Chapter 21

Approximating the Minimum Equivalent Digraph

Samir Khuller [*] Balaji Raghavachari [†] 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." The MEG problem is NP-hard; this paper gives an approximation algorithm achieving a performance guarantee of about 1.64 in polynomial time. We give a modification that improves the performance guarantee to about 1.61. 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.

Keywords: directed graph, reachability, approximation algorithm, strong connectivity, local improvement.

*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, Box 830688, Richardson, TX 75083-0688. E-mail : rbk@utdallas.edu.

‡Computer Science Department, Princeton University, Princeton, NJ 08544. E-mail : ney@cs.princeton.edu. This work was done while at University of Maryland Institute for Advanced Computer Studies, College Park, MD 20742 and was supported by NSF grants CCR-8906949 and CCR-9111348.

1 Introduction

Connectivity is fundamental to the study of graphs and graph algorithms. Recently, many approximation algorithms for finding 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 graphs which are strongly connected, 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, approximating the MEG problem is linear-time equivalent to approximating both restricted versions.

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 and corrects some errors in the paper by Moyles and Thompson.

The related problem of finding a *transitive reduction* of a digraph — a smallest set of edges yielding the same reachability relations was studied by

Aho, Garey and Ullman [2]. Transitive reduction differs from the MEG problem in that the edges 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, Garey and Ullman 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 is at least as hard as bipartite matching; in fact we can show that it has a polynomial-time algorithm. However, potential 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 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. A linear-time algorithm finding a minimal SCSS is given by Simon [21]. A parallel algorithm is given by Gibbons, Karp, Ramachandran, Soroker and Tarjan [10].

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 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 deleted 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 deleted. □

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) —
1  for $i = k, k-1, k-2, ..., 2$
2      while the graph contains a cycle
         with at least $i$ edges
3            Contract the edges on such a cycle.
4  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*

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

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

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

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

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

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

$$\frac{\pi^2}{6} \leq c_k = \frac{\pi^2}{6} + \frac{1}{k-1} - \sum_{i=k}^{\infty} \frac{1}{i^2}$$
$$\leq \frac{\pi^2}{6} + \frac{1}{k-1} - \sum_{i=k}^{\infty} \frac{1}{i(i+1)}$$
$$= \frac{\pi^2}{6} + \frac{1}{k-1} - \frac{1}{k}$$
$$= \frac{\pi^2}{6} + \frac{1}{(k-1)k}.$$

□

Standard techniques yield more accurate estimates of c_k, e.g., $c_k = \frac{\pi^2}{6} + \frac{1}{2k^2} + O\left(\frac{1}{k^3}\right)$.

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

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.

4.1 Lower Bounds on the Performance.

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.

