such as building the cut tree and contracting nodes. The total time is equal to the sum of the CutTime, ManipTime, initialization time, and postprocessing time. (We exclude IO time.) Note that relabel operation counts are correlated with CutTime but not with ManipTime. ManipTime is independent of how difficult the minimum cut subproblems are. For most problem families, ManipTime does not exceed CutTime by more than a factor of four, and the operation counts give a measure of performance to within an order of magnitude. For PATH and TREE families, in some cases ManipTime is much greater than CutTime because the subproblems are very simple. See Figs. 11 and 19.

We are especially interested in robust codes. Given a collection of codes and a collection of test problems, we say that a code is *robust* (with respect to the two collections) if for no test instance the code is significantly slower than the best code in the collection. This definition depends on the selection of codes and test instances. Our conclusions about robustness should be taken in the context of our study. However, our experiments include a wide range of problems, and the codes which are robust in our tests may be robust over an even wider range of problem instances.

Furthermore, the following argument suggests that a good implementation of the Gomory–Hu algorithm will be more robust than that of Gusfield's algorithm. Recall that both algorithm solve $n - 1$ minimum s-t cut problems. However, while the former always works with the input graph, the latter works with contractions of the input graph. Thus one would expect that when the average problem size in an execution of the Gomory–Hu algorithm is small, the algorithm is significantly faster than Gusfield's algorithm. On the other hand, if one assumes that hardness of the s-t cut subproblems solved by the algorithms depends mostly on the problem size, then one would expect that a good implementation of the Gomory–Hu algorithm is never significantly slower than an implementation of Gusfield's algorithm. Our experimental results confirm these expectations.

6.1. *Gusfield's Algorithm*

The data show that GUS is not robust. Although it is the fastest code on many problem families, in some cases it performs much worse than the Gomory–Hu algorithm.

Operation counts show that Gusfield's algorithm wins mostly due to its simplicity and better spatial locality resulting from the lack of contraction operations. For all our codes, edges of the input graph are stored in an array with edges adjacent to a vertex stored in adjacent array locations. This results in improved spatial locality when scanning adjacency lists of the original vertices. Contraction operations merge adjacency lists together and reduce locality.

For example, consider the NOI3 family (Fig. 7). On this family, the Gomory–Hu codes fail to reduce the average number of nodes. The numbers of relabel operations for GH and GHS are roughly the same as for GUS, but the running times are roughly 33 to 66% higher. Since GHG solves a harder problem at each iteration, its number of relabel operations is higher than for the other codes, and the running time is slower, although both are within a factor of two of those for GUS.

Similar behavior occurs on other problem families where GUS wins, such as CYC1, NOI1, PR1, and REG1. Gomory–Hu codes find very unbalanced cuts, never perform significantly fewer operations than GUS, and run slower. In particular, the numbers of relabel operations for GUS and GHS are always very close and performance difference is around 40%, which is consistent with our reduced locality theory.

The NOI5 family shows how the Gomory–Hu algorithm's ability to find balanced cuts affects performance. Graphs in this family have the following structure. Each graph is a random graph with 500 vertices and 62,375 edges. Each vertex is colored into one of k colors, selected independently at random. We pick edge capacities uniformly at random, from the range $[1, 100, 000]$ if the endpoints are of the same color and from the range $[1, 100]$ otherwise. For $k = 1$ and $k = 500$, the graph is a random graph with uniform random weights. Such graphs tend to have very unbalanced minimum cuts and GUS outperforms the Gomory–Hu codes (by a relatively small margin). See Fig. 9. For moderate values of k (between 3 and 100), large subgraphs have balanced minimum cuts and the Gomory–Hu codes work with subproblems that are small on the average: in particular, the average number of edges is smaller by over an order of magnitude.

For large k, the cuts become unbalanced again. The data show that GUS performance does not change much as k changes. Both the number of relabel operations and the running time vary by at most a factor of two. The Gomory–Hu codes, on the other hand, perform much better for moderate values of k.

The DBLCYC family gives another example of the effect of the balanced cuts. This family illustrates that the speedup due to size reduction can exceed the size reduction factor. In particular, for all but the smallest problem size tested, the average subproblem size in GUS exceeds that in GH by at most a factor of two, yet the speedup is often much greater. This is not surprising since the time to find a minimum s-t cut can be superlinear in the problem size.

Next we discuss how our least robust Gomory–Hu code, GH, compares with GUS. Table 2 shows that on the TREE family, GUS gets a higher score. Figure 19 shows, however, that GH running times are always within a factor of 2.5 or less of those for GUS. On the other hand, for $k = 7$ in the NOI5 family, GUS is slower by a factor of 15 (Fig. 9).

Finally we compare GUS to our most robust code, GHG. Although on some problem classes the former code is faster, the difference is always less than a factor of two. On the other hand, GHG can be much faster than GUS: in particular, it is about 45 times faster on a 1,024-vertex DBLCYC problem.

Our data support the expectation that Gusfield's algorithm is less robust than the Gomory–Hu algorithm because the former cannot take advantage of balanced minimum cuts while the latter can.

6.2. *The Gomory–Hu Implementations*

In this section we discuss performance of our Gomory–Hu codes. The performance depends on two factors: how balanced the "typical" cuts are and how much work is involved in looking for more balanced cuts.

The NOI6 family illustrates these phenomena. Graphs in this family are random, and the nodes are randomly partitioned into two color classes. Edge weights are chosen independently and uniformly, from the range $[1, 100]$ if the edge endpoints are in a different color class and from the range $[1, 100 * P]$ otherwise. This problem family is parameterized by P. Figure 10 shows a threshold phenomenon, with running times of the Gomory–Hu codes dropping by about a factor of four as P changes from 150 to 250. This is what one would expect. For $T = 1$, a NOI6 graph is a random graph with uniform weights. Minimum s-t cuts in such a graph are trivial, the Gomory–Hu codes do not find nontrivial cuts, and the number of nodes in the subproblems solved is equal to the original number of nodes. For $T = 200$, these codes start to find some nontrivial cuts, and by

$T = 250$ the average number of nodes is reduced by almost a factor of two and the running time by about a factor of four.

Next we compare GH and GHs. Recall that the GH implementation chooses the next source–sink pair at random. This is a natural selection strategy to try. Somewhat surprisingly, in our tests this strategy was not as robust as the source selection heuristic used in GHs, which chooses the node furthest away from the current sink as the source. On most problem families, GH performs similarly to GHs, but on a small number of families (in particular PATH and TREE), the former code is noticeably slower.

The NOI2 family (Fig. 6) gives a typical picture of the relative performance of GH and GHs. Running times, average problem sizes, and operation counts are similar for the two codes. While GH is slightly faster than GHs, the former code executes more relabel operations.

On the PATH family (Fig. 11), choosing a source and a sink far from each other in the path leads to more balanced cuts. For $K = 1$, the average size for both codes is the same, 2,000 nodes. After that, the average size for GHs is smaller than that for GH. For GHs, the size drops down sharply, dropping below 150 nodes for $K = 15$, reaching the minimum for $K = 200$, and then growing slowly while staying below 150. For GH, the size decreases slowly at first, reaching 15,504 nodes for $K = 50$, then drops sharply to 3,667 at $K = 200$, and stays below that number as K grows. For moderate values of K, GHs finds cuts that are significantly more balanced than those found by GH.

Finally we compare GHs and GHG, the most robust codes in our study. Receiving only one fair mark, GHG is the most robust code. On some input classes (BIKEWHE, DBLCYC, WHE), it outperforms the other codes by a wide margin. This is due to the fact that on these problem classes, GHG finds more balanced cuts and on the average works with smaller problems. One has to keep in mind, however, that the structure of these graph instances is quite special, and one has to be careful not to overestimate GHG's performance on "typical" problems. Often, the number of Hao–Orlin computations performed by GHG is close to $n/2$ because the code finds two minimum cuts in most iterations. However, the average problem size for GHG is close to that for GHs, and the running time is somewhat higher because a Hao–Orlin computation is more expensive than a minimum s-t cut computation. Thus on many graphs GHG is slightly slower than GHs.

The TREE family is the only problem family where GHG gets a fair score. On this problem family the average problem size for GHG is a little larger than that for GHs, and the latter code is faster because a minimum s-t cut computation is more efficient than a Hao–Orlin computation.

Finally, we look at the data for TSP problems, the only problems in our study that come from a real application. See Fig. 20. On these problems the Hao–Orlin algorithm outperforms Gusfield's, with the difference especially noticeable on the `rl5934` problems. Out of the Hao–Orlin codes, GH and GHs perform similarly and GHg usually performs a little better.

7. CONCLUDING REMARKS

In this section we summarize our work and discuss heuristics that work as well as those that do not work.

Currently, the cut tree problems are substantially harder than the related maximum flow and minimum cut problems, both in theory and in practice. This is a good motivation for improving theoretical bounds for the problem and developing faster codes for it. Our study is a step toward the faster codes.

We get a good understanding of implementation issues for the existing cut tree algorithms, as well as a good understanding of computational performance of these algorithms. In particular, we show that with a careful low-level implementation, the Gomory–Hu algorithm is more robust than Gusfield's algorithm. This is because all subproblems solved by Gusfield's algorithm have the same size as the input problem. For the Gomory–Hu algorithm, however, the average problem size can be much smaller than the original problem size. The Gomory–Hu algorithm performance is less predictable, because the average problem size depends on the heuristics used.

Good heuristics reduce the average problem size and can substantially improve performance of the Gomory–Hu algorithm on some problems. One such heuristic is our source selection heuristic. Random selection, although quite natural and easy to implement, does not work as well.

Other heuristics are based on the idea of finding several minimum cuts at every iteration of the Gomory–Hu algorithm and selecting the most balanced one. We experimented with the simple balance heuristic of selecting the best of two cuts at every recursive call of the algorithm. The resulting implementation was usually slower than GH, although not by much, and never significantly faster. This is because the best of the two cuts is usually not much more balanced than the first cut. The Hao–Orlin algorithm provides more opportunities for finding balanced cuts, and the GHg implementation was the most robust implementation in our tests. Further research may lead to even more effective heuristics, but we could not produce a more robust code.

