In SIAM JOURNAL ON COMPUTING, 1995. Copyright SIAM

Computing Minimal Spanning Subgraphs in Linear Time

Xiaofeng Han* Pierre Kelsen† Vijaya Ramachandran‡ Robert Tarjan§

Abstract

Let P be a property of undirected graphs. We consider the following problem: given a graph G that has property P, find a minimal spanning subgraph of G with property P. We describe general algorithms for this problem and prove their correctness under fairly weak assumptions about P. We establish that the worst-case running time of these algorithms is $\Theta(m+n\log n)$ for 2-edge-connectivity and biconnectivity where n and m denote the number of vertices and edges, respectively, in the input graph. By refining the basic algorithms we obtain the first linear time algorithms for computing a minimal 2-edge-connected spanning subgraph and for computing a minimal biconnected spanning subgraph.

We also devise general algorithms for computing a minimal spanning subgraph in directed graphs. These algorithms allow us to simplify an earlier algorithm of Gibbons, Karp, Ramachandran, Soroker and Tarjan for computing a minimal strongly connected spanning subgraph. We also provide the first tight analysis of the latter algorithm, showing that its worst-case time complexity is $\Theta(m+n\log n)$.

*Current affiliation: Zyga Corporation, 28 West Oak Street, Baskingridge, NJ 07920. Research done while he was a graduate student at the Department of Computer Science, Princeton University, Princeton, NJ 08544.

†Department of Computer Science, University of British Columbia, Vancouver, B.C. CANADA V6N 1Z4. Research done while he was a graduate student at the Department of Computer Sciences, University of Texas, Austin, TX 78712 and supported in part by NSF grant CCR-89-10707.

‡Department of Computer Sciences, University of Texas, Austin, TX 78712; supported in part by NSF grant CCR-89-10707.

§Department of Computer Science, Princeton University, Princeton, NJ 08544 and NEC Research Institute, 4 Independence Way, Princeton, NJ 08540. Research at Princeton University partially supported by the National Science Foundation, Grant No. CCR-8920505, the Office of Naval Research, Contract No. N00014-91-J-1463, and by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center, Grant No. NSF-STC88-09648.

Running head: Computing Minimal Spanning Subgraphs .

Key words: minimal subgraph, biconnectivity, two-edge-connectivity, strong connectivity, linear-time algorithm, worst case behavior.

AMS subject classifications: 05C85, 05C40, 68Q25, 68R10.

Send all correspondence to:

Pierre Kelsen
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver, B.C.
CANADA V6T 1Z4
E-mail: kelsen@cs.ubc.ca

1 Introduction

Let P be a monotone graph property. In this paper we consider the following problem: given a graph G having property P, find a minimal spanning subgraph of G with property P, i.e., a spanning subgraph of G with property P in which the deletion of any edge destroys the property. We are interested in the sequential and parallel complexity of this problem.

The corresponding problem of finding a *minimum* spanning subgraph having a given property has been widely studied. We mention two results: Chung and Graham ([3], [7]) proved that the problems of finding a minimum k-vertex-connected or k-edge-connected spanning subgraph are NP-hard for any fixed $k \geq 2$. For the more relaxed problem of finding sparse but not necessarily minimal k-edge-connected and k-vertex-connected spanning subgraphs, linear time algorithms are known ([18]). Yannakakis ([24]; see also [15]) showed that the related problem of deleting a minimum set of edges so that the resulting graph has a given property is NP-hard for several graph properties (e.g., planar, outerplanar, transitive digraph).

There is a natural sequential algorithm for finding a minimal spanning subgraph with property P: examine the edges of G one at a time; remove an edge if the resulting graph has property P. This gives a polynomial time algorithm for the problem if the property P can be verified in polynomial time. However, for most nontrivial properties the running time of the algorithm is at least quadratic in the input size. Further, this algorithm seems hard to parallelize. Our goal is to obtain efficient sequential algorithms that can be parallelized effectively.

The problem at hand may be phrased in the very general framework of *independence systems* described by Karp, Upfal, and Wigderson ([12]): an *independence system* is a finite set together with a collection of subsets, called *independent sets*, with the property that any subset of an independent set is independent. Define a subset S of edges in G to be independent if the graph $G - S$ has property P. Finding a minimal spanning subgraph with property P amounts to finding a maximal independent set in the independence system that we have just defined. Efficient (deterministic) parallel algorithms for finding a maximal independent set in an independence system are known for the special case where the size of a minimal dependent set is 2 or 3 ([16], [9], [5]). For the problems that are of interest to us minimal dependent sets may have nonconstant size and hence a different approach is needed for obtaining fast parallel algorithms.

The minimal spanning subgraph problem has been studied earlier for the property of strong connectivity (*transitive compaction* problem [8]) and for 2-edge-connectivity and biconnectivity

([13]). For these problems algorithms are given in ([8], [13]) that run in $O(m + n \log n)$ sequential time and can be implemented as NC algorithms; here n and m represent the number of vertices and edges in the input graph. Both papers have a similar high-level algorithm that is shown to terminate in $O(\log n)$ stages for the properties considered, and both papers leave open the question of whether this bound is tight.

In this paper we generalize the high-level algorithm of ([8],[13]) to a large class of graph properties. We show that for any graph property that implies 2-edge-connectivity the running time of these algorithms is within a logarithmic factor of the time required for minimally augmenting a spanning tree to achieve the given property. Because various computations on trees can be performed efficiently, both sequentially ([22]) and in parallel ([17], [20]), this algorithm provides a useful paradigm for the sequential and parallel determination of minimal spanning subgraphs with respect to connectivity properties.

We analyze the worst-case complexity of these algorithms. We show that the algorithms for 2-edge-connectivity and biconnectivity require $\Omega(\log n)$ iterations in the worst-case; this implies that the worst-case time of these algorithms is $\Theta(m + n \log n)$, thus settling open questions posed in [13].

