APPROXIMATING THE MINIMUM EQUIVALENT DIGRAPH

SAMIR KHULLER *, BALAJI RAGHAVACHARI †, AND NEAL YOUNG ‡

Abstract. The MEG (minimum equivalent graph) problem is the following: "Given a directed graph, find a smallest subset of the edges that maintains all reachability relations between nodes." This problem is NP-hard; this paper gives an approximation algorithm achieving a performance guarantee of about 1.64 in polynomial time. The algorithm achieves a performance guarantee of 1.75 in the time required for transitive closure.

The heart of the MEG problem is the minimum SCSS (strongly connected spanning subgraph) problem — the MEG problem restricted to strongly connected digraphs. For the minimum SCSS problem, the paper gives a practical, nearly linear-time implementation achieving a performance guarantee of 1.75.

The algorithm and its analysis are based on the simple idea of contracting long cycles. The analysis applies directly to 2-Exchange, a general "local improvement" algorithm, showing that its performance guarantee is 1.75.

AMS subject classifications. 68R10, 90C27, 90C35, 05C85, 68Q20.

Key words. directed graph, approximation algorithm, strong connectivity, local improvement.

1. Introduction.

Connectivity is fundamental to the study of graphs and graph algorithms. Recently, many approximation algorithms for finding minimum subgraphs that meet given connectivity requirements have been developed [1, 9, 11, 15, 16, 24]. These results provide practical approximation algorithms for NP-hard network-design problems via an increased understanding of connectivity properties.

Until now, the techniques developed have been applicable only to *undirected* graphs. We consider a basic network-design problem in *directed* graphs [2, 12, 13, 18] which is as follows: given a digraph, find a smallest subset of the edges (forming a *minimum equivalent graph (MEG)*) that maintains all reachability relations of the original graph.

When the MEG problem is restricted to strongly-connected graphs we call it *the minimum SCSS (strongly connected spanning subgraph) problem.* When the MEG problem is restricted to acyclic graphs we call it *the acyclic MEG problem.* The MEG problem reduces in linear time [5] to a single acyclic problem given by the so-called "strong component graph", together with one minimum SCSS problem for each strong component (given by the subgraph induced by that component). Furthermore, the reduction preserves approximation, in the sense that c-approximate solutions to the subproblems yield a c-approximate solution to the original problem.

Moyles and Thompson [18] observe this decomposition and give exponential-time algorithms for the restricted problems. Hsu [13] gives a polynomial-time algorithm for the acyclic MEG problem.

The related problem of finding a *transitive reduction* of a digraph — a smallest set of edges yielding the same reachability relations is studied by Aho, Garey and Ullman [2]. Transitive reduction differs from the MEG problem in that the edges

* Computer Science Department and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF Research Initiation Award CCR-9307462. E-mail : samir@cs.umd.edu.

† Computer Science Department, The University of Texas at Dallas, Richardson, TX 75083-0688. E-mail : rbk@utdallas.edu.

‡ School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853-3801. Part of this work was done while at UMIACS and supported in part by NSF grants CCR-8906949 and CCR-9111348. E-mail : ney@orie.cornell.edu.

in the transitive reduction are not required to be in the original graph. However, the transitive reduction problem decomposes just like the MEG problem into acyclic and strongly connected instances. For any strongly connected instance, a transitive reduction is given by any Hamilton cycle through the vertices. For an acyclic instance, the transitive reduction is unique and, as Aho et al. observe, is equivalent to the MEG problem: it consists of those edges (u, v) for which there is no alternate path from u to v. In fact, Aho, Garey and Ullman show that the transitive reduction problem is *equivalent* to the transitive closure problem. Thus, the acyclic MEG problem reduces to transitive closure.

The acyclic MEG problem can be solved in polynomial time, whereas the minimum SCSS problem is NP-hard [8]. Consequently, this paper focuses on approximation algorithms for the minimum SCSS problem. By the observations of the preceding paragraphs, the performance guarantees obtained for the minimum SCSS problem carry over to the general MEG problem with the overhead of solving a single instance of transitive closure.

1.1. Our Results. Given a strongly connected graph, our basic algorithm finds as long a cycle as it can, contracts the cycle, and recurses. The contracted graph remains strongly connected. When the graph finally collapses into a single vertex, the algorithm returns the set of edges contracted during the course of the algorithm as the desired SCSS.

The algorithm achieves a performance guarantee of any constant *greater* than $\pi^2/6 \approx 1.645$ in polynomial time. We give a nearly linear-time version that achieves a performance guarantee of 1.75. We give examples showing lower bounds on the performance guarantees of the algorithm. For the general algorithm, the lower bounds are slightly above 1.5. For the nearly linear-time version, the lower bound is 1.75, matching the upper bound.

The performance guarantee analysis extends directly to a simple "local improvement" algorithm called 2-EXCHANGE. 2-EXCHANGE starts with the given digraph and performs the following local improvement step as long as it is applicable: find two edges in the current graph that can be replaced by one edge from the original graph, maintaining strong connectivity. Similar local-improvement algorithms are natural candidates for many optimization problems but often elude analysis. We prove that the performance guarantee of 2-EXCHANGE is 1.75.

A natural improvement to the cycle-contraction algorithm is to modify the algorithm to solve the problem optimally once the contracted graph has no cycles longer than a given length c. For instance, for $c = 3$, this modification improves the performance guarantee to $\pi^2/6 - 1/36 \approx 1.617$. We use SCSS_c to denote the minimum SCSS problem restricted to digraphs with no cycle longer than c. The minimum SCSS_2 problem is trivial. The minimum SCSS_3 problem can be solved in polynomial time, as shown by Khuller, Raghavachari and Young [14]. However, further improvement in this direction is limited: we show that the minimum SCSS_5 problem is NP-hard. In fact, we show that the minimum SCSS_{17} problem is MAX SNP-hard. This precludes the possibility of a polynomial-time approximation scheme, assuming P$\neq$NP [4].

1.2. Other Related Work. The union of any incoming branching and any outgoing branching from the same root yields an SCSS with at most $2n - 2$ edges (where n is the number of vertices in the graph). This is a special case of the algorithm given by Frederickson and JáJá [6] that uses minimum weight branchings to achieve a performance guarantee of 2 for weighted graphs. Since any SCSS has at least n edges, this yields a performance guarantee of 2 for the SCSS problem.