Further study of heuristics for the Gomory–Hu algorithm may provide significant improvements.

Finally, we have set up a Web page, supplementary to this paper, which contains all the data and source code associated with our work. The address is: `http://www.cs.princeton.edu/~kt/cut-tree/`.

APPENDIX: DATA TABLES AND PLOTS

In the following tables and plots we present data for all the class instances we have considered in this study. We give the plots only when they make the corresponding tables easier to read. The plots are normal or logarithmic scale, as marked.

Notes:

- All the data reported have been averaged over multiple (5) runs of the algorithms for each input instance.
- N and M denote the number of vertices and edges, respectively.
- #HO is the number of Hao–Orlin computations in a run of GHG. Note that while other algorithms perform exactly $N - 1$ minimum-cut computations, GHG finds one or two minimum cut computations for each invocation of the Hao–Orlin subroutine; thus #HO can be less than $N - 1$.
- *Aver.N* and *Aver.M* are equal to the average size (vertices and edges, respectively) over all subgraphs during the min-cut computations, except for GHG where they are for the Hao–Orlin computations. For GUS these values are equal to N and M. For GH, GHS, and GHG they are often significantly smaller.
- IO time is not included.
- *CutTime* corresponds to the total running time required to find all the cuts; this time is dominated by the max-flow computations.
- *ManipTime* is the time required to manipulate the cuts. This includes the time to build the tree and for GH, GHS, and GHG also includes the time for the contractions done. Moreover, for GHG it includes the time to perform doublesplitting, according to the two most balanced cuts found so far.
- *Relabels* account for the total number of relabels during the min-cut computations.
- *TotalTime* is the total CPU time for each algorithm. *TotalTime* should be slightly bigger than *CutTime* + *ManipTime* (*TotalTime* also includes initialization and postprocessing).
- Some tables contain individual parameters for certain problem families.

BIKEWHE	N	M	Aver.N	Aver.M	Relabels	CutTime	ManipTime	TotalTime
gus	32	61	32.000	61.000	2590	0.008	0.006	0.014
gh	32	61	32.000	79.000	2173	0.008	0.008	0.016
ghs	32	61	32.000	77.000	3297	0.016	0.004	0.022
ghg	32	61	14.097	36.113	1412	0.006	0.006	0.014
#HO	31							
gus	64	125	64.000	125.000	18428	0.054	0.014	0.076
gh	64	125	64.000	159.000	16846	0.060	0.012	0.074
ghs	64	125	64.000	157.000	23681	0.078	0.014	0.096
ghg	64	125	27.603	74.738	10374	0.034	0.008	0.048
#HO	63							
gus	128	253	128.000	253.000	116305	0.414	0.056	0.476
gh	128	253	128.000	319.000	117001	0.404	0.072	0.480
ghs	128	253	128.000	317.000	160638	0.596	0.062	0.662
ghg	128	253	52.504	144.008	60367	0.210	0.056	0.270
#HO	127							
gus	256	509	256.000	509.000	736057	2.968	0.262	3.246
gh	256	509	256.000	639.000	771031	3.094	0.298	3.408
ghs	256	509	256.000	637.000	1107079	4.364	0.304	4.690
ghg	256	509	98.929	275.508	357203	1.604	0.178	1.788
#HO	255							
gus	512	1021	512.000	1021.000	4597115	20.266	1.664	21.960
gh	512	1021	512.000	1279.000	4778584	21.706	2.008	23.736
ghs	512	1021	512.000	1277.000	7713890	34.170	2.068	36.258
ghg	512	1021	198.252	556.555	2220836	10.874	1.302	12.202
#HO	511							
gus	1024	2045	1024.000	2045.000	30058422	142.122	9.520	151.714
gh	1024	2045	1024.000	2559.000	30431074	165.938	12.598	178.596
ghs	1024	2045	1024.000	2557.000	54754257	300.210	12.492	312.768
ghg	1024	2045	396.855	1113.205	15305374	83.208	8.358	91.608
#HO	1023							

FIG. 1. Running times for BIKEWHE family.

CYC1	N	M	Aver.N	Aver.M	Relabels	CutTime	ManipTime	TotalTime
gus	64	64	64.000	64.000	3782	0.008	0.016	0.024
gh	64	64	64.000	98.000	3781	0.010	0.010	0.026
ghs	64	64	64.000	96.000	3782	0.012	0.016	0.028
ghg	64	64	64.000	96.000	3782	0.014	0.014	0.028
#HO	63							
gus	128	128	128.000	128.000	15750	0.060	0.032	0.100
gh	128	128	128.000	194.000	15749	0.054	0.046	0.104
ghs	128	128	128.000	192.000	15750	0.034	0.048	0.086
ghg	128	128	128.000	192.000	15750	0.070	0.036	0.110
#HO	127							
gus	256	256	256.000	256.000	64262	0.174	0.148	0.338
gh	256	256	256.000	386.000	64263	0.150	0.202	0.368
ghs	256	256	256.000	384.000	64262	0.178	0.208	0.400
ghg	256	256	256.000	384.000	64262	0.252	0.212	0.480
#HO	255							
gus	512	512	512.000	512.000	259590	0.630	1.256	1.936
gh	512	512	512.000	770.000	259588	0.690	1.734	2.442
ghs	512	512	512.000	768.000	259590	0.656	1.666	2.346
ghg	512	512	512.000	768.000	259590	1.050	1.538	2.618
#HO	511							
gus	1024	1024	1024.000	1024.000	1043462	2.332	8.922	11.316
gh	1024	1024	1024.000	1538.000	1043462	2.922	10.916	13.884
ghs	1024	1024	1024.000	1536.000	1043462	2.974	10.434	13.438
ghg	1024	1024	1024.000	1536.000	1043462	4.924	9.994	14.980
#HO	1023							
gus	2048	2048	2048.000	2048.000	4184070	10.098	36.498	46.776
gh	2048	2048	2048.000	3074.000	4184071	12.242	45.612	57.956
ghs	2048	2048	2048.000	3072.000	4184070	12.474	43.338	55.946
ghg	2048	2048	2048.000	3072.000	4184070	20.188	42.934	63.196
#HO	2047							
gus	4096	4096	4096.000	4096.000	16756742	49.002	153.092	202.342
gh	4096	4096	4096.000	6146.000	16756740	67.768	210.736	278.702
ghs	4096	4096	4096.000	6144.000	16756742	69.980	205.226	275.426
ghg	4096	4096	4096.000	6144.000	16756742	103.438	194.652	298.294
#HO	4095							

FIG. 2. Running times for CYC1 family.

DBLCYC	N	M	Aver.N	Aver.M	Relabels	CutTime	ManipTime	TotalTime
gus	64	128	64.000	128.000	8904	0.030	0.012	0.048
gh	64	128	64.000	162.000	9940	0.038	0.014	0.054
ghs	64	128	64.000	160.000	19359	0.064	0.010	0.074
ghg	64	128	22.571	55.540	4593	0.016	0.014	0.032
#HO	63							
gus	128	256	128.000	256.000	158022	0.482	0.062	0.552
gh	128	256	65.118	164.843	36321	0.076	0.054	0.136
ghs	128	256	46.874	116.925	30128	0.072	0.040	0.116
ghg	128	256	16.168	39.028	5584	0.028	0.026	0.064
#HO	125							
gus	256	512	256.000	512.000	824857	2.758	0.204	2.972
gh	256	512	128.843	324.120	186897	0.514	0.196	0.716
ghs	256	512	93.463	233.465	211112	0.610	0.172	0.792
ghg	256	512	24.455	59.931	21506	0.096	0.140	0.236
#HO	253							
gus	512	1024	512.000	1024.000	9343902	31.642	1.662	33.340
gh	512	1024	257.411	645.535	1123988	3.320	1.382	4.726
ghs	512	1024	226.335	565.772	1787452	5.484	1.352	6.868
ghg	512	1024	43.299	107.011	85484	0.364	0.884	1.286
#HO	509							
gus	1024	2048	1024.000	2048.000	36782759	129.024	9.634	138.734
gh	1024	2048	513.879	1286.702	7506538	25.302	9.988	35.356
ghs	1024	2048	399.940	999.699	11568074	43.178	9.150	52.396
ghg	1024	2048	91.925	228.608	407532	1.646	6.890	8.584
#HO	1021							

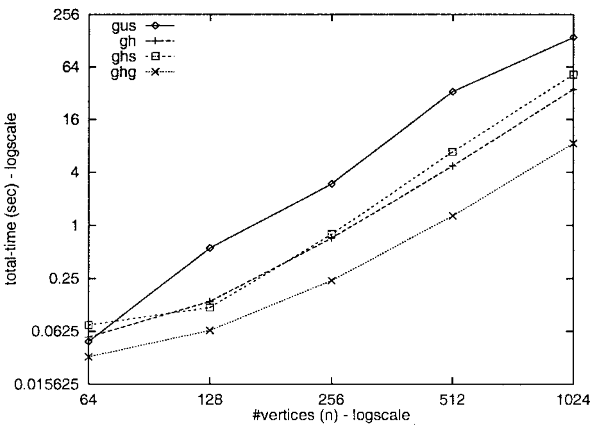

FIG. 3. Running times for DBLCYC family.