We describe refinements of the basic algorithms for 2-edge-connectivity and biconnectivity and obtain the first linear time algorithms for these properties. These algorithms still need a logarithmic number of iterations but by performing certain contractions and transformations on the current graph they reduce its size by a constant factor greater than 1 in a constant number of iterations. This result also reduces the work performed by the parallel algorithms for these problems by a logarithmic factor.

We also describe general algorithms for computing a minimal spanning subgraph in directed graphs with respect to any monotone property that implies strong connectivity. For the special case of strong connectivity, we are able to simplify the algorithm of [8] for computing a minimal strongly connected spanning subgraph. We also provide the first tight analysis of the latter algorithm, showing that its worst-case running time is $\Theta(m + n \log n)$. This answers a question posed in [8].

This paper is organized as follows. The next section defines the terms from graph theory used in this paper. In section 3 we describe and analyze general algorithms for computing minimal spanning subgraphs in undirected graphs. In section 4 we describe refinements that yield linear time algorithms for computing a minimal 2-edge-connected spanning subgraph and for computing a minimal biconnected spanning subgraph. In section 5 we show that the basic algorithms have a worst-case time complexity of $\Theta(m + n \log n)$ if we do not incorporate those refinements. In section 6 we develop and analyze general algorithms for computing minimal spanning subgraphs in directed graphs. In that section we also analyze an algorithm of [8] for computing a minimal strongly connected spanning subgraph and we provide a simplified algorithm for the problem.

The sequential model of computation that we assume in this paper is the RAM ([1]) with a word length of $O(\log n)$ bits where n is the length of the input (or the number of vertices in the input graph).

2 Definitions

We introduce graph terminology similar to that of [2]. A *graph* is a triple $G = (V, E, \phi_G)$ where V is a set of *vertices*, and E, disjoint from V, is a set of *edges* (both V and E are assumed to be finite); the *incidence function* ϕ_G maps edges in E to unordered (ordered) pairs of vertices in V if the graph is undirected (directed). Note that this definition allows for the representation of multigraphs. We write $V(G)$ and $E(G)$ for the set of vertices and the set of edges, respectively, of G and use $n(G)$ and $m(G)$ to denote the number of vertices and edges, respectively, in G. We refer to a directed graph also as a *digraph*. We write (u, v) both for ordered and unordered pairs. If (u, v) is an edge in a directed graph, then u is called the *tail* of this edge and v is called the *head* of this edge. If $\phi(e) = (u, v)$, we say that edge e is *incident* on the vertices u and v and vertices u and v are the endpoints of e. The degree of a vertex v in G, denoted by $deg_G(v)$, is the number of edges incident on vertex v. If the graph is directed, then the indegree (outdegree) of a vertex v is the number of edges $e \in E(G)$ such that $\phi_G(e) = (w, v)$ $(\phi_G(e) = (v, w))$ for some vertex w.

If every edge in a graph joins two distinct vertices and no two edges join the same pair of vertices, then we say that the graph is *simple*. The *underlying graph* of a digraph G is obtained by omitting the directions of the edges in the digraph. We say that a digraph G is an *orientation* of an undirected graph G' if G' is the underlying graph of G.

The following definitions apply both to directed and undirected graphs. A *path* P in G is an alternating sequence $P =< v_0, e_1, v_1, \dots v_{k-1}, e_k, v_k >$ of vertices and edges of G such that the following holds: (1) $\phi_G(e_i) = (v_{i-1}, v_i)$ for $0 < i \le k$, (2) $v_0 \dots v_{k-1}$ are distinct and $v_1 \dots v_k$ are distinct; the integer k (number of edges) is the *length* of the path. The path P is a path *from* v_0 *to* v_k; v_0 and v_k are the *endpoints* of P and $v_1, \dots, v_{k-1}$ are the *internal vertices* of P. A *subpath* of P is a path of the form $< v_i, e_{i+1}, \dots, e_j, v_j >$ where $0 \le i \le j \le k$. The path P is a *chain* if it has length at least 2 and its nternal vertices all have degree 2 in G. The path P is a *cycle* if $v_0 = v_k$ and all edges e_i of P are distinct.

If $G = (V, E, \phi_G)$ and $G' = (V', E', \phi_{G'})$ are two graphs such that $V' \subseteq V'$, $E' \subseteq E$ and $\phi_{G'}$ is the restriction of ϕ_G to E', then G' is a *subgraph* of G; it is a *proper subgraph* if $G' \neq G$. If the graph G is understood, we may represent a subgraph H of G simply by the pair $(V(H), E(H))$. A *spanning subgraph* of G is a subgraph G' with $V(G') = V$. If $G' = (V', E')$ is a spanning subgraph of G and $E'' \subseteq E$, then $G' + E''$ denotes the subgraph $(V', E' \cup E'')$ of G and $G' - E''$ denotes the subgraph $(V', E' - E'')$ of G.

An undirected (directed) graph G is *connected* (strongly connected) if there exists a path from u to v for all $u, v \in V$. A subgraph of G is a *maximal* subgraph having a given property if it is not a proper subgraph of another subgraph of G with the same property. A *connected component* of an undirected graph G is a maximal connected subgraph of G. A *tree* in a graph G is a spanning connected subgraph of G without cycles. A *strong component* of a directed graph G is a maximal strongly connected subgraph of G.