Any *minimal* SCSS (one from which no edge can be deleted) has at most $2n-2$ edges and also yields a performance guarantee of 2. The problem of efficiently finding a minimal SCSS is studied by Simon [21]. Gibbons, Karp, Ramachandran, Soroker and Tarjan [10] give a parallel algorithm.

A related problem in undirected graphs is to find a smallest subset of the edges forming a biconnected (respectively bridge-connected (i.e., 2-edge-connected)) spanning subgraph of a given graph. These problems are NP-hard. Khuller and Vishkin [15] give a DFS-based algorithm that achieves a factor of $\frac{5}{3}$ for biconnectivity and $\frac{3}{2}$ for bridge-connectivity. Garg, Santosh and Singla [9] subsequently improve the approximation factors, using a similar approach, to $\frac{3}{2}$ and $\frac{5}{4}$, respectively. None of these methods appear to extend directly to the minimum SCSS problem.

Undirected graphs having bounded cycle length have bounded tree width. Arnborg, Lagergren and Seese [3] have shown that many NP-hard problems, including the minimum biconnected-spanning-subgraph problem, have polynomial-time algorithms when restricted to such graphs.

2. Preliminaries. To *contract* a pair of vertices u, v of a digraph is to replace u and v (and each occurrence of u or v in any edge) by a single new vertex, and to delete any subsequent self-loops and multi-edges. Each edge in the resulting graph is identified with the corresponding edge in the original graph or, in the case of multi-edges, the single remaining edge is identified with any one of the corresponding edges in the original graph. To contract an edge (u, v) is to contract the pair of vertices u and v. To contract a set S of pairs of vertices in a graph G is to contract the pairs in S in arbitrary order. The contracted graph is denoted by G/S. Contracting an edge is also analogously extended to contracting a set of edges.

Let $\mathcal{OPT}(G)$ be the minimum size of any subset of the edges that strongly connects G. In general, the term "cycle" refers only to simple cycles.

3. Lower Bounds on $\mathcal{OPT}(G)$. We begin by showing that if a graph has no long cycles, then the size of any SCSS is large.

LEMMA 3.1 (CYCLE LEMMA). *For any directed graph G with n vertices, if a longest cycle of G has length $\mathcal{C}$, then*

$$\mathcal{OPT}(G) \geq \frac{\mathcal{C}}{\mathcal{C}-1}(n-1).$$

Proof. Starting with a minimum-size subset that strongly connects the graph, repeatedly contract cycles in the subset until no cycles are left. Observe that the maximum cycle length does not increase under contractions. Consequently, for each cycle contracted, the ratio of the number of edges contracted to the decrease in the number of vertices is at least $\frac{\mathcal{C}}{\mathcal{C}-1}$. Since the total decrease in the number of vertices is $n-1$, at least $\frac{\mathcal{C}}{\mathcal{C}-1}(n-1)$ edges are contracted. □

Note that the above lemma gives a lower bound which is existentially tight. For all values of $\mathcal{C}$, there exist graphs for which the bound given by the lemma is equal to $\mathcal{OPT}(G)$. Also note that $\mathcal{C}$ has a trivial upper bound of n and, using this, we get a lower bound of n for $\mathcal{OPT}(G)$, which is the known trivial lower bound.

LEMMA 3.2 (CONTRACTION LEMMA). *For any directed graph G and set of edges S,*

$$\mathcal{OPT}(G) \geq \mathcal{OPT}(G/S).$$

Proof. Any SCSS of G, contracted around S (treating the edges of S as pairs), is an SCSS of G/S. □

4. Cycle-Contraction Algorithm. The algorithm is the following. Fix k to be any positive integer.

CONTRACT-CYCLES$_k(G)$ —

for $i = k, k-1, k-2, ..., 2$
  while the graph contains a cycle with at least i edges
    Contract the edges on such a cycle.
return the contracted edges

In Section 6, we will show that the algorithm can be implemented to run in $O(m\alpha(m,n))$ time for the case $k=3$ and in polynomial time for any fixed value of k. It is clear that the edge set returned by the algorithm strongly connects the graph. The following theorem establishes an upper bound on the number of edges returned by the algorithm.

THEOREM 4.1. *CONTRACT-CYCLES$_k(G)$ returns at most $c_k \cdot \mathcal{OPT}(G)$ edges, where*

$$\frac{\pi^2}{6} \le c_k \le \frac{\pi^2}{6} + \frac{1}{(k-1)k}.$$

Proof. Initially, let the graph have n vertices. Let n_i vertices remain in the contracted graph after contracting cycles with i or more edges ($i = k, k-1, ..., 2$).

How many edges are returned? In contracting cycles with at least k edges, at most $\frac{k}{k-1}(n-n_k)$ edges are contributed to the solution. For $i<k$, in contracting cycles with i edges, $\frac{i}{i-1}(n_{i+1}-n_i)$ edges are contributed. The number of edges returned is thus at most

$$\frac{k}{k-1}(n-n_k) + \sum_{i=2}^{k-1}\frac{i}{i-1}(n_{i+1}-n_i) \le \left(1+\frac{1}{k-1}\right)n + \sum_{i=3}^{k}\frac{n_i-1}{(i-1)(i-2)}.$$

Clearly $\mathcal{OPT}(G) \ge n$. For $2 \le i \le k$, when n_i vertices remain, no cycle has more than $i-1$ edges. By Lemmas 3.1 and 3.2, $\mathcal{OPT}(G) \ge \frac{i-1}{i-2}(n_i-1)$. Thus the number of edges returned, divided by $\mathcal{OPT}(G)$, is at most

$$\frac{\left(1+\frac{1}{k-1}\right)n}{\mathcal{OPT}(G)} + \sum_{i=3}^{k}\frac{\frac{n_i-1}{(i-1)(i-2)}}{\mathcal{OPT}(G)} \le \frac{(1+\frac{1}{k-1})n}{n} + \sum_{i=3}^{k}\frac{\frac{n_i-1}{(i-1)(i-2)}}{\frac{i-1}{i-2}(n_i-1)} = \frac{1}{k-1} + \sum_{i=1}^{k-1}\frac{1}{i^2} = c_k.$$