IRREG1									
	N	M	Aver.N	Aver.M	K	Relabels	CTime	MTime	TotTime
gus	1000	4500	1000.000	4500.000	0	939855	4.426	11.538	16.024
gh	1000	4500	1000.000	4984.400	0	971814	5.200	15.158	20.410
ghs	1000	4500	1000.000	4982.400	0	963312	5.788	14.368	20.214
ghg	1000	4500	290.212	2037.711	0	533195	4.506	9.522	14.086
#HO	999.0								
gus	1000	4502	1000.000	4502.000	4	861165	5.454	11.554	17.076
gh	1000	4502	1000.000	4986.400	4	973293	5.448	14.816	20.324
ghs	1000	4502	1000.000	4984.400	4	891036	5.382	14.504	19.942
ghg	1000	4502	287.231	1991.661	4	525460	4.452	9.446	13.942
#HO	996.0								
gus	1000	4508	1000.000	4508.000	16	904534	4.252	11.562	15.908
gh	1000	4508	1000.000	4992.400	16	982946	5.328	15.110	20.494
ghs	1000	4508	1000.000	4990.400	16	904950	5.594	14.390	20.036
ghg	1000	4508	303.170	2081.578	16	561800	4.590	9.590	14.232
#HO	984.0								
gus	1000	4532	1000.000	4532.000	64	907691	4.276	11.546	15.916
gh	1000	4532	1000.000	5016.200	64	1007777	5.662	14.970	20.694
ghs	1000	4532	1000.000	5014.200	64	896267	5.622	14.512	20.200
ghg	1000	4532	335.035	2215.969	64	622135	4.864	9.570	14.480
#HO	954.4								
gus	1000	4564	1000.000	4564.000	128	891371	4.260	11.658	15.990
gh	1000	4564	1000.000	5048.200	128	1016060	5.620	15.180	20.848
ghs	1000	4564	1000.000	5046.200	128	912639	5.902	14.658	20.600
ghg	1000	4564	369.576	2346.377	128	680013	5.042	9.690	14.778
#HO	922.8								
gus	1000	4628	1000.000	4628.000	256	888729	4.298	11.640	16.044
gh	1000	4628	1000.000	5111.800	256	1027584	5.726	15.274	21.040
ghs	1000	4628	1000.000	5109.800	256	895799	6.236	15.060	21.344
ghg	1000	4628	348.266	2239.891	256	651540	4.958	9.580	14.582
#HO	938.2								
gus	1000	4756	1000.000	4756.000	512	853441	4.138	11.840	16.058
gh	1000	4756	1000.000	5238.800	512	1020769	5.974	15.688	21.706
ghs	1000	4756	1000.000	5236.800	512	914240	6.738	15.648	22.442
ghg	1000	4756	337.123	2200.561	512	631418	4.888	9.578	14.500
#HO	946.2								
gus	1000	4884	1000.000	4884.000	768	906560	4.362	11.882	16.322
gh	1000	4884	1000.000	5366.200	768	998317	5.974	16.132	22.144
ghs	1000	4884	1000.000	5364.200	768	902697	6.676	16.100	22.820
ghg	1000	4884	325.292	2337.340	768	600495	5.070	9.982	15.102
#HO	965.2								
gus	1000	4948	1000.000	4948.000	896	917560	4.528	12.006	16.604
gh	1000	4948	1000.000	5429.600	896	982031	5.946	16.430	22.438
ghs	1000	4948	1000.000	5427.600	896	909856	6.720	16.156	22.938
ghg	1000	4948	308.378	2316.998	896	571518	5.062	9.920	15.024
#HO	979.6								
gus	1000	5000	1000.000	5000.000	1000	954540	4.712	11.970	16.776
gh	1000	5000	1000.000	5481.400	1000	966769	5.936	16.760	22.740
ghs	1000	5000	1000.000	5479.400	1000	937960	6.896	16.100	23.042
ghg	1000	5000	293.303	2252.571	1000	544863	4.982	9.850	14.886
#HO	999.0								

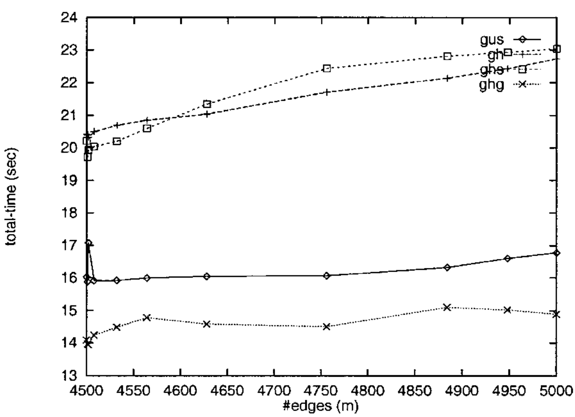

FIG. 4. Running times for IRREG1 family ($K = 9$).

NOI1								
	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	100	2475	100.000	2475.000	8072	0.104	0.160	0.268
gh	100	2475	100.000	2001.800	10210	0.146	0.188	0.340
ghs	100	2475	100.000	1999.800	7467	0.144	0.176	0.328
ghg	100	2475	100.000	1999.800	15586	0.264	0.132	0.398
#HO	50							
gus	200	9950	200.000	9950.000	32507	1.074	1.778	2.870
gh	200	9950	200.000	7934.800	40733	1.400	2.566	3.976
ghs	200	9950	200.000	7932.800	30857	1.486	2.590	4.086
ghg	200	9950	200.000	7932.800	63019	2.570	1.962	4.534
#HO	100							
gus	300	22425	300.000	22425.000	76437	4.300	7.064	11.380
gh	300	22425	300.000	17784.200	91305	4.982	11.034	16.024
ghs	300	22425	300.000	17782.200	73696	6.336	10.800	17.152
ghg	300	22425	300.000	17782.200	146903	11.098	8.660	19.770
#HO	150							
gus	400	39900	400.000	39900.000	137977	10.984	16.548	27.552
gh	400	39900	400.000	31615.400	161840	12.432	26.732	39.182
ghs	400	39900	400.000	31613.400	134418	15.944	25.864	41.836
ghg	400	39900	400.000	31613.400	245538	25.932	20.964	46.908
#HO	200							
gus	500	62375	500.000	62375.000	220971	22.872	31.966	54.876
gh	500	62375	500.000	49346.798	252343	25.296	52.608	77.924
ghs	500	62375	500.000	49344.802	215845	32.290	50.726	83.044
ghg	500	62375	500.000	49344.800	380652	52.146	40.948	93.108
#HO	250							
gus	600	89850	600.000	89850.000	322305	41.338	55.262	96.650
gh	600	89850	600.000	70937.800	362894	45.098	91.016	136.146
ghs	600	89850	600.000	70935.800	316684	57.624	87.770	145.410
ghg	600	89850	600.000	70935.800	553239	94.168	71.246	165.432
#HO	300							
gus	700	122325	700.000	122325.000	440571	67.810	87.848	155.718
gh	700	122325	700.000	96604.798	493805	73.208	145.326	218.562
ghs	700	122325	700.000	96602.799	427978	92.548	139.968	232.552
ghg	700	122325	700.000	96602.798	754564	151.842	114.130	265.982
#HO	350							
gus	800	159800	800.000	159800.000	572226	102.916	131.600	234.584
gh	800	159800	800.000	126152.799	646932	112.004	218.238	330.288
ghs	800	159800	800.000	126150.798	560812	139.660	209.908	349.590
ghg	800	159800	800.000	126150.800	1028239	241.606	171.500	413.130
#HO	400							

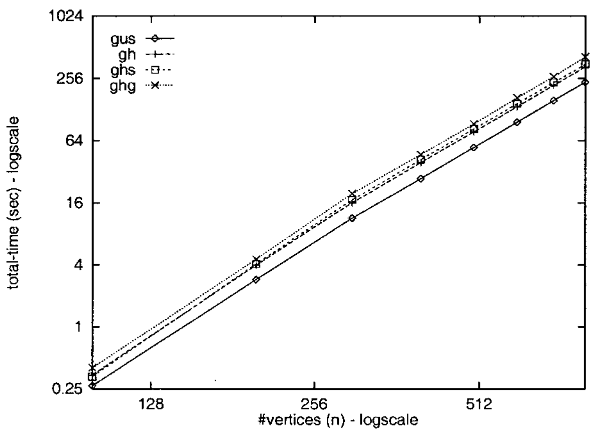

FIG. 5. Running times for NOI1 family.

NOI2	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	100	2475	100.000	2475.000	9405	0.142	0.126	0.280
gh	100	2475	51.877	576.979	5716	0.048	0.072	0.128
ghs	100	2475	52.632	606.605	3849	0.044	0.082	0.128
ghg	100	2475	54.109	650.697	8296	0.098	0.056	0.154
#HO	50							
gus	200	9950	200.000	9950.000	37259	1.188	1.796	3.006
gh	200	9950	101.851	2129.315	21855	0.378	0.746	1.132
ghs	200	9950	104.755	2319.641	16095	0.408	0.830	1.244
ghg	200	9950	104.764	2319.622	34968	0.778	0.550	1.334
#HO	100							
gus	300	22425	300.000	22425.000	83617	4.710	7.130	11.866
gh	300	22425	151.963	4661.092	47928	1.302	3.138	4.450
ghs	300	22425	157.648	5204.805	38543	1.774	3.204	4.990
ghg	300	22425	156.200	5069.948	78217	3.102	2.548	5.672
#HO	150							
gus	400	39900	400.000	39900.000	150555	12.098	16.576	28.710
gh	400	39900	201.916	8185.691	84492	3.342	8.286	11.644
ghs	400	39900	202.461	8288.572	66213	4.220	7.886	12.126
ghg	400	39900	203.600	8424.673	136564	7.948	6.672	14.634
#HO	200							
gus	500	62375	500.000	62375.000	237081	24.498	32.022	56.558
gh	500	62375	252.429	12785.183	130191	6.750	17.114	23.880
ghs	500	62375	258.742	13782.186	108203	9.532	17.306	26.862
ghg	500	62375	258.939	13810.678	216870	17.532	14.256	31.788
#HO	250							
gus	600	89850	600.000	89850.000	340552	43.788	55.380	99.212
gh	600	89850	302.181	18244.816	186689	12.000	29.950	41.982
ghs	600	89850	307.650	19283.536	156111	16.572	29.428	46.020
ghg	600	89850	313.466	20326.974	317277	32.174	25.562	57.744
#HO	300							
gus	700	122325	700.000	122325.000	459326	71.594	88.092	159.740
gh	700	122325	352.168	24717.824	252436	19.490	47.348	66.884
ghs	700	122325	353.985	25164.450	208410	25.874	45.468	71.382
ghg	700	122325	355.617	25505.119	421099	48.946	38.556	87.526
#HO	350							
gus	800	159800	800.000	159800.000	608897	110.190	132.296	242.536
gh	800	159800	402.020	32160.971	329015	29.468	70.306	99.802
ghs	800	159800	402.206	32281.823	277282	38.460	66.712	105.204
ghg	800	159800	403.286	32537.609	536687	73.142	57.308	130.474
#HO	400							

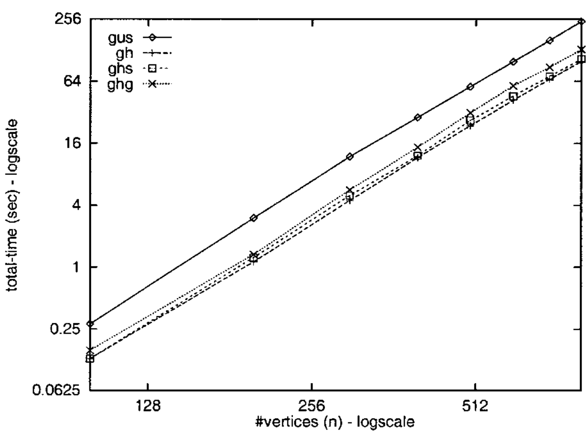

FIG. 6. Running times for NOI2 family.