All remaining definitions apply only to undirected graphs. A *cutedge* in G is an edge in G whose removal increases the number of connected components in G. The graph G is *2-edge-connected* if it is connected and has no cutedge or, equivalently, it is connected and every edge of G lies on a cycle. A *2-edge-connected component* of G is a maximal 2-edge-connected subgraph of G. A graph is *k-edge-connected* ($k > 0$) if $G - S$ is connected for every subset $S \subseteq E(G)$ of size less than k. A vertex v in G is a *cutpoint* if removing v together with all incident edges increases the number of connected components in G or, equivalently, there exist distinct vertices u and w in G other than v such that any path from u to w contains v. A graph G is *biconnected* if it is connected, has at least three vertices, and does not contain a cutpoint. A *block of G* is a maximal subgraph of G with the property that it is connected and has no cutpoint. A graph G is *k-vertex-connected* if G has at least $k + 1$ vertices and $G - S$ is connected for every $S \subseteq V(G)$ with $|S| < k$.

Let $G = (V, E, \phi_G)$ and let $V' \subseteq V$. The operation of *collapsing the vertices of* V' *in* G produces a graph G' defined as follows: $V(G') = (V - V') \cup \{z\}$ where z is a new vertex not in V; $E(G')$ is the subset of those edges in E that have at least one endpoint outside V'; $\phi_{G'}(e) = \phi_G(e)$ if no endpoint of e belongs to V' and $\phi_{G'}(e) = (z, v)$ if $\phi_G(e) = (u, v)$ where $u \in V'$ and $v \in V - V'$.

An *ear decomposition* ([21]) $D = [P_0, P_1, \ldots, P_{r-1}]$ of a graph G is a partition of $E(G)$ into an ordered collection of edge disjoint paths $P_0, \ldots, P_{r-1}$ such that P_0 is a cycle and the two endpoints of P_i, for $i \geq 1$, are contained in some P_j, $j < i$, and none of the internal vertices of P_i are contained in any P_j, $j < i$. The paths in D are called *ears*. Ear decomposition D is an *open ear decomposition* if no P_i with $i > 0$ is a cycle. A *trivial ear* is an ear containing a single edge.

3 Finding a Minimal Spanning Subgraph in Undirected Graphs

In this section we describe three closely related algorithms for finding a minimal spanning subgraph of a graph for various properties of undirected graphs. Algorithm 1 was first described in [14] while algorithms 2 and 3 are generalizations of algorithms for finding a minimal 2-edge-connected and a minimal biconnected spanning subgraph given in [10].

A *graph property* P is a Boolean-valued function on graphs. If $P(G)$ is true for some graph G, we say that G *has property* P or G *is a* P*-graph*. A P*-subgraph* of G is a subgraph of G that has property P. An edge e of a P-graph G is P*-redundant* in G if $G - e$ has property P, otherwise e is P*-essential* in G. We may not mention G or P if the graph or the property is clear from the context.

In this paper we concern ourselves with the problem of finding a minimal spanning P-subgraph of a P-graph G, i.e., a spanning P-subgraph of G in which every edge is P-essential. Throughout this paper we shall implicitly assume that property P is decidable, i.e., there is an algorithm that checks whether a given graph has property P. We restrict our attention to decidable properties that satisfy conditions $C1$ and $C2$ below:

($C1$) P is monotone, i.e., the addition of an edge to a P-graph results in a P-graph;

($C2$) any P-graph is connected.

As an immediate consequence of condition $C1$ we make the following basic observation.

Observation 1 *Let G be a P-graph and let H be a spanning P-subgraph of G. Any edge that is P-redundant in H is P-redundant in G.*

There is an obvious (sequential) algorithm for computing a minimal spanning P-subgraph of G: examine the edges of G one at a time; remove an edge if it is redundant in the current graph. By observation 1 the resulting subgraph is a minimal spanning P-subgraph of G.

The following algorithm is a generalization of algorithms given in [13] and [8] (for finding a minimal 2-edge-connected, a minimal biconnected, and a minimal strongly connected spanning subgraph of a graph) to graph properties satisfying $C1$ and $C2$. This algorithm has been shown to outperform the obvious algorithm on undirected graphs for 2-edge-connectivity and biconnectivity, and we believe that this is true for a number of other properties of undirected graphs. Moreover, it is inherently easier to parallelize.

Algorithm 1 Computing a minimal spanning P-subgraph of G.

Input P-graph G.

Output Minimal spanning P-subgraph H of G.

(1) $H := G$;

(2) while H has P-redundant edges, do:

(2.1) compute a spanning tree T_H in H with a maximum number of P-essential edges;

(2.2) compute a minimal subset A of edges in H such that $T_H + A$ has property P;

(2.3) $H := T_H + A$.

A spanning tree T_H of H containing a maximum number of essential edges (constructed in step 2.1) is called an *optimal tree in* H and the set A constructed in step 2.2 is called a *minimal augmentation for* T_H (in H).

Theorem 1 *Algorithm 1 computes a minimal spanning P-subgraph of G for any property P satisfying $C1$ and $C2$.*

Proof. By induction on the number of iterations of the while-loop, one shows that H, as computed in step 2.3, is always a spanning P-subgraph of G. To prove termination, consider one execution of the while-loop. Since T_H is an optimal tree in H, it does not contain all redundant edges of H. Furthermore all edges of A are essential in $T_H + A$. Therefore, the number of redundant edges in H decreases by at least one at each iteration of the while-loop. Thus algorithm 1 terminates. $\square$

By the proof of theorem 1 the number of iterations of algorithm 1 is bounded by the number of edges in the input graph G. For several graph properties much sharper bounds hold. In this paper we are primarily interested in properties P that imply 2-edge-connectivity. The following theorem shows that for these properties algorithm 1 terminates quickly. We use n to denote the number of vertices in G.