Using the identity (from [17, p.75]) $\sum_{i=1}^{\infty}\frac{1}{i^2} = \frac{\pi^2}{6}$, we get

$$\begin{aligned}
\frac{\pi^2}{6} &\le c_k = \frac{\pi^2}{6} + \frac{1}{k-1} - \sum_{i=k}^{\infty}\frac{1}{i^2} \\
&\le \frac{\pi^2}{6} + \frac{1}{k-1} - \sum_{i=k}^{\infty}\frac{1}{i\,(i+1)} \\
&= \frac{\pi^2}{6} + \frac{1}{k-1} - \frac{1}{k} \\
&= \frac{\pi^2}{6} + \frac{1}{(k-1)k}.
\end{aligned}$$

□

If desired, standard techniques can yield more accurate estimates of c_k, e.g., $c_k = \frac{\pi^2}{6} + \frac{1}{2k^2} + O\left(\frac{1}{k^3}\right)$. If the graph initially has no cycle longer than ℓ ($\ell \geq k$), then the analysis can be generalized to show a performance guarantee of $\frac{k^{-1}-\ell^{-1}}{1-k^{-1}} + \sum_{i=1}^{k-1} 1/i^2$. For instance, in a graph with no cycle longer than 5, the analysis bounds the performance guarantee (when $k = 5$) by 1.424.

Table 1 gives lower and upper bounds on the performance guarantee of the algorithm for small values of k and in the limit as $k \to \infty$. The lower bounds are shown in the next subsection.

k	Upper Bound	Lower Bound
3	1.750	1.750
4	1.694	1.666
5	1.674	1.625
∞	1.645	1.500

TABLE 1
Bounds on the performance guarantee

4.1. Lower Bounds on the Performance Ratio. In this section we present lower bounds on the performance ratio of CONTRACT-CYCLES$_k(G)$. The graph in Fig. 1 has $\frac{n}{2k-2}$ groups of vertices. Each group consists of a $(2k-2)$-cycle "threaded" with a k-cycle.

In the first iteration, CONTRACT-CYCLES$_k(G)$ can contract the k-cycle within each group, leaving the graph with only 2-cycles. The algorithm subsequently must contract all the remaining edges. Thus, all the $(3k-2)\frac{n}{2k-2} - 2$ edges are in the returned SCSS. The graph contains a Hamilton cycle and the optimal solution is thus n. Hence, for arbitrarily large n, $1 + \frac{k}{2k-2} - 2/n$ is a lower bound on the performance guarantee of CONTRACT-CYCLES$_k(G)$. As k approaches ∞, the lower bound tends to 1.5.

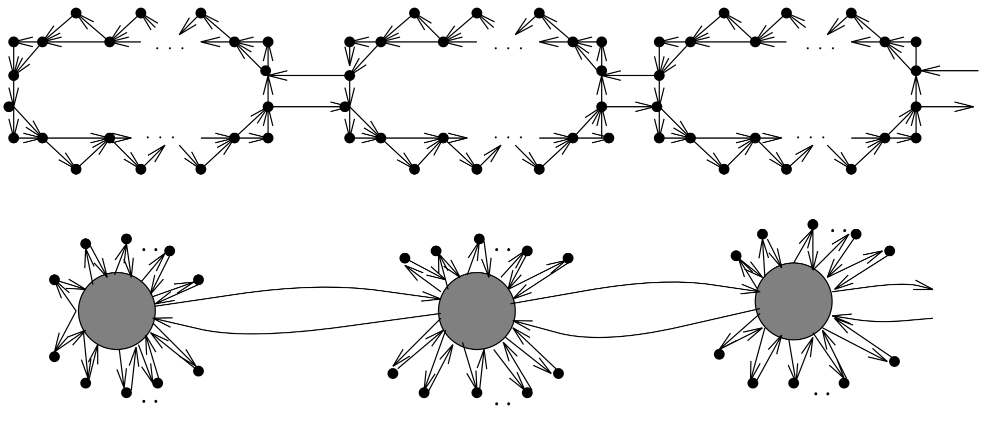

FIG. 1. *Bad example for* CONTRACT-CYCLES$_k(G)$.

5. 2-Exchange Algorithm. In this section, we use the cycle-contraction analysis to show that 2-EXCHANGE has a performance guarantee of 1.75. 2-EXCHANGE is a special case of k-EXCHANGE, which is defined as follows.

k-EXCHANGE$(G = (V, E))$ — *Local improvement algorithm*

$E' \leftarrow E$
while the following improvement step is possible
  Pick a set E_k of k edges in E' and a set E_{k-1} of up to $k-1$ edges in E such that the set of edges $E'' = (E' - E_k) \cup E_{k-1}$ forms an SCSS.
  $E' \leftarrow E''$.
return E'

Note that for fixed k, each step can be performed in polynomial time and it reduces the size of E', so k-EXCHANGE runs in polynomial time. The following theorem shows that the approximation factor achieved by 2-EXCHANGE is 1.75.

THEOREM 5.1. *The performance guarantee of* 2-EXCHANGE *is* 1.75.

Proof. We will show that the edges output by 2-EXCHANGE(G) could be output by CONTRACT-CYCLES$_3(G)$. Thus, the performance guarantee of 1.75 for CONTRACT-CYCLES$_3$ carries over to 2-EXCHANGE.

First we show that the performance guarantee is at most 1.75. Let E' be the set of edges returned by 2-EXCHANGE$(G = (V, E))$. Run CONTRACT-CYCLES$_3$ on the graph $G' = (V, E')$. Let H be the set of edges contracted during the first iteration when cycles of at least three edges are contracted. The resulting graph G'/H is strongly connected and has only 2-cycles. Such a graph has a tree-like structure. In particular, an edge (u, v) is present iff the reverse edge (v, u) is present.

The important observation is that G/H is equivalent to G'/H. Clearly G'/H is a subgraph of G/H; to prove the converse, suppose that some edge (u, v) of G/H was not in G'/H. Consider adding edge (u, v) to G'/H. By the structure of G'/H, u and v are not adjacent in G'/H and for each edge on the path from v to u the reverse edge is also in G'/H. If (u, v) is added to G'/H, these (at least two) reverse edges can be deleted from G'/H without destroying the strong connectivity of G'/H. Consequently, the original edge in G corresponding to (u, v) can be added to G' and the original edges in G' corresponding to the reverse edges can be deleted from G' without destroying the strong connectivity of G'. This contradicts the fact that E' was output by 2-EXCHANGE(G), since E' is eligible for an improvement step.