NOI3								
	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	500	6237	500.000	6237.000	234599	2.420	4.376	6.840
gh	500	6237	500.000	6328.000	322827	3.702	6.062	9.774
ghs	500	6237	500.000	6326.000	223514	3.546	6.032	9.612
ghg	500	6237	500.000	6326.000	460778	6.622	4.520	11.152
#HO	250							
gus	500	12475	500.000	12475.000	227950	4.824	8.222	13.076
gh	500	12475	500.000	12119.600	289029	6.206	12.216	18.450
ghs	500	12475	500.000	12117.600	210491	6.668	12.066	18.760
ghg	500	12475	500.000	12117.600	441605	12.716	9.370	22.104
#HO	250							
gus	500	31187	500.000	31187.000	223723	11.844	17.840	29.728
gh	500	31187	500.000	27831.800	261830	14.004	28.962	42.992
ghs	500	31187	500.000	27829.800	214189	17.026	28.258	45.316
ghg	500	31187	500.000	27829.800	411799	30.082	22.452	52.550
#HO	250							
gus	500	62375	500.000	62375.000	220971	23.178	32.394	55.614
gh	500	62375	500.000	49346.798	252343	25.992	53.198	79.218
ghs	500	62375	500.000	49344.802	215845	33.014	51.830	84.872
ghg	500	62375	500.000	49344.800	380652	53.724	41.768	95.504
#HO	250							
gus	500	93562	500.000	93562.000	220040	33.236	45.304	78.582
gh	500	93562	500.000	66115.799	248833	36.102	74.178	110.296
ghs	500	93562	500.000	66113.801	216280	46.624	71.364	118.014
ghg	500	93562	500.000	66113.800	375963	75.262	58.814	134.092
#HO	250							
gus	500	124750	500.000	124750.000	217498	41.942	56.684	98.662
gh	500	124750	500.000	79109.401	247546	44.302	91.810	136.142
ghs	500	124750	500.000	79107.399	214949	57.942	88.292	146.254
ghg	500	124750	500.000	79107.400	356866	88.402	72.900	161.318
#HO	250							

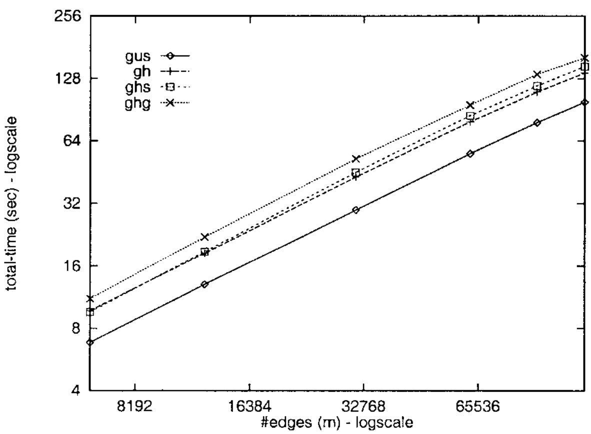

FIG. 7. Running times for NOI3 family.

NOI4								
	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	500	6237	500.000	6237.000	299294	3.166	4.316	7.530
gh	500	6237	252.132	1932.215	202376	1.454	2.534	4.020
ghs	500	6237	251.897	1933.070	126560	1.174	2.504	3.704
ghg	500	6237	252.508	1946.744	285626	2.558	1.712	4.286
#HO	250							
gus	500	12475	500.000	12475.000	256193	5.618	8.190	13.880
gh	500	12475	252.653	3413.276	165373	1.960	4.230	6.220
ghs	500	12475	253.184	3441.145	112729	1.940	4.158	6.124
ghg	500	12475	254.690	3494.726	262285	4.090	2.920	7.018
#HO	250							
gus	500	31187	500.000	31187.000	241792	13.452	17.986	31.484
gh	500	31187	252.846	7428.089	139161	3.912	9.436	13.376
ghs	500	31187	253.974	7547.216	107543	4.622	9.272	13.912
ghg	500	31187	256.278	7739.150	236752	9.766	7.412	17.184
#HO	250							
gus	500	62375	500.000	62375.000	234115	25.362	32.676	58.062
gh	500	62375	252.846	12856.068	130833	6.852	17.210	24.082
ghs	500	62375	257.453	13595.623	107365	9.348	17.004	26.378
ghg	500	62375	261.205	14156.969	217833	17.798	14.544	32.350
#HO	250							
gus	500	93562	500.000	93562.000	233187	36.232	45.568	81.828
gh	500	93562	252.846	17110.384	127955	9.352	23.408	32.776
ghs	500	93562	262.583	19145.877	108811	13.832	24.016	37.878
ghg	500	93562	259.179	18452.244	205689	23.764	19.890	43.666
#HO	250							
gus	500	124750	500.000	124750.000	230629	45.208	56.792	102.040
gh	500	124750	252.550	20354.711	125878	11.144	28.238	39.416
ghs	500	124750	267.743	24122.766	112151	17.680	30.308	48.016
ghg	500	124750	262.722	22895.740	205917	30.128	24.922	55.062
#HO	250							

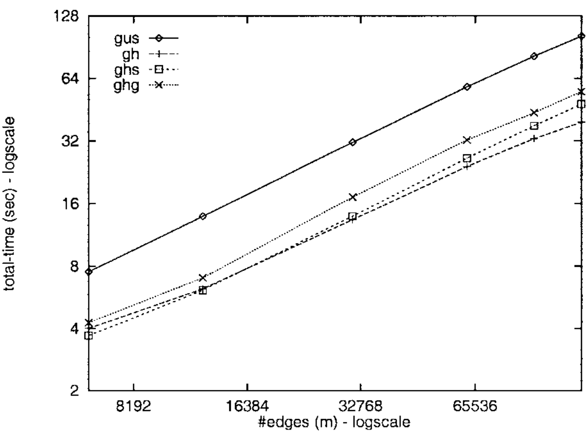

FIG. 8. Running times for NOI4 family.

NOI5									
	N	M	Aver.N	Aver.M	K	Relabels	CTime	MTime	TotTime
gus	500	62375	500.000	62375.000	1	220971	24.022	32.682	56.738
gh	500	62375	500.000	49346.798	1	252343	25.604	52.996	78.626
ghs	500	62375	500.000	49344.802	1	215845	32.550	51.262	83.832
ghg	500	62375	500.000	49344.800	1	380652	53.052	41.544	94.614
#HO	250.0								
gus	500	62375	500.000	62375.000	2	237081	25.706	32.596	58.352
gh	500	62375	252.429	12785.183	2	130191	6.774	17.122	23.926
ghs	500	62375	258.742	13782.186	2	108203	9.424	17.176	26.622
ghg	500	62375	258.939	13810.678	2	216870	17.536	14.348	31.894
#HO	250.2								
gus	500	62375	500.000	62375.000	5	260338	28.826	32.632	61.478
gh	500	62375	106.179	2511.966	5	59546	1.724	3.882	5.640
ghs	500	62375	114.860	3321.283	5	50017	2.388	4.450	6.864
ghg	500	62375	119.040	3821.242	5	101139	4.382	3.784	8.182
#HO	250.8								
gus	500	62375	500.000	62375.000	10	286834	33.108	32.614	65.740
gh	500	62375	64.779	1540.898	10	46341	2.292	2.650	4.962
ghs	500	62375	68.250	1894.698	10	31876	1.882	2.798	4.708
ghg	500	62375	75.330	2587.768	10	66670	3.856	2.788	6.656
#HO	251.8								
gus	500	62375	500.000	62375.000	30	365211	47.326	32.578	79.942
gh	500	62375	68.388	3860.096	30	88462	8.212	5.418	13.654
ghs	500	62375	42.680	1995.552	30	28218	2.648	2.976	5.642
ghg	500	62375	52.471	2915.390	30	66338	7.220	3.280	10.524
#HO	266.0								
gus	500	62375	500.000	62375.000	50	392129	52.442	32.618	85.110
gh	500	62375	105.720	7861.901	50	143414	15.520	10.108	25.648
ghs	500	62375	49.181	2841.665	50	34782	3.630	4.002	7.662
ghg	500	62375	58.218	3728.042	50	92422	11.296	4.440	15.744
#HO	282.4								
gus	500	62375	500.000	62375.000	100	351396	43.902	32.568	76.502
gh	500	62375	192.105	17281.079	100	213174	25.974	21.354	47.362
ghs	500	62375	119.814	9484.272	100	71602	9.538	12.206	21.770
ghg	500	62375	119.526	9710.727	100	189397	28.534	11.648	40.196
#HO	291.2								
gus	500	62375	500.000	62375.000	200	293732	33.912	32.630	66.578
gh	500	62375	308.293	30180.289	200	274840	34.084	35.098	69.202
ghs	500	62375	245.541	23328.355	200	129297	20.714	28.554	49.292
ghg	500	62375	223.251	21197.483	200	332313	58.980	24.424	83.414
#HO	287.4								
gus	500	62375	500.000	62375.000	400	271053	29.974	32.546	62.566
gh	500	62375	396.257	39309.944	400	292451	34.096	43.654	77.768
ghs	500	62375	350.669	34872.916	400	176511	27.914	39.758	67.690
ghg	500	62375	322.797	32046.821	400	483703	84.238	33.234	117.486
#HO	276.0								
gus	500	62375	500.000	62375.000	500	262441	28.658	32.622	61.310
gh	500	62375	411.329	40854.055	500	291816	33.526	45.222	78.778
ghs	500	62375	369.372	36897.739	500	181231	28.600	41.338	69.956
ghg	500	62375	340.988	33999.203	500	517790	89.634	34.482	124.130
#HO	274.8								