Theorem 2 *If P satisfies $C1$ and $C2$ and also implies 2-edge-connectivity, then algorithm 1 terminates after $O(\log n)$ iterations of the while-loop.*

Proof. Fix H at the beginning of an iteration of the while-loop. An *essential component* of H is a connected component of the subgraph of H with vertex set $V(H)$ whose edges are the essential edges in H. Let r denote the number of redundant edges in H and c the number of essential components of H. Fix an optimal tree T in H and an optimal augmentation A for T. The tree T contains exactly $c-1$ redundant edges of H. Furthermore, if $c > 1$, then each essential component of H is incident with at least 3 redundant edges in H and hence $c \leq 2r/3$. Since the edges of A are essential in $T+A$, less than $2r/3$ edges of $T+A$ are redundant and hence the number of redundant edges goes down by a constant factor greater than 1 in each iteration of the while-loop. The claim follows. $\square$

Corollary 1 *Algorithm 1 computes a minimal k-vertex-connected and a minimal k-edge-connected spanning subgraph in $O(\log n)$ iterations of the while-loop.* $\square$

One drawback of algorithm 1 is that the redundant edges of H need to be computed at each iteration. In general, it is not clear whether computing these edges is easier than the original problem of finding a minimal spanning $P-$subgraph. One can avoid this computation by gradually building up a set of essential edges, as shown in the following algorithm. We We do not need to assume here that P implies 2-edge-connectivity.

Algorithm 2 Computing a minimal spanning P-subgraph of G.
Input P-graph G.
Output Minimal spanning P-subgraph H of G.

(1) $H := G$; $S := \emptyset$;

(2) while $S \neq E(H)$, do:

(2.1) compute a spanning tree T_H in H with a maximum number of edges in S;

(2.2) compute a minimal subset A of edges in H such that $T_H + A$ has property P;

(2.3) $H := T_H + A$; $S := S \cup A \cup \{\text{cutedges in } H\}$;

Theorem 3 *Algorithm 2 computes a minimal spanning P-subgraph of G for any property P satisfying $C1$ and $C2$.*

Proof. By induction on the number of iterations of the while-loop, one shows that at the end of each iteration H is a spanning P-subgraph of G and the edges in S are essential in H. To prove termination, consider one execution of the while-loop. Let $e \in E(H) - S$. If e is a cutedge in $T_H + A$, then e will be added to S in step 2.3. Otherwise the optimal tree does not contain all edges of $E(H) - S$. At least one edge of $E(H) - S$ will thus be added to S in step 2.3 (as an edge of A) or discarded. In all these cases $|E(H) - S|$ decreases by at least one in this iteration. Termination follows. $\square$

Unlike algorithm 1 algorithm 2 is not guaranteed to terminate quickly if P implies 2-edge-connectivity. For instance, if P denotes 2-edge-connectivity and G is a cycle on n vertices then algorithm 2 adds exactly one edge to S in each iteration of the while-loop and thus requires n iterations.

We overcome this problem as follows. Let $S \subseteq E(H)$. An edge of H is called *S-critical* if it is one of exactly two edges connecting some connected component of $(V(H), S)$ with the set of vertices outside this component. Note that any S-critical edge is essential in H provided P implies 2-edge-connectivity. Thus, at each iteration we can add to the current set of essential edges the edges of A as well as the S-critical edges in H. The following algorithm makes use of this idea.

Algorithm 3 Computing a minimal spanning P-subgraph of G.

Input P-graph G.

Output Minimal spanning P-subgraph H of G.

(1) $H := G$; $S := \emptyset$;

(2) while $S \neq E(H)$, do:

(2.1) compute a spanning tree T_H in H with a maximum number of edges of S;

(2.2) compute a minimal $A \subseteq E(H)$ such that $T_H + A$ has property P;

(2.3) $H := T_H + A$; $B := \{S\text{-critical edges in } H\}$; $S := S \cup A \cup B$;

As before we say that T_H is an optimal tree in H and A is a minimal augmentation for T_H in H.

Theorem 4 *Algorithm 3 computes a minimal spanning P-subgraph of G for any property P satisfying $C1$ and $C2$ and implying 2-edge-connectivity.*

Proof. An induction on the iteration number shows that, at the end of each iteration of the while-loop, H is a spanning $P-$subgraph of H and all edges in S are essential in H. Thus, upon termination H is a minimal spanning P-subgraph of H. To prove termination, fix an iteration of the while-loop. The tree T_H contains a minimum number of edges in $E(H) - S$. Since H is

2-edge-connected, it has no cutedges. Thus there exists a spanning tree in H excluding a single edge of $E(H) - S$. It follows that T_H does not contain all edges of $E(H) - S$. Any edge of $E(H) - S$ that does not belong to T_H is either discarded or added to S in step 2.3. Hence, $|E(H) - S|$ is strictly decreasing. Termination follows. $\Box$

The proof of theorem 2 suggests that the number of iterations of algorithm 3 may be proportional to the number of edges in the input graph. As was the case for algorithm 1 a much sharper bound holds for properties implying 2-edge-connectivity.

Theorem 5 *Assume property P implies 2-edge-connectivity. Let C_i denote $E(H) - S$ at the beginning of iteration i of the while-loop of algorithm 3. Then, $|C_{i+1}| \leq 3|C_i|/4$ for $|C_i| > 0$. Thus algorithm 3 terminates after $O(\log n)$ iterations.*

Proof. Let H_i and S_i denote H and S at the start of iteration i of the while-loop in algorithm 3. Hence $C_i = E(H_i) - S_i$. The claim certainly holds if the number of edges of C_i in tree T_{H_i} (computed in step 2.1) is at most $3|C_i|/4$. Now assume that more than $3|C_i|/4$ edges of C_i belong to T_{H_i}. Construct H' from H_i by collapsing the vertex sets of the connected components of $(V(H_i), S_i)$ in H_i (one at a time). Let a be the number of degree 2 vertices in H'. Note that a is a lower bound on the number of edges added to B (and hence to S) in step 2.3. It suffices to show that $a \geq |C_i|/4$.