Next consider executing CONTRACT-CYCLES$_3(G)$. Since G/H is equivalent to G'/H, the sequence of cycles chosen in the first iteration of CONTRACT-CYCLES$_3(G')$ could also be chosen by the first iteration of CONTRACT-CYCLES$_3(G)$. Similarly, the second iteration in CONTRACT-CYCLES$_3(G')$ could be mimicked by CONTRACT-CYCLES$_3(G)$, in which case CONTRACT-CYCLES$_3(G)$ would return the same edge set as CONTRACT-CYCLES$_3(G')$. Since E' is minimal (otherwise an improvement step applies), the edge set returned is exactly E'. Thus, the upper bound on the performance guarantee of CONTRACT-CYCLES$_3$ from Theorem 4.1 is inherited by 2-EXCHANGE. □

For the lower bound on the performance guarantee, given the graph in Fig. 2, 2-EXCHANGE can choose a number of edges arbitrarily close to 1.75 times the minimum. There are $\frac{n}{4}$ groups with 4 vertices in each group. First observe that the graph has a directed Hamilton cycle. The edges marked in Fig. 2 form a solution that 2-EXCHANGE could terminate with. This solution clearly has $\frac{7n}{4}$ edges. This gives the lower bound of 1.75 on the performance of the algorithm.

6. Implementation. For any fixed k, CONTRACT-CYCLES$_k$ can be implemented in polynomial time using exhaustive search to find long cycles. For instance, if a cycle of size at least k exists, one can be found in polynomial time as follows. For each

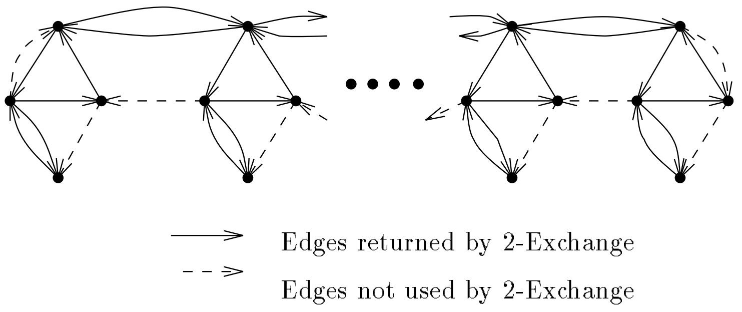

FIG. 2. *Worst-case example for* 2-EXCHANGE.

simple path P of $k-1$ edges, check whether a path from the head of P to the tail exists after P's internal vertices are removed from the graph. If k is even, there are at most $m^{k/2}$ such paths; if k is odd, the number is at most $n\, m^{(k-1)/2}$. It takes $O(m)$ time to decide if there is a path from the head of P to the tail of P. For the first iteration of the for loop, we may have $O(n)$ iterations of the while loop. Since the first iteration is the most time consuming, the algorithm can be implemented in $O(n\, m^{1+k/2})$ time for even k and $O(n^2\ m^{(k+1)/2})$ time for odd k.

6.1. A practical implementation yielding 1.75. Next we give a practical, near linear-time implementation of CONTRACT-CYCLES$_3$. The performance guarantee achieved is $c_3 = 1.75$. CONTRACT-CYCLES$_3$ consists of two phases: (1) repeatedly finding and contracting cycles of three or more edges (called *long* cycles), until no such cycles exist, and then (2) contracting the remaining 2-cycles.

High-level description of the algorithm. To perform Phase (1), the algorithm does a depth-first search (DFS) of the graph from an arbitrary root. During the search, the algorithm identifies edges for contraction by adding them to a set S. At any point in the search, G' denotes the subgraph of edges and vertices traversed so far. The rule for adding edges to S is as follows: when a new edge is traversed, if the new edge creates a long cycle in G'/S, the algorithm adds the edges of the cycle to S. The algorithm thus maintains that G'/S has no long cycles. When the DFS finishes, G'/S has only 2-cycles. The edges on these 2-cycles, together with S, are the desired SCSS.

Because G'/S has no long cycles and the fact that the original graph is strongly connected, G'/S maintains a simple structure:

LEMMA 6.1. *After the addition of any edge to G' and the possible contraction of a cycle by adding it to S: (i) The graph G'/S consists of an outward branching and some of its reverse edges. (ii) The only reverse edges that might not be present are those on the "active" path: from the super-vertex containing the root to the super-vertex in G'/S containing the current vertex of the DFS.*

Proof. Clearly the invariant is initially true. We show that each given step of the algorithm maintains the invariant. In each case, if u and w denote vertices in the graph, then let U and W denote the vertices in G'/S containing u and w, respectively.

When the DFS traverses an edge (u, w) to visit a new vertex w:

Vertex w and edge (u, w) are added to G'. Vertex w becomes the current vertex. In G'/S, the outward branching is extended to the new vertex W by the addition of edge (U, W). No other edge is added, and no cycle is created. Thus, part (i) of the invariant is maintained. The super-vertex containing the current vertex is now W, and the new "active path" contains the old "active path". Thus, part (ii) of the invariant is also maintained.

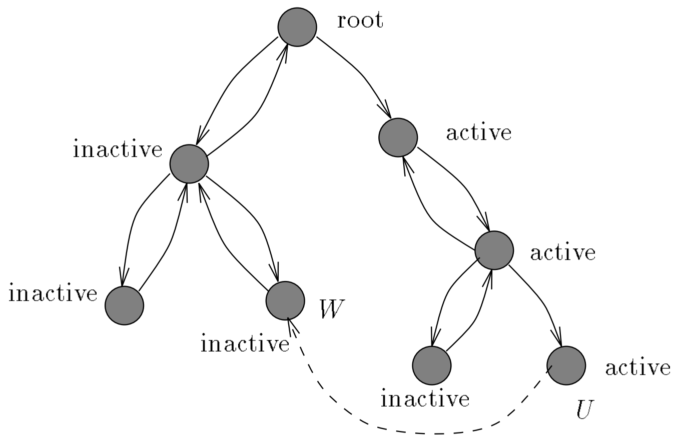

FIG. 3. *Contracted graph G'/S.*

When the DFS traverses an edge (u, w) and w is already visited:

If $U = W$ or the edge (U, W) already exists in G'/S, then no cycle is created, G'/S is unchanged, and the invariant is clearly maintained. Otherwise, the edge (u, w) is added to G' and a cycle with the simple structure illustrated in Fig. 3 is created in G'/S. The cycle consists of the edge (U, W), followed by the (possibly empty) path of reverse edges from W to the lowest-common-ancestor (lca) of U and W, followed by the (possibly empty) path of branching edges from $\mathrm{lca}(U, W)$ to U. Addition of (U, W) to G'/S and contraction of this cycle (in case it is a long cycle) maintains part (i) of the invariant. If the "active path" is changed, it is only because part of it is contracted, so part (ii) of the invariant is maintained.