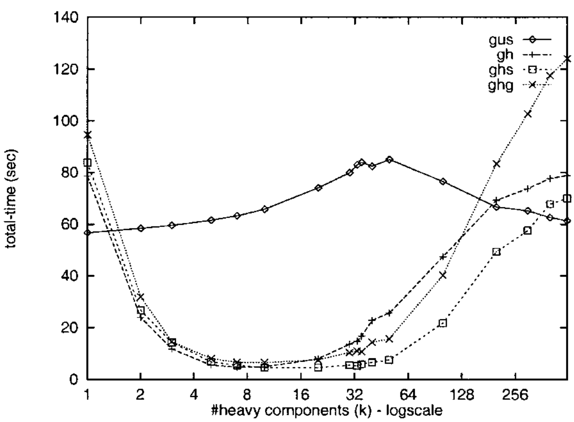

FIG. 9. Running times for NOI5 family.

NOI6									
	N	M	Aver.N	Aver.M	P	Relabels	CTime	MTime	TTime
gus	500	62375	500.000	62375.000	1	219399	23.818	32.858	56.710
gh	500	62375	500.000	49346.798	1	251919	25.696	53.142	78.848
ghs	500	62375	500.000	49344.802	1	214555	32.540	51.210	83.762
ghg	500	62375	500.000	49344.800	1	377032	52.874	41.346	94.228
#HO	250.0								
gus	500	62375	500.000	62375.000	10	228515	25.192	32.910	58.152
gh	500	62375	500.000	49346.798	10	269007	27.750	53.210	80.974
ghs	500	62375	500.000	49344.802	10	224059	33.830	51.176	85.034
ghg	500	62375	500.000	49344.800	10	445558	63.914	41.410	105.342
#HO	250.0								
gus	500	62375	500.000	62375.000	50	248876	28.786	33.030	61.846
gh	500	62375	500.000	49346.798	50	302002	32.308	52.966	85.298
ghs	500	62375	500.000	49344.802	50	246872	37.252	51.208	88.486
ghg	500	62375	500.000	49344.800	50	472712	69.060	41.366	110.440
#HO	250.0								
gus	500	62375	500.000	62375.000	100	279440	32.808	32.976	65.832
gh	500	62375	500.000	49346.798	100	331128	36.240	52.928	89.188
ghs	500	62375	500.000	49344.802	100	274995	40.974	51.220	92.216
ghg	500	62375	500.000	49344.800	100	499262	73.148	41.374	114.538
#HO	250.0								
gus	500	62375	500.000	62375.000	150	298280	35.760	32.944	68.738
gh	500	62375	500.000	49346.798	150	350966	39.718	52.924	92.670
ghs	500	62375	500.000	49344.802	150	292915	44.228	51.254	95.506
ghg	500	62375	500.000	49344.800	150	483343	69.976	41.412	111.406
#HO	250.0								
gus	500	62375	500.000	62375.000	200	295824	35.412	32.874	68.330
gh	500	62375	498.111	49048.220	200	372396	44.042	52.604	96.662
ghs	500	62375	401.013	34743.268	200	227063	31.242	37.206	68.478
ghg	500	62375	401.613	34831.698	200	398620	53.524	30.380	83.922
#HO	250.0								
gus	500	62375	500.000	62375.000	250	236392	25.654	32.856	58.542
gh	500	62375	255.246	13199.009	250	132532	7.156	17.426	24.608
ghs	500	62375	256.773	13486.431	250	106236	9.146	16.862	26.040
ghg	500	62375	261.921	14258.794	250	212484	17.160	14.624	31.802
#HO	250.0								
gus	500	62375	500.000	62375.000	500	237081	25.788	32.710	58.552
gh	500	62375	252.429	12785.183	500	130191	6.850	17.122	23.998
ghs	500	62375	258.742	13782.186	500	108203	9.488	17.126	26.652
ghg	500	62375	258.939	13810.678	500	216870	17.638	14.260	31.916
#HO	250.2								
gus	500	62375	500.000	62375.000	1000	236496	25.784	32.914	58.724
gh	500	62375	252.527	12799.722	1000	135934	7.120	17.110	24.254
ghs	500	62375	257.090	13532.462	1000	108467	9.364	16.912	26.312
ghg	500	62375	258.637	13761.942	1000	214945	17.304	14.246	31.556
#HO	250.2								
gus	500	62375	500.000	62375.000	5000	237857	25.960	32.736	58.742
gh	500	62375	252.329	12770.479	5000	153392	7.924	17.056	25.004
ghs	500	62375	256.846	13494.085	5000	110165	9.480	16.882	26.398
ghg	500	62375	258.707	13769.050	5000	226332	18.336	14.258	32.620
#HO	250.2								

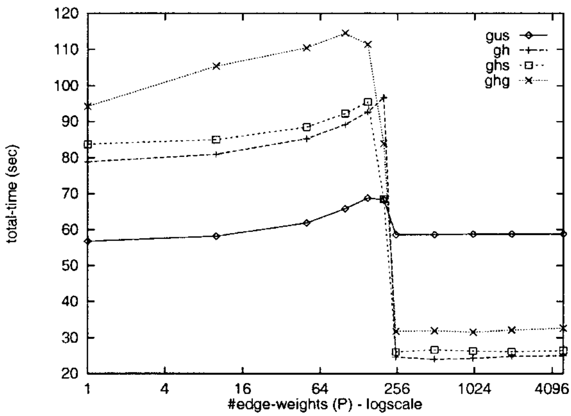

FIG. 10. Running times for NOI6 family.

PATH									
	N	M	Aver.N	Aver.M	K	Relabels	CTime	MTime	TTime
gus	2000	21990	2000.0	21990.0	1	11	3.178	81.58	84.940
gh	2000	21990	2000.0	22864.0	1	3462058	49.170	125.66	174.922
ghs	2000	21990	2000.0	22862.0	1	11	22.004	127.19	149.312
ghg	2000	21990	2000.0	22862.0	1	1674374	62.438	90.64	153.154
#HO	1000.0								
gus	2000	21990	2000.0	21990.0	4	757042	14.064	81.74	95.970
gh	2000	21990	1989.4	22709.4	4	5323846	72.246	124.90	197.262
ghs	2000	21990	504.1	2776.6	4	7483	2.604	40.12	42.844
ghg	2000	21990	505.6	2794.4	4	254095	5.036	22.09	27.202
#HO	1000.4								
gus	2000	21990	2000.0	21990.0	15	2950640	41.560	82.01	123.740
gh	2000	21990	1778.3	20386.2	15	7040216	118.074	114.96	233.150
ghs	2000	21990	143.3	628.7	15	30779	1.218	31.91	33.238
ghg	2000	21990	149.9	707.4	15	118591	2.026	15.90	17.988
#HO	1002.8								
gus	2000	21990	2000.0	21990.0	50	5057903	75.244	81.90	157.256
gh	2000	21990	1260.4	15504.2	50	6536603	125.182	94.26	219.540
ghs	2000	21990	60.5	381.1	50	90910	2.072	31.21	33.356
ghg	2000	21990	76.2	581.4	50	177543	3.660	15.95	19.646
#HO	1011.4								
gus	2000	21990	2000.0	21990.0	200	6804712	103.848	81.88	185.902
gh	2000	21990	266.6	3667.4	200	2037790	38.976	44.38	83.448
ghs	2000	21990	43.0	458.3	200	196448	4.106	31.61	35.854
ghg	2000	21990	66.7	767.8	200	347546	7.860	17.01	24.926
#HO	1047.8								
gus	2000	21990	2000.0	21990.0	800	13343244	229.246	81.92	311.344
gh	2000	21990	124.9	1846.7	800	1658894	34.674	36.91	71.702
ghs	2000	21990	64.5	956.4	800	499685	10.706	33.58	44.392
ghg	2000	21990	85.6	1229.8	800	1243340	31.410	21.02	52.488
#HO	1188.8								
gus	2000	21990	2000.0	21990.0	2000	28851669	527.552	81.86	609.586
gh	2000	21990	126.6	2030.5	2000	2671666	58.364	37.49	95.944
ghs	2000	21990	103.9	1701.2	2000	1501881	32.634	36.37	69.100
ghg	2000	21990	100.3	1576.6	2000	3371605	86.170	24.91	111.134
#HO	1335.0								

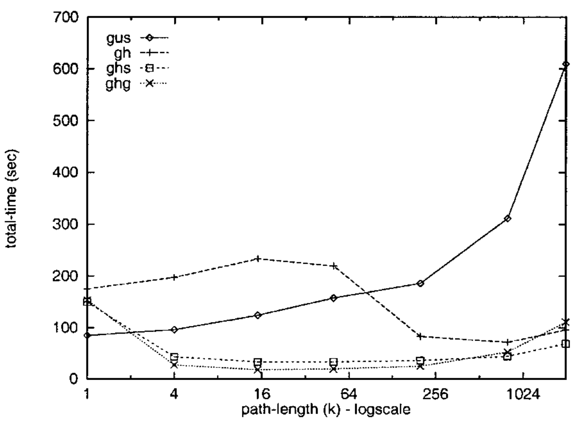

FIG. 11. Running times for PATH family.