We have $n(H') > 3|C_i|/4$ (*) since we assumed that T_{H_i} contains more than $3|C_i|/4$ edges of C_i. Also $m(H') \leq |C_i|$ and therefore, by inequality (*), $m(H') < 4n(H')/3$. Hence, $\sum_{v \in V(H')} deg_{H'}(v) < 8n(H')/3$. Moreover, $\sum_{v \in V(H')} deg_{H'}(v) \geq 2a + 3(n(H') - a)$. Thus $a > n(H')/3$. With inequality (*) we get $a > |C_i|/4$. $\Box$

Theorem 6 *Assume that P satisfies $C1$ and $C2$ and also implies 2-edge-connectivity. If a minimal augmentation of a spanning tree can be computed in time $t(m,n)$, then a minimal spanning P-subgraph can be computed using algorithm 3 in time $O(t(m,n) \log n)$. In particular this holds for k-vertex- and k-edge-connectivity for $k \geq 2$.*

Proof. We claim that all steps in algorithm 3 other than the minimal augmentation step can be done in linear time and space. We compute an optimal tree T_H as follows: we compute spanning trees T_i for the connected components of $(V(H), S)$. We collapse the connected components of $(V(H), S)$ in H and let T be a spanning tree in the resulting graph. The edges in $\cup_i E(T_i) \cup E(T)$ form an optimal tree in H and can be computed in linear time and space using depth-first search. To compute the S-critical edges, we determine the connected components of $(V(H), S)$ in linear time and space using depth-first search. Thus each iteration of the while-loop requires time $O(t(m,n))$. By the previous result algorithm 3 terminates after $O(\log n)$ iterations of the while-loop. The claim follows. □

In [13] linear time algorithms for computing a minimal augmentation for 2-edge-connectivity and biconnectivity are described. By theorem 6 algorithm 3 computes a minimal 2-edge-connected and a minimal biconnected spanning subgraph in time $O((m+n)\log n)$. In fact the following stronger bound applies.

Theorem 7 *Algorithm 3 runs in time $O(m + n\log n)$ time for 2-edge-connectivity and biconnectivity.*

Proof. It suffices to show that the number of edges in the graph H is $O(n)$ after one iteration of the while-loop. We prove this by showing that the number of edges in the minimal augmentation computed in the first iteration is $O(n)$. In the remainder of this proof T_H and A denote the optimal tree and its minimal augmentation computed in the first iteration of the while-loop of algorithm 3. For 2-edge-connectivity we argue as follows: since every edge of A is essential in $T_H + A$, for every edge $e \in A$ there exists an edge e' in T_H such that e is the only edge in A with the property that e' lies on the (unique) cycle of $T_H + e$. Thus $|A| \leq m(T_H) = n - 1$. For biconnectivity a result of Plummer ([19]) states that every minimal biconnected graph (i.e., biconnected graph with no redundant edges) has $O(n)$ edges. Thus $|A| = O(n)$ for biconnectivity as well. □

Algorithms for the same problems achieving similar time bounds are presented in [13]. Those algorithms are more complicated than algorithm 3 because they require redundant edges to be computed explicitly (a fairly involved procedure using ideas from triconnectivity testing). In section 4 we shall prove that the bound given in the last theorem is tight. In the next section we describe how to modify algorithm 3 so that it runs in linear time for 2-edge-connectivity and biconnectivity.

We obtain a result similar to theorem 6 for the parallel complexity of computing a minimal spanning P-subgraph. (For a definition of the PRAM model see [11].)

Theorem 8 *Assume that P satisfies $C1$ and $C2$ and also implies 2-edge-connectivity. If a minimal augmentation of a spanning tree can be computed in time $t(m,n)$ on $p(m,n)$ PRAM processors, then a minimal spanning P-subgraph can be computed using algorithm 3 in time $O((t(m,n) + c(m,n))\log n)$ on $p(m,n)$ processors, where $c(m,n)$ is the time required by steps 2.1 and 2.3 of algorithm 3 on $p(m,n)$ processors.* □

We note that on an ARBITRARY PRAM the complexity of steps 2.1 and 2.3 is dominated by that of finding connected components, i.e., they can be performed with almost optimal speedup in time $O(\log n)$ using $O((m+n)\alpha(m,n)/\log n)$ processors ([4]), where α denotes the inverse Ackermann function. The minimal augmentation step can be done in polylogarithmic time on a linear number of processors as shown in [13] for biconnectivity and 2-edge-connectivity.

4 Linear Time Algorithms

In this section we adapt algorithm 3 to compute minimal spanning subgraphs for 2-edge-connectivity and biconnectivity in linear time. The linear time bound is achieved by combining the linear time minimal augmentation procedures given in [13] with a method for reducing the size of the current graph while preserving its 2-edge-connectivity (biconnectivity) structure.

4.1 Finding a Minimal 2-Edge-Connected Spanning Subgraph

We start with a description of two graph operations that preserve the 2-edge-connectivity structure. Let H be a graph that may not be 2-edge-connected and let $S \subseteq E(H)$. An *S-component of H* is a 2-edge-connected component of $(V(H), S)$. The operation of *shrinking an S-component of H* consists of collapsing the vertex set of this S-component in H. For the second operation define a *chain in H* to be a path in H of length at least 2 all of whose internal vertices have degree 2 in H. Note that the edges in a chain are essential in H. A chain is *maximal* if it is not a proper subgraph of another chain in H. The operation of *contracting a chain in H* consists of collapsing the set of internal nodes of the chain in H.