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($G = (V, E)$) —
1  $E' \leftarrow E$
2  while the following improvement step is possible
3        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.
4        $E' \leftarrow E''$.
5  return $E'$
```

Note that for fixed k, each step can be performed in polynomial time and it reduces the size

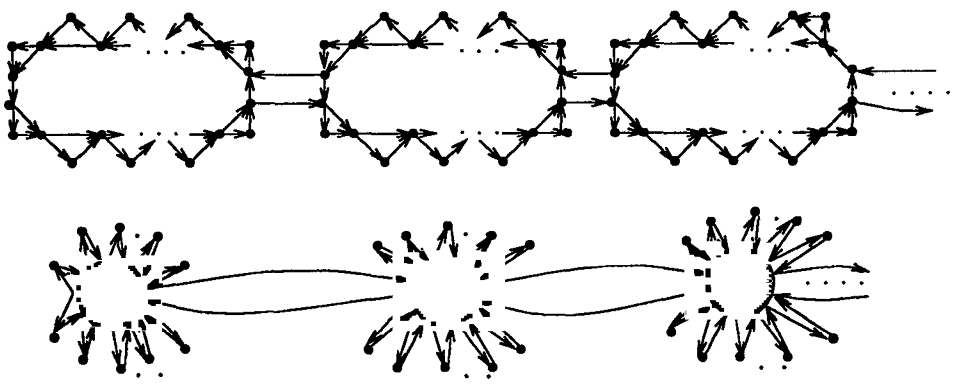

Figure 1: Bad example for CONTRACT-CYCLES$_k(G)$.

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. 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 the procedure call 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. □

6 Implementation Details

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 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

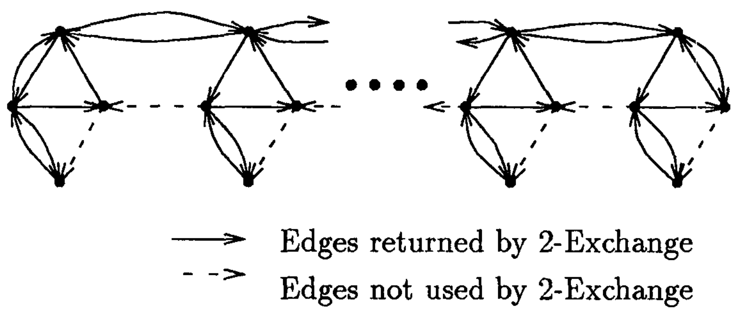

Figure 2: Worst-case example for 2-EXCHANGE.

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. 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. *The algorithm maintains the following invariant. (i) The graph G'/S consists of a branching with edges directed outwards from the super-vertex containing the root, together with some reverse edges that are of the form (W, U) such that (U, W) is in the branching. (ii) If such a reverse edge (W, U) is not present in G'/S, then the edge (U, W) is on the path in the branching from the super-vertex containing the root to the super-vertex containing the vertex currently being visited by the DFS. This path is called the "active" path.*

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.

*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', then no cycle is created, G' 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. Dark nodes represent nodes that are still active. The cycle consists of the edge (U, W), followed by the (possibly

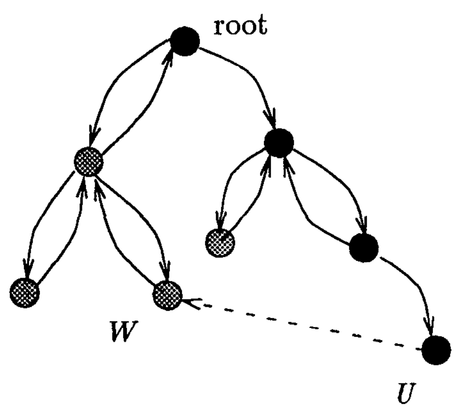

Figure 3: Contracted graph G'/S.

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 the lca(U,W) to W. 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 below.

```
CONTRACT-CYCLES3($G = (V,E)$) —
1  $S \leftarrow \{\}$
2  Choose $r \in V$.
3  DFS($r$)
4  Add 2-cycles remaining in $G'/S$ to $S$.
5  return $S$

DFS($u$) —
1  to-active[FIND($u$)] $\leftarrow$ current
2  for each vertex $w$ adjacent to $u$
      — traverse edge $(u,w)$ —
3     if ($w$ is not yet visited)
4        MAKE-SET($w$)
```

```
        to-active[FIND($u$)] $\leftarrow (u,w)$
        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 2, through parent* —
                to-root[FIND($u$)] $\leftarrow (u,w)$
            **else**
                $(x,y) \leftarrow$ from-root[FIND($w$)]
                **if** (FIND($x$) $\neq$ FIND($u$))
                    — *not 2-cycle through child* —
                    CONTRACT-CYCLE($w$)
                    $S \leftarrow S \cup \{(u,w)\}$
to-active[FIND($u$)] $\leftarrow$ **nil**
```