When the DFS finishes visiting a vertex w:

No edge is added and no cycle is contracted, so part (i) is clearly maintained. Let u be the new current vertex, i.e., w's parent in the DFS tree. If $U = W$, then part (ii) is clearly maintained. Otherwise, consider the set D of descendants of w in the DFS tree. Since the original graph is strongly connected, some edge (x, y) in the original graph goes from the set D to its complement $V - D$. All vertices in D have been visited, so (x, y) is in G'. By part (i) of the invariant, the vertex in G'/S containing x must be W, while the vertex in G'/S containing y must be U. Otherwise the edge corresponding to (x, y) in G'/S would create a long cycle. □

The algorithm maintains the contracted graph G'/S using a union-find data structure [22] to represent the vertices in the standard way and using three data structures to maintain the branching, the reverse edges discovered so far, and the "active path". When a cycle arises in G'/S, it must be of the form described in the proof of Lemma 6.1 and illustrated in Fig. 3. Using these data structures, the algorithm discovers it and, if it is long, contracts it in a number of union-find operations proportional to the length of the cycle. This yields an $O(m\alpha(m, n))$-time algorithm.

The vertices of G'/S are represented in union-find sets as follows:

MAKE-SET(v)**:** Adds the set $\{v\}$ corresponding to the new vertex of G'/S.

FIND(v)**:** Returns the set in G'/S that contains vertex v.

UNION(u, v)**:** Joins into a single set the two sets corresponding to the vertices in G'/S containing G''s vertices u and v.

The data structures representing the branching, reverse edges, and the active paths, respectively are:

from-root$[W]$**:** For each branching edge (U, W) in G'/S, from-root$[W] = (u, w)$ for some $(u, w) \in (U \times W) \cap E$.

to-root$[U]$**:** For each reverse edge (U, W) in G'/S, to-root$[U] = (u, w)$ for some $(u, w) \in (U \times W) \cap E$.

to-active$[U]$**:** For each vertex U on the "active path" in G'/S, to-active$[U] = (u, w)$ where $(u, w) \in (U \times W) \cap E$ and W is the child of U for which the recursive DFS call is currently executing, unless no recursive DFS is executing, in which case to-active$[U] =$ **current**.

For all other vertices, to-active$[U] =$ **nil**.

Pseudo-code for the algorithm is given in Figures 4 and 5.

Contract-Cycles$_3(G = (V, E))$ — *Pseudo-code.*
$S \leftarrow \{\}$
Choose $r \in V$.
DFS(r)
Add 2-cycles remaining in G'/S to S.
return S

DFS(u) —
to-active[Find(u)] $\leftarrow$ **current**
for each vertex w adjacent to u — *traverse edge* (u, w) —
  if (w is not yet visited) — *new vertex* —
    Make-Set(w)
    to-active[Find(u)] $\leftarrow$ from-root[Find(w)] $\leftarrow (u, w)$
    DFS(w)
    to-active[Find(u)] $\leftarrow$ **current**
  else — *edge creates cycle in* G'/S —
      if (Find$(u) \neq$ Find(w)) — *cycle length at least 2* —
        $(x, y) \leftarrow$ from-root[Find(u)]
        if (Find$(x) =$ Find(w)) — *length two cycle through parent,* $U - W - U$ —
          to-root[Find(u)] $\leftarrow (u, w)$ — *record edge to parent* —
        else
          $(x, y) \leftarrow$ from-root[Find(w)]
          if (Find$(x) \neq$ Find(u)) — *not a forward edge to child; length of cycle* ≥ 3 —
            Contract-Cycle(w)
            $S \leftarrow S \cup \{(u, w)\}$
to-active[Find(u)] $\leftarrow$ **nil**

Fig. 4. *Practical implementation of* Contract-Cycles$_3$

By the preceding discussion, the algorithm implements Contract-Cycles$_3$. It is straightforward to show that it runs in $O(m\alpha(m, n))$ time. Hence, we have the following theorem.

Theorem 6.2. *There is an* $O(m\alpha(m, n))$*-time approximation algorithm for the minimum SCSS problem achieving a performance guarantee of* 1.75 *on an* m*-edge,* n*-vertex graph.* Here $\alpha(m, n)$ is the inverse-Ackermann function associated with the union-find data structure [22].

Example to Illustrate Algorithm. In the example in Fig. 6, the algorithm begins the DFS from vertex 1. It visits vertices 2,3,4 and then traverses the reverse edge $(4, 2)$.

```
CONTRACT-CYCLE(w) —
while (to-active[FIND(w)] ≠ current) do
    if (to-active[FIND(w)] = nil) then    — Go up towards l. c. a. along reverse edges. —
      (c, p) ← to-root[FIND(w)]
      a ← to-active[FIND(p)]
    else    — Go down from l. c. a. along active path. —
        (p, c) ← to-active[FIND(w)]
        a ← to-active[FIND(c)]
      — Contract parent p and child c. —
    f ← from-root[FIND(p)]
    t ← to-root[FIND(p)]
    UNION(p, c)
    to-active[FIND(w)] ← a
    from-root[FIND(w)] ← f
    to-root[FIND(w)] ← t
```

FIG. 5. *Subroutine* CONTRACT-CYCLE

Since this edge creates a 3-cycle $(2,3),(3,4),(4,2)$ in G'/S it contracts the cycle. Next it traverses the reverse edge $(3,1)$, but does not contract it since it forms only a 2-cycle in the contracted graph. Continuing the DFS, it visits vertices 5 and 6. When it traverses the edge $(6,4)$, it discovers and contracts the cycle $(3,1),(1,5),(5,6),(6,4)$. Next it visits vertices 7 and 8, traversing the reverse edges $(8,7)$ and $(7,6)$. Traversing the edge $(6,8)$, it discovers and contracts the 3-cycle $(8,7),(7,6),(6,8)$. In this example, no 2-cycles remain, so it returns just the contracted edges.