If graph Q is obtained from graph H by shrinking all S-components in H and contracting all maximal chains in the resulting graph, we say that Q is a *full contraction of H* with respect to S. The following algorithm is a variant of algorithm 3 in which, at the end of the while-loop, H is replaced by its full contraction. In step 2.0 we replace H by a sparse 2-edge-connected subgraph to speed up subsequent steps; this also simplifies the analysis of algorithm 3. A linear time algorithm for computing a minimal augmentation for 2-edge-connectivity (step 2.2) is given in [13]. We finally note that the intermediate graphs need not be simple even if the input graph G is simple.

Algorithm 4 Computing a minimal 2-edge-connected spanning subgraph of G.
Input 2-edge-connected graph G.
Output Minimal 2-edge-connected spanning subgraph of G.

(1) $H := G$; $S := \emptyset$;

(2) while $E(H) \not\subseteq S$, do:

(2.0) replace H by an ear decomposition of H; discard the edges in the trivial ears from H;

(2.1) compute a spanning tree T_H in H with a maximum number of edges of S;

(2.2) compute a minimal $A \subseteq E(H)$ such that $T_H + A$ is 2-edge-connected;

(2.3) $H := T_H + A$; $B := \{S\text{-critical edges in } H\}$; $S := S \cup A \cup B$;

(2.4) replace H by its full contraction with respect to $S \cap E(H)$;

(3) return graph $(V(G), S)$.

To prove the correctness of algorithm 3, we first need to establish that the two operations of shrinking S-components and contracting chains preserve the 2-edge-connectivity structure of H.

Lemma 1 *Let H be a graph that may not be 2-edge-connected and let $S \subseteq E(H)$. If H' is obtained from H by contracting a chain or shrinking an S-component, then H' is 2-edge-connected if and only if H is 2-edge-connected. Thus, a full contraction of H is 2-edge-connected if and only if H is 2-edge-connected.*

Proof. The claim for a full contraction follows with a simple inductive argument from the first claim. We now prove the first claim. Let H' be obtained from a 2-edge-connected graph H by contracting a chain P. A cycle C in H containing an edge e of $E(H')$ ($\subseteq E(H)$) either contains all edges of P or none. In the latter case C is also a cycle in H' containing e. In the former case we obtain a cycle C' in H' containing e by contracting chain P on cycle C. Now suppose that H' is obtained from 2-edge-connected graph H by collapsing vertex set $X \subseteq V(H)$ of an S-component. An edge $e \in E(H')$ lies on a cycle C in H. Let P be a maximal subpath of C containing e and not having a vertex of X as an internal vertex. The edges on P form a cycle in H' containing e. Since H' is also connected in both cases, we conclude that H' is indeed 2-edge-connected.

Now assume that H' is 2-edge-connected and obtained from H by contracting a chain or shrinking an S-component. Again we see that H is connected. Next we note that a cycle C' in H' yields a cycle C in H that includes all edges on C (and possibly other edges). Since every edge of H' lies on a cycle of H', every edge in $E(H')$ lies on a cycle in H. If $e \in E(H) - E(H')$, then either e connects two vertices in the same S-component of H or it lies on a chain in H. In the first case it lies on a cycle whose vertices belong to the S-component. In the second case some other edge on the same chain in H belongs to $E(H')$ and thus lies on a cycle of H; this cycle must also include e. We conclude that H is 2-edge-connected. $\Box$

Corollary 2 *Let H be a 2-edge-connected graph and let H' be obtained from H by contracting a chain or shrinking an S-component ($S \subseteq E(H)$). An edge $e \in E(H')$ is essential in H' if and only if it is essential in H. The same claim holds if H' is a full contraction of H.*

Proof. As before the claim for the full contraction follows by a straightforward induction from the first claim. To prove the first claim, fix $e \in E(H')$. If H' is obtained from H by shrinking an S-component, then $H' - e$ is obtained from $H - e$ by shrinking the same S-component since e does not belong to that S-component. The claim of the corollary follows with the previous lemma. If H' is obtained from H by contracting a chain, then e is essential in both H and H' if it belongs to a chain in H' (and thus belongs to a chain in H). Otherwise $H' - e$ is obtained from $H - e$ by contracting a chain and the previous lemma implies the claim of the corollary. $\square$

Theorem 9 *Algorithm 4 outputs a minimal 2-edge-connected spanning subgraph of G.*

Proof. Replacing H by its full contraction in step 2.4 does not increase $|E(H) - S|$. Exactly as in the proof of theorem 4 one argues that $|E(H) - S|$ decreases by at least 1 during one iteration of the while-loop. The termination of algorithm 4 follows.

We number the iterations of the while-loop from 0 to k (in iteration k algorithm 4 finds that $E(H) \subseteq S$). Let H_i and S_i denote the graph H and the set S at the start of the ith iteration, let T_i and A_i stand for T_H and A in iteration i for $i < k$, and let H_i' denote the graph $(V(H_i), S_k \cap E(H_i))$. Lemma 1 implies that each H_i is 2-edge-connected. It suffices to prove that H_i' is a minimal 2-edge-connected spanning subgraph of H_i for $0 \leq i \leq k$. The claim of the theorem then follows since $H_0 = G$ and H_0' is returned in step 3 of algorithm 4.

To prove the claim, we need the following fact:

(*) any edge in $E(H_i) \cap S_i$ is essential in H_i for $0 \leq i \leq k$.

We prove (*) by induction on i. Since $S_0 = \emptyset$, (*) holds for $i = 0$. Suppose it holds for $i = j$ where $j < k$. Since $E(H_{j+1}) \subseteq E(H_j)$, every edge in $E(H_{j+1}) \cap S_j$ is essential in H_j and hence essential in $T_j + A_j$. Since H_{j+1} is a full contraction of $T_j + A_j$, corollary 2 implies that these edges are essential in H_{j+1} as well. Furthermore, the edges in $E(H_{j+1}) \cap (S_{j+1} - S_j)$ are essential in $T_j + A_j$ and thus essential in H_{j+1} (by corollary 2). Claim (*) follows.