PR1								
	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	200	583	200.000	583.400	38043	0.134	0.164	0.304
gh	200	583	191.417	667.144	60243	0.242	0.192	0.448
ghs	200	583	180.615	630.366	32129	0.130	0.190	0.328
ghg	200	583	178.357	621.946	73554	0.312	0.114	0.432
#HO	103.4							
gus	400	1968	400.000	1968.000	145921	0.914	1.190	2.146
gh	400	1968	399.895	2169.843	228495	1.272	1.682	2.964
ghs	400	1968	399.645	2166.681	136425	0.934	1.676	2.630
ghg	400	1968	398.960	2163.322	286573	1.818	1.162	2.986
#HO	200.2							
gus	600	4157	600.000	4157.000	306644	2.056	4.678	6.798
gh	600	4157	600.000	4459.000	481212	3.156	6.018	9.200
ghs	600	4157	600.000	4457.000	298350	2.464	6.002	8.492
ghg	600	4157	599.375	4453.666	607139	4.894	3.982	8.890
#HO	300.2							
gus	800	7175	800.000	7175.200	549325	4.992	10.318	15.374
gh	800	7175	800.000	7577.200	826197	7.628	14.972	22.640
ghs	800	7175	800.000	7575.200	506020	6.852	15.320	22.214
ghg	800	7175	799.352	7570.602	1046093	12.028	10.342	22.388
#HO	400.2							
gus	1000	10923	1000.000	10923.400	865279	9.426	19.476	28.960
gh	1000	10923	1000.000	11425.400	1252763	13.896	27.822	41.772
ghs	1000	10923	1000.000	11423.400	806876	13.060	28.174	41.294
ghg	1000	10923	999.543	11418.585	1550385	21.372	19.806	41.198
#HO	500.2							

FIG. 12. Running times for PR1 family.

PR5								
	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	200	583	200.000	583.400	63118	0.208	0.160	0.382
gh	200	583	79.653	273.606	34759	0.144	0.102	0.262
ghs	200	583	61.963	214.261	15153	0.046	0.120	0.172
ghg	200	583	60.636	208.331	30988	0.124	0.092	0.222
#HO	114.6							
gus	400	1968	400.000	1968.000	222721	1.432	1.208	2.682
gh	400	1968	193.153	858.583	150966	0.748	0.794	1.562
ghs	400	1968	182.832	816.417	81576	0.484	0.814	1.330
ghg	400	1968	180.921	808.056	184164	1.024	0.558	1.590
#HO	208.0							
gus	600	4157	600.000	4157.000	470455	3.088	4.652	7.774
gh	600	4157	299.150	1629.344	322511	1.758	3.310	5.102
ghs	600	4157	295.839	1614.090	186273	1.300	3.220	4.558
ghg	600	4157	296.005	1617.053	432119	2.718	1.966	4.702
#HO	302.2							
gus	800	7175	800.000	7175.200	790617	6.824	10.430	17.296
gh	800	7175	400.874	2585.057	530152	3.318	7.700	11.062
ghs	800	7175	400.393	2589.191	320746	2.796	7.856	10.672
ghg	800	7175	401.240	2601.755	767179	6.326	4.768	11.116
#HO	400.6							
gus	1000	10923	1000.000	10923.400	1220926	13.242	19.518	32.870
gh	1000	10923	501.456	3727.116	795713	5.606	13.420	19.078
ghs	1000	10923	501.725	3734.425	506897	5.010	13.252	18.318
ghg	1000	10923	502.742	3749.011	1128825	10.292	8.218	18.542
#HO	500.0							

FIG. 13. Running times for PR5 family.

PR6	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	200	2172	200.000	2171.800	41936	0.374	0.340	0.730
gh	200	2172	101.361	742.568	28229	0.186	0.246	0.438
ghs	200	2172	102.459	761.375	16747	0.138	0.260	0.408
ghg	200	2172	103.509	778.459	39328	0.270	0.192	0.466
#HO	100.2							
gus	400	8307	400.000	8307.200	162709	2.888	4.062	7.000
gh	400	8307	201.997	2490.413	103221	1.092	2.052	3.156
ghs	400	8307	202.097	2500.436	69329	0.996	2.012	3.040
ghg	400	8307	203.189	2533.501	157521	1.886	1.458	3.352
#HO	200.0							
gus	600	18481	600.000	18481.400	360964	10.408	14.766	25.230
gh	600	18481	301.499	5197.957	219253	3.422	7.896	11.352
ghs	600	18481	302.198	5241.647	154055	3.584	7.696	11.312
ghg	600	18481	303.385	5295.499	354662	7.200	5.606	12.816
#HO	300.8							
gus	800	32740	800.000	32740.000	633331	25.330	33.666	59.058
gh	800	32740	401.499	8946.378	378635	8.162	19.722	27.922
ghs	800	32740	402.099	8999.726	277903	9.026	19.378	28.446
ghg	800	32740	403.189	9065.754	626897	18.996	14.686	33.702
#HO	401.0							
gus	1000	50959	1000.000	50959.200	992344	50.222	64.026	114.340
gh	1000	50959	501.500	13734.130	579733	16.132	38.800	54.976
ghs	1000	50959	502.399	13824.069	439993	18.912	38.036	57.010
ghg	1000	50959	503.391	13899.077	981210	39.670	29.632	69.342
#HO	500.8							

FIG. 14. Running times for PR6 family.

PR7	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotalTime
gus	200	10053	200.000	10053.400	36113	1.486	2.048	3.552
gh	200	10053	101.497	2693.447	20295	0.422	0.866	1.296
ghs	200	10053	101.995	2753.068	15494	0.486	0.842	1.338
ghg	200	10053	102.570	2796.854	30685	0.782	0.664	1.454
#HO	100.8							
gus	300	22564	300.000	22564.400	83127	5.790	7.862	13.680
gh	300	22564	151.498	5917.411	45649	1.638	3.714	5.362
ghs	300	22564	152.096	6019.273	35553	1.916	3.624	5.550
ghg	300	22564	153.171	6140.391	71827	3.434	2.878	6.324
#HO	151.0							
gus	400	40047	400.000	40047.200	149747	15.426	18.350	33.822
gh	400	40047	201.499	10343.649	80879	4.006	10.230	14.256
ghs	400	40047	201.997	10465.661	65574	5.258	9.810	15.100
ghg	400	40047	202.980	10613.285	127392	9.248	8.108	17.372
#HO	201.0							
gus	500	62596	500.000	62595.800	232184	30.850	36.056	66.946
gh	500	62596	251.499	16100.493	126032	8.430	20.584	29.040
ghs	500	62596	251.998	16253.585	102698	10.978	19.930	30.934
ghg	500	62596	252.984	16438.631	200462	19.640	16.364	36.008
#HO	251.0							
gus	600	90090	600.000	90090.400	337965	55.394	62.294	117.734
gh	600	90090	301.499	23028.875	181342	14.830	35.610	50.476
ghs	600	90090	302.298	23280.924	152023	19.902	34.426	54.358
ghg	600	90090	303.583	23570.435	297449	36.224	28.616	64.858
#HO	301.0							

FIG. 15. Running times for PR7 family.

PR8	N	M	Aver.N	Aver.M	Relabels	CutTime	ManTime	TotTime
gus	200	19694	200.000	19693.800	35768	3.246	4.378	7.654
gh	200	19694	101.497	5074.955	19206	0.834	1.918	2.760
ghs	200	19694	102.492	5269.992	15394	1.046	1.864	2.922
ghg	200	19694	103.941	5485.778	28500	1.732	1.566	3.298
#HO	101.0							
gus	300	44397	300.000	44396.600	81803	12.316	15.132	27.486
gh	300	44397	151.498	11324.736	43258	3.236	8.272	11.518
ghs	300	44397	151.997	11507.513	35758	4.464	7.992	12.468
ghg	300	44397	152.974	11725.606	62164	6.904	6.686	13.604
#HO	151.0							
gus	400	79002	400.000	79002.200	147182	32.698	35.890	68.626
gh	400	79002	201.499	20051.519	77550	8.344	20.346	28.712
ghs	400	79002	202.596	20474.217	66382	11.590	19.636	31.244
ghg	400	79002	204.168	20941.947	114847	18.358	16.532	34.900
#HO	201.0							
gus	500	123495	500.000	123495.000	230464	66.368	70.920	137.316
gh	500	123495	251.499	31249.892	121358	17.036	39.992	57.056
ghs	500	123495	252.597	31779.008	102978	23.276	38.358	61.654
ghg	500	123495	253.582	32145.071	171577	35.516	32.296	67.826
#HO	250.8							
gus	600	177903	600.000	177903.000	333353	116.038	124.000	240.084
gh	600	177903	301.499	44922.063	175338	30.440	69.726	100.206
ghs	600	177903	302.497	45513.105	152255	41.236	66.304	107.574
ghg	600	177903	303.980	46174.351	260187	66.522	56.506	123.056
#HO	301.0							

FIG. 16. Running times for PR8 family.

REG2	N	M	Aver.N	Aver.M	Relabels	CutTime	ManipTime	TotalTime
gus	50	1250	50.000	1250.000	2248	0.038	0.032	0.070
gh	50	1250	50.000	827.000	2252	0.030	0.038	0.070
ghs	50	1250	50.000	825.000	2240	0.034	0.040	0.076
ghg	50	1250	23.082	326.418	3151	0.036	0.024	0.062
#HO	49							
gus	100	2500	100.000	2500.000	9439	0.156	0.114	0.284
gh	100	2500	100.000	2036.000	9446	0.138	0.166	0.316
ghs	100	2500	100.000	2034.000	9464	0.186	0.172	0.362
ghg	100	2500	42.303	841.432	11548	0.188	0.112	0.308
#HO	99							
gus	200	5000	200.000	5000.000	38813	0.640	0.750	1.404
gh	200	5000	200.000	4521.200	38775	0.608	1.270	1.884
ghs	200	5000	200.000	4519.200	38705	0.870	1.234	2.112
ghg	200	5000	82.338	2064.235	42566	1.054	0.612	1.676
#HO	199							
gus	400	10000	400.000	10000.000	157202	3.202	4.958	8.196
gh	400	10000	400.000	9626.800	156654	3.094	7.204	10.314
ghs	400	10000	400.000	9624.800	157579	4.188	7.188	11.400
ghg	400	10000	144.792	4299.959	137136	4.426	3.400	7.850
#HO	399							
gus	800	20000	800.000	20000.000	632618	14.580	22.564	37.210
gh	800	20000	800.000	19806.600	631305	14.172	34.688	48.890
ghs	800	20000	800.000	19804.600	629028	20.088	34.474	54.594
ghg	800	20000	257.503	8466.638	449468	17.950	17.092	35.088
#HO	799							

FIG. 17. Running times for REG2 family.