7. Potential Improvement of CONTRACT-CYCLES$_k$**.** A natural modification to CONTRACT-CYCLES$_k$ would be to stop when the contracted graph has no cycles of length more than some c and somehow solve the remaining problem optimally.

For instance, for $c = 3$, by following the proof of Theorem 4.1, one can show that this would improve the performance guarantee of CONTRACT-CYCLES$_k$ to $c_k - 1/36$ (for $k \geq 4$), matching the lower bound in Table 1. (The lower bound given holds for the modified algorithm.)

This leads us to consider the minimum SCSS$_c$ problem — the minimum SCSS problem restricted to graphs with cycle length bounded by c. The following theorem is shown in [14].

THEOREM 7.1. *There is a polynomial-time algorithm for the SCSS$_3$ problem.*

We make no conjecture concerning the SCCS$_4$ problem. However, we next show that the SCCS$_5$ problem is NP-hard, and that the SCSS$_{17}$ problem is MAX SNP-hard.

7.1. NP-hardness of SCSS$_5$. We prove the following theorem.

THEOREM 7.2. *The minimum SCSS$_5$ problem is NP-hard.*

Proof. The proof is by a reduction from SAT [8]. Fix an arbitrary CNF formula F. We will build a rooted digraph such that any SCSS contains all the edges out of the root (d of them) and such that F is satisfiable iff there exists an SCSS E' in which each of the remaining $n-1$ non-root vertices has out-degree equal to one. Thus the formula will be satisfiable if and only if there is an SCSS with $n-1+d$ edges.

The graph has a fixed root vertex r and a vertex for each clause in F (these vertices are not shown in Fig. 7). Each clause vertex has a return edge to the root. For each variable in F, the graph has an instance of the gadget illustrated in Fig. 7. The edges into the gadget come from the root. Each such edge is present in any

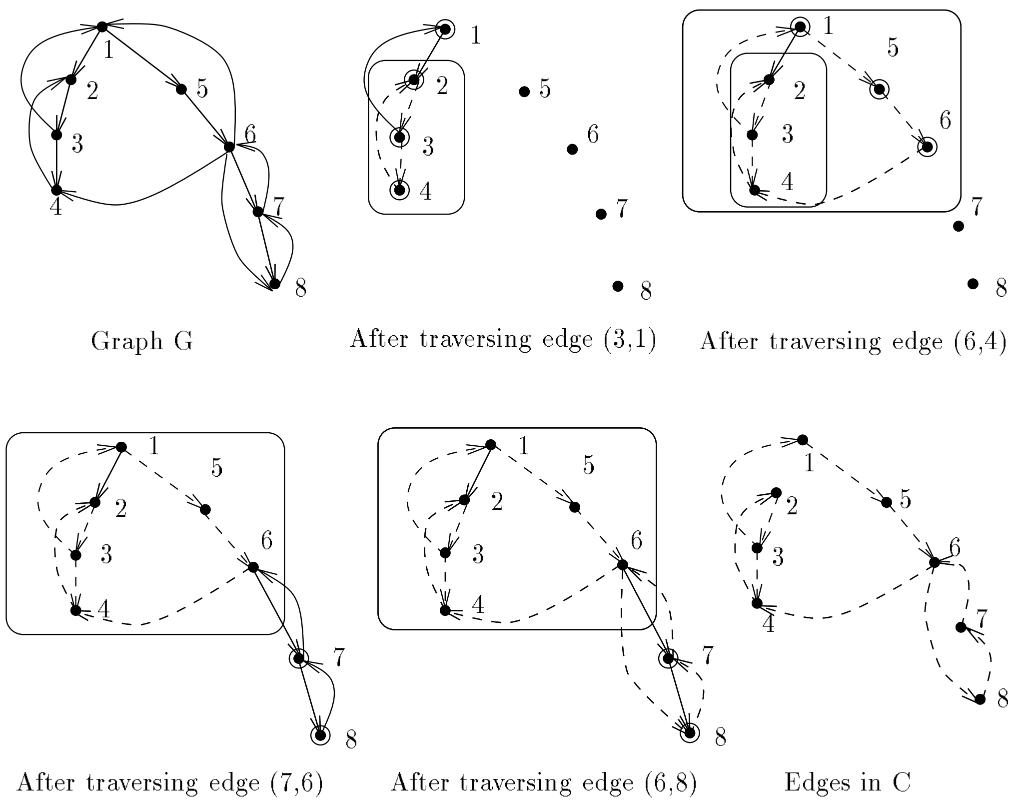

FIG. 6. *Example to illustrate execution of algorithm.*

SCSS. The edges out of the gadget are alternately labeled + and −. For every clause with a positive instance of the variable, one of the + edges goes to the clause vertex. For every clause with a negative instance of the variable, one of the − edges goes to the clause vertex. Unassigned + and − edges go to the root. (The gadget is easily enlarged to allow any number of occurrences.)

The key property of the gadget is that if every (non-root) vertex has out-degree one in some SCSS, then either all of the counter-clockwise edges are in the SCSS (corresponding to the variable being true) or all of the clockwise edges are in the SCSS (corresponding to the variable being false). Thus, given any SCSS of $d+n-1$ edges, where d is the outdegree of the root and n is the number of vertices in the digraph constructed, it is easy to construct a satisfying assignment for F. Conversely, given any satisfying assignment for F, is is easy to construct an SCSS of size $d+n-1$. ∎

7.2. MAX SNP-hardness of SCSS$_{17}$. Next we consider the MAX SNP-hardness of the problem. To prove this we do a reduction from the vertex-cover problem in bounded-degree graphs to the SCSS problem. Since the proof closely follows the reduction from vertex cover to Hamiltonian circuits (see [8]), it is suggested that the reader study this reduction before reading this subsection. It is known that the problem of finding a minimum vertex cover is MAX SNP-hard in graphs whose maximum degree is bounded by seven [19].