We now prove that each H_i' is a minimal 2-edge-connected spanning subgraph of H_i by induction on $k - i$. For the base case $i = k$ we first note that $H_k = H_k'$ is 2-edge-connected. With (*) it also follows that all edges in H_k are essential since $E(H_k) \cap S_k = E(H_k)$. This completes the proof of the base case.

For the induction step assume that H'_{i+1} is a minimal 2-edge-connected spanning subgraph of H_{i+1}. In particular H'_{i+1} is 2-edge-connected. By lemma 1 it suffices to show that H'_{i+1} is a full contraction of H'_i in order to establish that H'_i is 2-edge-connected. Since the edges in $E(H_i) - E(T_i + A_i)$ are redundant in H_i, (*) implies that they do not belong to S_i and hence do not belong to S_k. Thus $E(H_i) \cap S_k = E(T_i + A_i) \cap S_k$. The fact that H_{i+1} is a full contraction of $T_i + A_i$ (with respect to a subset of S_k) implies that H'_{i+1} is a full contraction of the graph $(V(H_i), E(T_i + A_i) \cap S_k)$ (with respect to the same subset of S_k). But the latter graph is simply H'_i. Thus H'_i is 2-edge-connected. Since each edge in H'_{i+1} is essential and H'_{i+1} is a full contraction of H'_i, each edge of H'_{i+1} is essential in H'_i by corollary 2. Furthermore each edge in $E(H'_i) - E(H'_{i+1})$ either belongs to a chain in $T_i + A_i$ or belongs to $S_k \cap E(T_i + A_i)$. In the former case the edge is essential in H'_i because it has an endpoint of degree 2. In the latter case it belongs either to S_i, in which case it is essential by (*), or to $S_{i+1} - S_i$, in which case it is essential in $T_i + A_i$ and hence essential in H'_i. We conclude that H'_i is a minimal 2-edge-connected spanning subgraph of H_i. □

The following technical lemma will be used in the analysis of algorithm 4.

Lemma 2 *Let F be a forest in which r nodes are marked. Let $n_i(F)$ denote the number of degree i nodes in F. If every chain in F that does not contain a marked node as an internal vertex has length at most k, then $n(F) < n_0(F) + k(2n_1(F) + r)$.*

Proof. An *unmarked chain* in F is a chain that does not contain a marked vertex as an internal node. Construct F' by contracting all maximal unmarked chains of F into single edges. Assume that all nodes in F and hence all nodes in F' have degree at least 1. Let n_1, n_2, and n_3 denote the number of nodes of degree 1, degree 2, and degree ≥ 3, respectively, in F'. We know that $\sum_{v \in V(F')} deg_{F'}(v) \leq 2n(F') - 2$. Since the left-hand side is at least $n_1 + 2n_2 + 3n_3$ and $n(F') = n_1 + n_2 + n_3$, we find that $n_3 \leq n_1 - 2$ and hence $n_3 < n_1$. By the definition of F' we have $n_2 \leq r$ and thus $n(F') < 2n_1 + r$. By noting that $n(F) \leq n(F') + (k-1)m(F')$ and hence $n(F) < kn(F')$, it follows that $n(F) < k(2n_1 + r)$. Since $n_1 = n_1(F)$, the claim of the lemma follows. □

Lemma 3 *Let H be a 2-edge-connected graph and let S be a proper subset of $E(H)$. Let Q be a full contraction of H with respect to S. Then $n(Q) < 13|E(H) - S|$.*

Proof. Let C denote the set $E(H) - S$ and let Q' denote the graph $(V(Q), S \cap E(Q))$. The graph Q' is a forest. We may assume that $n(Q) > 1$ (otherwise the claim is trivial). Since Q is 2-edge-connected, each node of degree 0 in Q' is incident in Q with at least 2 edges of C (since Q is 2-edge-connected) and each node of degree 1 in Q' is incident in Q with at least one edge of C.

Thus, Q' has at most $|C|$ nodes of degree 0 and at most $2|C|$ nodes of degree 1. Mark each endpoint of an edge of C in Q'. Each unmarked chain in Q' has length at most 2. By applying lemma 2 to Q', we get $n(Q') < |C| + 2(4|C| + 2|C|)$ and hence $n(Q) = n(Q') < 13|C|$ as claimed. □

Corollary 3 *Let k denote the number of iterations of the while-loop of algorithm 4. Let H_i denote the graph H at the start of the ith iteration of the while-loop. Then $m(H_{i+14}) < .83m(H_i)$ for $i + 14 \leq k$.*

Proof. First, we show that

$$m(H_{i+1}) < 2n(H_i) \tag{1}$$

for $i < k$. To see this, let H' be the subgraph of H_i consisting of the nontrivial ears in the ear decomposition found in step 2.0 of algorithm 4 and let q denote the number of those ears. Then $m(H') \leq n(H') + q$ and $q < n(H')$, hence $m(H') < 2n(H')$. Since $n(H') = n(H_i)$ and $m(H_{i+1}) \leq m(H')$, inequality 1 follows.