REG1								
	N	M	Aver.N	Aver.M	Relabels	CutTime	ManipTime	TotalTime
gus	301	301	301.000	301.000	89102	0.224	0.250	0.496
gh	301	301	301.000	453.500	89101	0.264	0.314	0.594
ghs	301	301	301.000	451.500	89102	0.224	0.376	0.614
ghg	301	301	301.000	451.500	89102	0.370	0.330	0.710
#HO	300							
gus	301	602	301.000	602.000	92461	0.300	0.322	0.644
gh	301	602	301.000	752.100	90589	0.320	0.446	0.776
ghs	301	602	301.000	750.100	82518	0.298	0.432	0.748
ghg	301	602	79.731	237.356	41615	0.246	0.258	0.518
#HO	300							
gus	301	1505	301.000	1505.000	88339	0.436	0.462	0.914
gh	301	1505	301.000	1638.900	86939	0.452	0.780	1.246
ghs	301	1505	301.000	1636.900	87014	0.480	0.790	1.280
ghg	301	1505	99.064	714.244	58392	0.438	0.402	0.856
#HO	300							
gus	301	4816	301.000	4816.000	87408	1.048	1.488	2.564
gh	301	4816	301.000	4743.900	88418	1.148	2.402	3.562
ghs	301	4816	301.000	4741.900	87735	1.430	2.406	3.854
ghg	301	4816	107.843	2120.813	73879	1.520	1.144	2.676
#HO	300							
gus	301	15050	301.000	15050.000	88836	3.276	5.052	8.358
gh	301	15050	301.000	12999.900	88677	3.384	7.766	11.162
ghs	301	15050	301.000	12997.900	88771	4.550	7.608	12.174
ghg	301	15050	126.786	5847.286	113435	6.424	3.692	10.132
#HO	300							
gus	301	49966	301.000	49966.000	88942	9.778	13.180	22.982
gh	301	49966	301.000	30416.700	89063	9.390	20.888	30.292
ghs	301	49966	301.000	30414.700	88995	12.878	19.846	32.740
ghg	301	49966	136.869	12248.858	148442	20.932	9.520	30.472
#HO	300							
gus	301	90300	301.000	90300.000	89105	13.822	18.382	32.226
gh	301	90300	301.000	39205.700	89148	12.958	28.680	41.664
ghs	301	90300	301.000	39203.700	89147	17.598	26.914	44.532
ghg	301	90300	170.299	18951.678	187139	34.972	15.470	50.452
#HO	300							

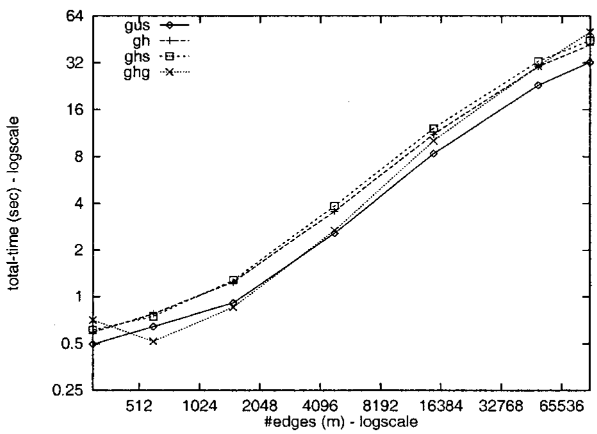

FIG. 18. Running times for REG1 family.

TREE									
	N	M	Aver.N	Aver.M	K	Relabels	CTime	MTime	TTime
gus	800	160600	800.0	160600.0	1	127	14.294	142.540	156.884
gh	800	160600	800.0	126637.9	1	654497	126.490	228.620	355.154
ghs	800	160600	800.0	126635.9	1	127	49.686	219.462	269.194
ghg	800	160600	800.0	126636.0	1	353357	174.360	179.180	353.564
#HO	400.0								
gus	800	160600	800.0	160600.0	3	170830	43.622	140.792	184.458
gh	800	160600	793.2	125034.0	3	765747	150.172	225.952	376.166
ghs	800	160600	403.7	38863.5	3	7532	18.594	79.312	97.942
ghg	800	160600	403.9	38939.4	3	130896	49.464	66.936	116.416
#HO	400.2								
gus	800	160600	800.0	160600.0	5	237671	55.274	139.952	195.302
gh	800	160600	792.8	124943.0	5	849349	168.986	225.898	394.926
ghs	800	160600	296.0	22990.6	5	11237	11.914	49.046	61.000
ghg	800	160600	296.3	23121.2	5	91232	28.274	40.404	68.692
#HO	401.2								
gus	800	160600	800.0	160600.0	10	349191	75.138	139.712	214.914
gh	800	160600	789.1	124159.1	10	968735	198.038	224.254	422.344
ghs	800	160600	194.2	11909.4	10	16703	7.320	27.390	34.748
ghg	800	160600	195.3	12186.3	10	67741	15.468	21.994	37.484
#HO	402.0								
gus	800	160600	800.0	160600.0	20	451983	93.642	142.118	235.822
gh	800	160600	776.1	121513.2	20	1055504	221.666	220.828	442.546
ghs	800	160600	131.7	7341.1	20	22917	5.820	18.552	24.422
ghg	800	160600	134.1	7860.0	20	61922	12.026	15.014	27.058
#HO	405.6								
gus	800	160600	800.0	160600.0	50	545664	109.066	140.932	250.064
gh	800	160600	727.9	112241.4	50	1109984	245.172	208.126	453.352
ghs	800	160600	99.2	6610.6	50	37107	7.410	17.400	24.842
ghg	800	160600	105.8	7734.6	50	87509	17.582	15.592	33.192
#HO	411.0								
gus	800	160600	800.0	160600.0	100	598344	117.590	139.492	257.152
gh	800	160600	664.0	101533.4	100	1054070	235.968	193.200	429.190
ghs	800	160600	102.3	8889.5	100	55722	11.392	22.124	33.556
ghg	800	160600	113.3	10720.5	100	162158	35.394	21.240	56.658
#HO	423.2								
gus	800	160600	800.0	160600.0	200	644957	128.920	139.846	268.820
gh	800	160600	594.6	91044.8	200	1005458	233.726	176.602	410.360
ghs	800	160600	121.0	13033.3	200	86009	19.156	30.694	49.906
ghg	800	160600	136.4	15507.1	200	303894	74.578	31.124	105.720
#HO	440.2								
gus	800	160600	800.0	160600.0	400	704840	143.538	139.092	282.692
gh	800	160600	508.5	78112.8	400	930439	220.754	156.168	376.946
ghs	800	160600	156.7	19788.6	400	123498	30.248	44.200	74.490
ghg	800	160600	169.3	22068.0	400	502969	137.732	45.582	183.340
#HO	463.4								
gus	800	160600	800.0	160600.0	800	809715	170.614	138.424	309.094
gh	800	160600	449.8	69666.4	800	895883	219.890	142.246	362.182
ghs	800	160600	192.2	26378.7	800	166622	43.002	57.154	100.194
ghg	800	160600	192.7	26883.6	800	696121	204.622	56.716	261.368
#HO	479.2								

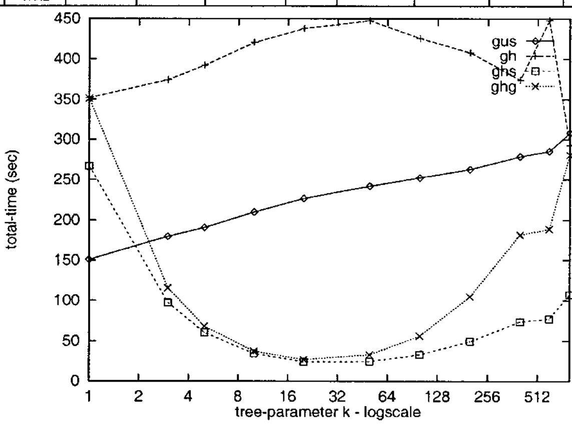

FIG. 19. Running times for TREE family.