Let G be a connected, undirected graph whose maximum degree is bounded by

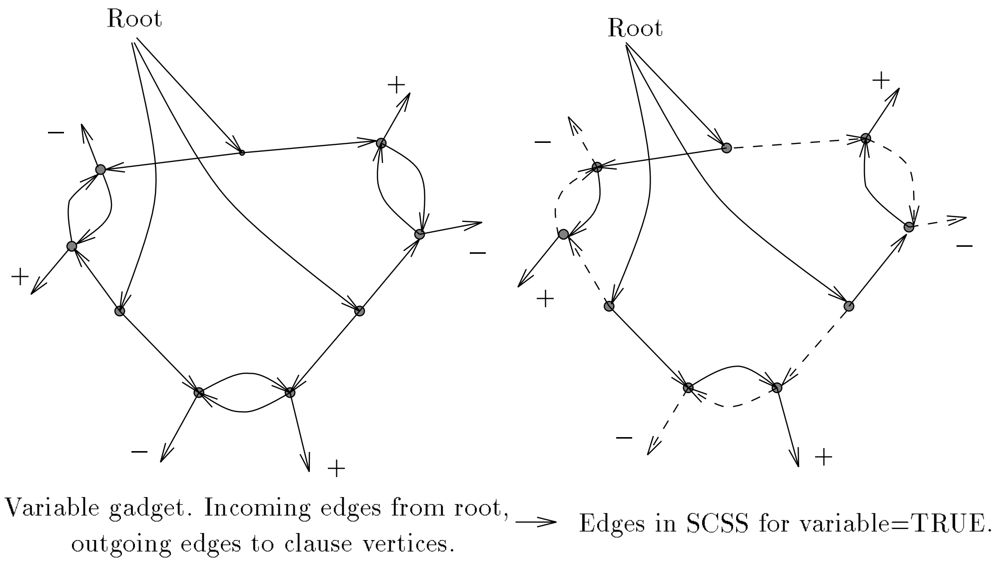

FIG. 7. *Variable gadget for NP-hardness proof.*

seven. Let G have m edges and n vertices. We construct a digraph D with $2m+1$ vertices and no cycle longer than 17. Any vertex cover of G of size s will yield an SCSS in D of size $2m+s$, and vice versa. We then show that, since G has $O(n)$ edges, this yields an L-reduction (i.e., an approximation-preserving reduction [19]).

7.2.1. The construction of D. Applying Vizing's theorem [23], color the edges of G in polynomial time with at most eight colors so that no two edges incident to a vertex share the same color. Let the colors of the edges be chosen from the set $\{1, 2, ..., 8\}$.

The construction begins with a special "root" vertex r in D.

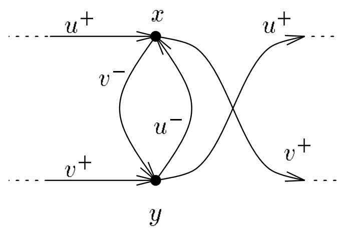

FIG. 8. *A cover-testing component.*

As the construction proceeds, each vertex in G will have a "current vertex" in D, initially the root vertex. We process edges in each color class starting with color 1. For each edge (u, v), add a "cover-testing gadget" to D, as illustrated in Fig. 8. Specifically, add two new vertices x and y. Add two edges into x: the first, labeled u^+, from the current vertex of u; the second, labeled u^-, from y. Similarly, add two edges into y: the first, labeled v^+, from the current vertex of v; the second, labeled v^-, from x. Make y the new current vertex of u; make x the new current vertex of v. Finally, after all edges of G have been considered, for each vertex v in G, add an edge labeled v^+ from its final current vertex to the root. The gadgets are implicitly layered, with each gadget being assigned to a layer corresponding to the color of the associated edge in G.

LEMMA 7.3. *The graph D constructed above has no cycle with more than 17 edges.*

Proof. We first assign numbers to the vertices of D. The root r is assigned the number 0. The construction above proceeds in order of increasing color of the edges of G. When considering an edge (u, v) of color c, we add two new vertices: x is added to v's path and y is added to u's path. We assign the vertices x and y the number c. Consider any cycle X of length greater than two in D. It is clear that such a cycle must pass through r, since D is layered. Hence the cycle is of the form $(r, x_1, x_2, \ldots, x_k, r)$. Because we considered the edges in order of increasing color, the numbers assigned to the vertices in X increases at least every two steps in any path in D (not including r). In other words, the numbers assigned to the vertices $x_1, \ldots, x_k$ forms a non-decreasing sequence in which no three consecutive vertices get the same number. Since the edges of G were colored with 8 colors, the numbers assigned to the vertices of D range from 0 to 8 (only r gets the number 0). Combining all these, the length of the cycle X is at most 17. □

7.2.2. The analysis. We now show that every vertex cover of G has a corresponding SCSS in D. The proof is similar to the corresponding proof (in the reduction from vertex cover to Hamiltonian circuits) that every vertex cover has a corresponding Hamiltonian circuit. Consider an arbitrary vertex cover S of G. The idea is to choose in the SCSS the paths corresponding to S in D. The paths of the vertices of $V - S$ are yet to be connected. Since S forms a vertex cover, the vertices in the paths of $V - S$ can be connected using the cover-testing components.

LEMMA 7.4. *Given a vertex cover of size s in G, an SCSS of D of size $2m + s$ can be constructed.*

Proof. Construct a subgraph H of D as follows. For each vertex u in G, let d_u be the degree of u in G. If u is in the vertex cover, add the $d_u + 1$ edges labeled u^+ in D to H. Otherwise, add the d_u edges labeled u^- in D to H. It is easy to verify that H has the following properties:

1. H has $2m + s$ edges.
2. H has no cycles of length 2.
3. Every vertex of H has at least one outgoing and at least one incoming edge.

As mentioned earlier, D is layered and every cycle of length greater than 2 contains r. Therefore Property 2 above implies that every cycle of H passes through r. By the above conditions, H contains a path from r to every vertex v and another path from v to r, and is therefore strongly connected. To obtain a path from any v to r, start from v and keep traversing an outgoing edge (which exists by Property 3) from the current vertex. Such a path must eventually reach r because r is contained in every cycle of H. Hence H satisfies the lemma. □

We now show that every SCSS of D corresponds to a vertex cover of G. The proof works by showing that any SCSS can be converted into a "canonical" SCSS whose size is no larger, that corresponds to a vertex cover of G.

LEMMA 7.5. *Given an SCSS in D of size $2m + s$, a vertex cover of G of size s can be constructed.*

Proof. As long as some non-root vertex y has both of its incoming edges in the SCSS, modify the SCSS as follows. Let (x, y) be the edge labeled v^- for some v. Remove the edge (x, y) and add the other edge out of x, if it is not already present.