```
CONTRACT-CYCLE($w$) —
**while** (to-active[FIND($w$)] $\neq$ **current**) **do**
    **if** (to-active[FIND($w$)] $\neq$ **nil**) **then**
        — *Go down from lca along active path.* —
        $(x,y) \leftarrow$ to-active[FIND($w$)]
        Contract edge (FIND($x$), FIND($y$))
            of $G'/S$, updating all data structures.
    **else**
        — *Go up towards lca along reverse edges.* —
        $(x,y) \leftarrow$ to-root[FIND($w$)]
        Contract edge (FIND($x$), FIND($y$)) of
            $G'/S$, updating all data structures.
```

By the preceding discussion, the above 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 6.1. *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].

7 Graphs of bounded cycle length

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 improves 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.

7.1 Polynomially solvable cases. SCSS$_2$ problem is trivial. It is easy to show that the graph will not be strongly connected if any of its edges are deleted. SCSS$_3$ problem is much harder and has a rich structure. It is not hard to show that the SCSS$_3$ problem is as hard as bipartite matching. In fact we show that the SCSS$_3$ problem is reducible to matching and hence has a polynomial time algorithm. Due to lack of space, we omit the details.

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

COROLLARY 7.1. *The performance guarantee of the modified* CONTRACT-CYCLES$_k$ *algorithm is $c_k - 1/36$ (for $k \geq 4$), where c_k is the performance guarantee of* CONTRACT-CYCLES$_k$.

7.2 NP-hardness. We make no conjecture concerning the SCCS$_4$ problem. However, we next show that the SCCS$_5$ problem is NP-hard, and that for some $c > 0$, the SCSS$_c$ problem is SNP-hard.

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

Proof. The proof is by a reduction from SAT. We omit the proof here. The complete proof is given in the full version of the paper [14]. □

7.3 SNP-hardness. Next we consider the SNP-hardness of the problem.

THEOREM 7.3. *The minimum SCSS$_{17}$ problem is SNP-hard.*

Proof. The proof is by a reduction from the vertex cover problem. Finding a minimum vertex cover is MAX SNP-hard in graphs whose maximum degree is bounded by seven [19]. The reduction is similar to the reduction from vertex cover to Hamiltonian circuits [8].

Let G be an undirected graph G whose maximum degree is bounded by seven. Let G have m edges and n vertices. We construct a $2m+1$ vertex digraph D with root vertex r and no cycle longer than 17. Any vertex cover of G of size s will yield an SCSS of D of size $2m+s$, and vice versa. 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.

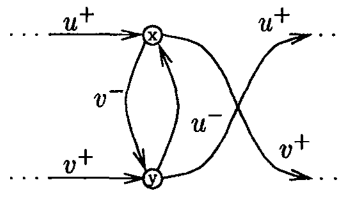

Figure 4: A cover-testing component.

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. Label the edges $\{1, 2, ..., 8\}$ corresponding to the coloring.

As the construction proceeds, each vertex in G will have a "current vertex," initially the root vertex, in D. For each edge (u, v), in order of increasing label, add a "cover-testing gadget" to D, as illustrated in Fig. 4. 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 label of the associated edge in G. Except for the 2-cycle edges and edges incident to the root in D, the edges go forward in this layering. It is easily verified that due to this layering D has no cycle with more than 17 edges.

Given a vertex cover of size s of G, construct an SCSS of D of size $2m+s$ as follows. For each vertex u in G, let d be the degree of u in G. If u is in the vertex cover, add the $d+1$ edges labeled u^+ in D to the SCSS. Otherwise, add the d edges labeled u^- in D to the SCSS. It is easy to verify that the resulting SCSS is in fact an SCSS and has $2m+s$ edges.

Conversely, given an SCSS in D of size $2m+s$, construct a vertex cover of size s as follows. First, 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. □

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.

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 Foundations of Computer Science Conference*, 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, Maintaining directed reachability with few edges, Tech. Rep. UTDCS-10-93, University of Texas at Dallas, October 1993.

[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).