TSP										
	Name	N	M	Aver.N	Aver.M	Relabels	CTime	MTime	TTime
gus	tsp.	76	90	76.0	90.0	11266	0.028	0.018	0.052
gh	pr	76	90	26.3	47.1	3349	0.018	0.016	0.034
ghs	76.	76	90	30.8	53.5	4535	0.016	0.014	0.034
ghg	x.2	76	90	14.4	23.7	1406	0.014	0.018	0.040
#HO		75							
gus	tsp.	532	787	532.0	787.0	1063351	3.192	1.636	4.866
gh	att	532	787	44.0	90.3	113526	0.318	1.160	1.524
ghs	532.	532	787	52.6	104.6	153862	0.416	1.104	1.566
ghg	x.1	532	787	21.6	40.6	37661	0.184	1.002	1.208
#HO		499							
gus	tsp.	1084	1252	1084.0	1252.0	4620e3	14.61	11.09	25.80
gh	vm	1084	1252	151.0	251.2	692e3	1.55	9.23	10.83
ghs	1084.	1084	1252	1.443	2.495	729e3	1.56	9.33	10.94
ghg	x.1	1084	1252	50.2	79.1	111e3	0.47	7.764	8.30
#HO		1083							
gus	tsp.	1323	2195	1323.0	2195.0	13154e3	54.94	18.99	74.05
gh	rl	1323	2195	136.0	331.1	1858e3	6.01	14.00	20.08
ghs	1323.	1323	2195	161.6	367.1	2587e3	8.54	14.08	22.70
ghg	x.2	1323	2195	107.4	242.0	1776e3	6.98	10.97	18.02
#HO		1146							
gus	tsp.	1748	2336	1748.0	2336.0	14378e3	52.29	30.92	83.35
gh	vm	1748	2336	84.0	179.3	1690e3	4.75	23.89	28.73
ghs	1748.	1748	2336	146.6	276.7	2141e3	6.07	24.02	30.21
ghg	x.1	1748	2336	76.4	139.6	857e3	3.26	19.76	23.09
#HO		1558							
gus	tsp.	5934	7287	5934.0	7287.0	215e6	787.05	357.12	1144.68
gh	r	5934	7287	94.7	179.3	6548e3	18.15	272.39	290.84
ghs	15934.	5934	7287	166.7	290.1	10326e3	27.86	273.57	301.76
ghg	x.1	5934	7287	67.3	113.7	4211e3	16.45	237.13	253.89
#HO		5575							
gus	tsp.	5934	7627	5934.0	7627.0	323e6	1277.87	376.52	1654.93
gh	r	5934	7627	124.6	248.8	11642e3	30.82	273.45	304.56
ghs	15934.	5934	7627	160.5	294.5	21090e3	56.71	272.83	329.84
ghg	x.2	5934	7627	88.8	158.1	7401e3	28.91	223.50	252.70
#HO		5309							
gus	usa	13509	15631	13509.0	15631.0	531e6	2520.30	1538.14	4059.54
gh	13509.	13509	15631	214.0	390.3	30e6	102.20	1073.83	1176.74
ghs	xo.	13509	15631	381.4	662.4	45e6	157.45	1093.62	1251.71
ghg	15631	13509	15631	140.7	239.2	18e6	90.40	935.78	1026.80
#HO		12273							
gus	usa	13509	17494	13509.0	17494.0	549e6	2497.66	1926.69	4425.51
gh	13509.	13509	17494	509.4	1011.7	69e6	230.62	1375.33	1606.70
ghs	xo.	13509	17494	1029.4	1907.0	81e6	273.42	1411.00	1685.11
ghg	17494	13509	17494	357.5	651.6	52e6	229.56	1254.50	1484.75
#HO		13122							
gus	d	15112	19057	15112.0	19057.0	646e6	3094.80	2483.76	5579.82
gh	15112.	15112	19057	765.8	1491.2	96e6	347.44	1828.83	2177.02
ghs	xo.	15112	19057	1470.4	2678.7	130e6	488.61	1917.67	2407.07
ghg	19057	15112	19057	675.6	1215.4	111e6	542.74	1744.84	2288.32
#HO		14817							

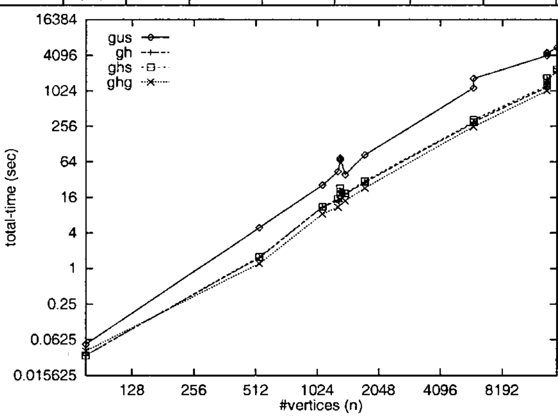

FIG. 20. Running times for TSP family.

WHE	N	M	Aver.N	Aver.M	Relabels	CutTime	ManipTime	TotalTime
gus	64	126	64.000	126.000	20093	0.056	0.018	0.082
gh	64	126	64.000	160.000	18083	0.052	0.014	0.070
ghs	64	126	64.000	158.000	20186	0.066	0.014	0.082
ghg	64	126	26.317	63.333	9851	0.036	0.018	0.060
#HO	63							
gus	128	254	128.000	254.000	132799	0.440	0.058	0.504
gh	128	254	128.000	320.000	128092	0.418	0.062	0.484
ghs	128	254	128.000	318.000	125855	0.460	0.056	0.522
ghg	128	254	51.992	127.500	63320	0.240	0.042	0.288
#HO	127							
gus	256	510	256.000	510.000	826401	3.204	0.232	3.452
gh	256	510	256.000	640.000	803582	3.032	0.316	3.360
ghs	256	510	256.000	638.000	741497	3.052	0.304	3.366
ghg	256	510	102.863	254.655	367431	1.518	0.194	1.728
#HO	255							
gus	512	1022	512.000	1022.000	5062917	22.178	1.790	24.008
gh	512	1022	512.000	1280.000	5355418	23.732	2.146	25.902
ghs	512	1022	512.000	1278.000	4573090	22.300	2.116	24.430
ghg	512	1022	203.014	505.063	1989225	9.480	1.338	10.842
#HO	511							
gus	1024	2046	1024.000	2046.000	32290667	157.518	9.700	167.292
gh	1024	2046	1024.000	2560.000	35834725	189.736	12.724	202.532
ghs	1024	2046	1024.000	2558.000	29847073	178.860	12.512	191.418
ghg	1024	2046	409.289	1020.676	11480901	61.508	8.688	70.242
#HO	1023							

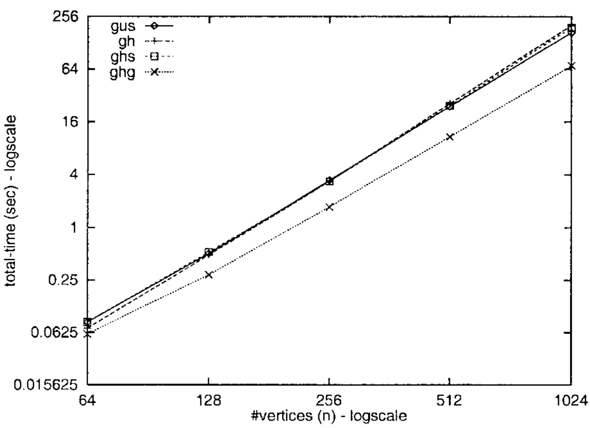

FIG. 21. Running times for WHE family.

REFERENCES

[AC97] D. L. Applegate and W. J. Cook, Personal communication, Rice University, 1997.

[AS93] R. J. Anderson and J. C. Setubal. Goldberg's Algorithm for the Maximum Flow in Perspective: A Computational Study, AMS, pp. 1–18, *in* "Network Flows and Matchings: First DIMACS Implementation Challenge," (D. S. Johnson and C. C. McGeoch, Eds.), 1993.

[CG97] B. V. Cherkassky and A. V. Goldberg, On implementing push-relabel method for the maximum flow problem, *Algorithmica* **19** (1997), 390–410.

[CGK+97] C. S. Cherkuri, A. V. Goldberg, D. R. Karger, M. S. Levine, and C. Stein, Experimental study of minimum cut algorithms, *in* "Proc. 8th ACM-SIAM Symposium on Discrete Algorithms," pp. 324–333, 1997.

[DM89] U. Derigs and W. Meier, Implementing Goldberg's max-flow algorithm—a computational investigation, *in* "ZOR—Methods and Models of Operations Research," Vol. 33, pp. 383–403, 1989.

[GH61] R. E. Gomory and T. C. Hu, Multi-terminal network flows, *J. SIAM*, **9** (1961), 551-570.

[Gol87] A. V. Goldberg, "Efficient Graph Algorithms for Sequential and Parallel Computers," PhD Thesis, M.I.T., January 1987. [Also available as Technical Report TR-374, Lab. for Computer Science, M.I.T., 1987]

[GR98] A. V. Goldberg and S. Rao, Beyond the flow decomposition barrier, *J. Assoc. Comput. Mach.* **45** (1998), 753–782.

[Gro97] M. Groetschel, Personal communication, ZIB Berlin, 1997.

[GT88] A. V. Goldberg and R. E. Tarjan, A new approach to the maximum flow problem, *J. Assoc. Comput. Mach.* **35** (1988), 921–940.

[Gus90] D. Gusfield, Very simple methods for all pairs network flow analysis, *SIAM J. Comput.* **19** (1990), 143–155.

[HO94] J. Hao and J. B. Orlin, A faster algorithm for finding the minimum cut in a directed graph, *J. Algorithms* **17** (1994), 424–446.

[Hu82] T. C. Hu, "Combinatorial Algorithms," Addison-Wesley, Reading, MA, 1982.

[Lev97] M. S. Levine, "Experimental Study of Minimum Cut Algorithms," Technical Report MIT-LCS-TR-719, Lab for Computer Science, MIT, 1997.

[LFF62] Jr. L. R. Ford and D. R. Fulkerson, "Flows in Networks," Princeton Univ. Press, Princeton, NJ, 1962.

[NOI94] H. Nagamochi, T. Ono, and T. Ibaraki, Implementing an efficient minimum capacity cut algorithm, *Math. Prog.* **67** (1994), 297–324.

[NV93] Q. C. Nguyen and V. Venkateswaran, Implementations of Goldberg-Targan Maximum Flow Algorithm, AMS, pp. 19–42, *in* "Network Flows and Matchings: First DIMACS Implementation Challenge," (D. S. Johnson and C. C. McGeoch, Eds.), 1993.

[PR90] M. Padberg and G. Rinaldi, An efficient algorithm for the minimum capacity cut problem, *Math Prog.* **41** (1990), 19–36.

[RT97] M. Juenger, G. Rinaldi, and S. Thienel, Practical performance of efficient minimum cut algorithms, *in* "Proc. 1st Workshop on Algorithm Engineering, Venice, Italy, 1997." [Available at http://www.dsi.unive.it/ wae97/proceedings/.]