Alternatively, if some non-root vertex x has both of its outgoing edges in the SCSS, remove the edge (x, y) and add the other edge into y. Repeat either modification as long as applicable.

By the layering of D, each modification maintains the strong connectivity of the SCSS. Clearly none of the modifications increases the size. Each step reduces the number of edges labeled u^- for some u in the SCSS, so after at most $2m$ steps, neither modification applies, and in the resulting SCSS every non-root vertex has exactly one incoming edge and one outgoing edge in the SCSS.

An easy induction on the layering shows that for any vertex v in G, either all of the edges labeled v^+ in D are in the SCSS or none are, in which case all of the edges labeled v^- are in the SCSS. Let C be the set of vertices in G of the former kind. It is easy to show that the size of the SCSS is $2m + |C|$, so that $|C| \leq s$. For every edge (u, v) in G, the form of the gadget ensures that at least one of the two endpoints is in C. Hence, C is the desired cover. □

THEOREM 7.6. *The minimum $SCSS_{17}$ problem is MAX SNP-hard.*

Proof. Let G be an arbitrary undirected graph G whose maximum degree is bounded by seven. Let G have m edges and n vertices. Construct the digraph D as shown earlier. By Lemma 7.3, D has no cycles greater than 17. By Lemma 7.4, any vertex cover in G of size s can be used to obtain an SCSS of D of size $2m + s$. Conversely, by Lemma 7.5, an SCSS of D of size $2m + s$ can be used to obtain a vertex cover of G of size s. Since the degree of G is bounded, $m = O(n) = O(s)$ and it is easily verified that this yields an L-reduction from degree-bounded vertex cover to the minimum SCSS_{17} problem. □

8. Open Problems.

An obvious problem is to further characterize the various complexities of the minimum SCSS_k problems.

The most interesting open problem is to obtain a performance guarantee that is less than 2 for the weighted strong connectivity problem (as mentioned earlier, the performance factor of 2 is due to Frederickson and JáJá [6]). Such an algorithm may have implications for the weighted 2-connectivity problem [15] in undirected graphs as well.

The performance guarantee of k-EXCHANGE probably improves as k increases. Proving this would be interesting — similar "local improvement" algorithms are applicable to a wide variety of problems.

Acknowledgements: We would like to thank the referees for useful comments.

REFERENCES

[1] A. AGRAWAL, P. KLEIN AND R. RAVI, *When trees collide: An approximation algorithm for the generalized Steiner problem on networks*, Proc. 23rd ACM Symposium on Theory of Computing, pp. 134–144, (1991).

[2] A. V. AHO, M. R. GAREY AND J. D. ULLMAN, *The transitive reduction of a directed graph*, SIAM Journal on Computing, 1 (2), pp. 131–137, (1972).

[3] S. ARNBORG, J. LAGERGREN AND D. SEESE, *Easy problems for tree-decomposable graphs*, Journal of Algorithms, 12 (2), pp. 308–340, (1991).

[4] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN AND M. SZEGEDY, *Proof verification and hardness of approximation problems*, Proc. 33rd IEEE Symposium on Foundations of Computer Science, pp. 14–23, (1992).

[5] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, The MIT Press, 1989.

[6] G. N. FREDERICKSON AND J. JÁJÁ, *Approximation algorithms for several graph augmentation problems*, SIAM Journal on Computing, 10 (2), pp. 270–283, (1981).

[7] H. N. Gabow, Z. Galil, T. Spencer and R. E. Tarjan, *Efficient algorithms for finding minimum spanning trees in undirected and directed graphs*, Combinatorica, 6 (2), pp. 109–122, (1986).

[8] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[9] N. Garg, V. Santosh and A. Singla, *Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques*, Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 103–111, (1993).

[10] P. Gibbons, R. M. Karp, V. Ramachandran, D. Soroker and R. E. Tarjan, *Transitive compaction in parallel via branchings*, Journal of Algorithms, 12 (1), pp. 110–125, (1991).

[11] M. Goemans and D. Williamson, *A general approximation technique for constrained forest problems*, Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 307–316, (1992).

[12] F. Harary, R. Z. Norman and D. Cartwright, *Structural models: An introduction to the theory of directed graphs*, Wiley, New York, 1965.

[13] H. T. Hsu, *An algorithm for finding a minimal equivalent graph of a digraph*, Journal of the ACM, 22 (1), pp. 11–16, (1975).

[14] S. Khuller, B. Raghavachari and N. Young, *On strongly connected digraphs with bounded cycle length*, UMIACS-TR-94-10/CS-TR-3212, (1994).

[15] S. Khuller and U. Vishkin, *Biconnectivity approximations and graph carvings*, Proc. 24th ACM Symposium on Theory of Computing, pp. 759–770, (1992). Also, to appear in Journal of the ACM.

[16] P. N. Klein and R. Ravi, *When cycles collapse: A general approximation technique for constrained two-connectivity problems*, Proc. 3rd Integer Programming and Combinatorial Optimization Conference, pp. 39–56, (1993).

[17] D. E. Knuth, *Fundamental Algorithms*, Addison-Wesley, Menlo Park, CA, 1973.

[18] D. M. Moyles and G. L. Thompson, *An algorithm for finding the minimum equivalent graph of a digraph*, Journal of the ACM, 16 (3), pp. 455–460, (1969).

[19] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, Journal of Computer and Systems Sciences, 43 (3), pp. 425–440, (1991).

[20] S. Sahni, *Computationally related problems*, SIAM Journal on Computing, 3, pp. 262–279, (1974).

[21] K. Simon, *Finding a minimal transitive reduction in a strongly connected digraph within linear time*, Proc. 15th International Workshop WG'89, LNCS 411, *Springer Verlag*, pp. 245–259, (1989).

[22] R. E. Tarjan, *Data structures and network algorithms*, Society for Industrial and Applied Mathematics, 1983.

[23] V. G. Vizing, *On an estimate of the chromatic class of a P-graph (Russian)*, Diskret. Anal., 3, pp. 25–30, (1964).

[24] D. P. Williamson, M. X. Goemans, M. Mihail and V. V. Vazirani, *A primal-dual approximation algorithm for generalized Steiner network problems*, Proc. 25th ACM Symposium on Theory of Computing, pp. 708–717, (1993).