Let q_i denote $|E(H) - S|$ at the start of the *ith* iteration of the while-loop of algorithm 4. By theorem 5 we have $q_{i+1} \leq 3q_i/4$ for $i < k$ and hence $q_{i+j} \leq (3/4)^j q_i$ for $i + j \leq k$. With lemma 3 we obtain

$$n(H_{i+j+1}) < 13q_{i+j} = 13(3/4)^j q_i \leq 13(3/4)^j m(H_i) \tag{2}$$

for $i + j + 1 \leq k$. Combining (1) and (2) yields

$$m(H_{i+j+2}) < 26(3/4)^j m(H_i) \tag{3}$$

for $i + j + 2 \leq k$. Substituting $j = 12$ yields the claim. □

Theorem 10 *Algorithm 4 computes a minimal 2-edge-connected spanning subgraph of any 2-edge-connected graph on n vertices and m edges in time and space $O(n + m)$.*

Proof. Consider one iteration of the while-loop. We compute an ear decomposition in linear time and space using the algorithm of [21]. We compute an optimal tree T_H and S-critical edges in linear time and space as described in the proof of theorem 6. We compute a minimal augmentation in linear time and space using the algorithm of [13]. To compute the S-critical edges, we determine the connected components of $(V(H), S)$. Finally, the complexity of computing a full contraction is dominated by the complexity of computing 2-edge-connected components ([22]), i.e., it can be done in linear time and space. In summary one iteration of algorithm 4 can be performed in linear time and space. The claim follows with corollary 3. □

Note: An efficient NC algorithm for this problem is given in [13]. With theorem 10 the work of this algorithm (time-processor product) can be reduced by a factor of $\Theta(\log n)$ using standard techniques.

4.2 Finding a Minimal Biconnected Spanning Subgraph

The linear time algorithm for computing a minimal 2-edge-connected spanning subgraph makes use of the two operations of shrinking S-components and contracting chains. In this section we exhibit a pair of operations on biconnected graphs with similar properties although they are more complicated. They will be used in the linear time algorithm for computing a minimal biconnected spanning subgraph.

Let H be biconnected and $S \subseteq E(H)$. An *S-block* of H is a block of the graph $(V(H), S)$. The graph $(V(H), S)$ need not be connected. Thus, an S-block of H is either an isolated vertex in $(V(H), S)$, a cutedge in $(V(H), S)$, or a maximal biconnected subgraph of $(V(H), S)$ with at least 3 vertices.

Let B be an S-block of H with at least 3 vertices. An *internal vertex* of B is a vertex in B that is neither a cutpoint in $(V(H), S)$ nor is it incident in H with an edge of $E(H) - S$; we write $I(B)$ for the set of internal vertices of B. The operation of *shrinking the S-block B in H* consists of deleting all edges of B in H as well as all internal vertices of B, connecting the remaining vertices of B into a cycle in arbitrary order, and subdividing each edge of this cycle with a new vertex. Thus, if $u_1, \ldots, u_k$ are the non-internal vertices of B, and $V' = \{v_1, \ldots, v_k\}$ is the set of k new vertices used to subdivide the edges of the cycle, then the resulting graph has vertex set $(V(H) - I(B)) \cup V'$ and edge set $(E(H) - E(B)) \cup C_B$ where $C_B = \{(u_1, v_1), (v_1, u_2), \ldots, (u_{k-1}, v_{k-1}), (v_{k-1}, u_k), (u_k, v_k), (v_k, u_1)\}$. The operation is illustrated in figure 1.

The following results establish that shrinking S-blocks preserves the biconnectivity structure of a graph.

Lemma 4 *Let H be a graph that may not be biconnected and let $S \subseteq E(H)$. Construct H' from H by shrinking an S-block B in H. Then H' is biconnected if and only if H is biconnected.*

Proof. Assume that H is biconnected. Thus, for any three distinct vertices u, v, w in H there exists a path from u to w avoiding v. Now suppose that H' is not biconnected. Let C_B denote the cycle in H' that corresponds to S-block B in H. For some distinct $u, v, w \in V(H')$ every path from u to w in H' passes through v. We may assume that $u, v, w \in V(H)$. To see this, note that if a vertex

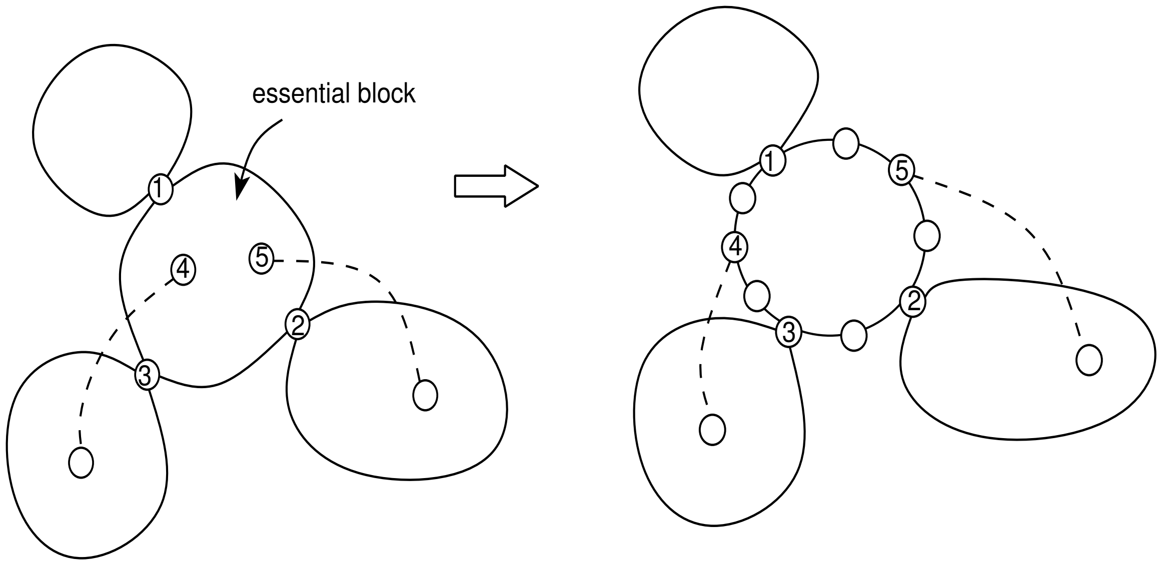

Figure 1: Shrinking the essential block containing vertices numbered $1-5$. The dashed edges are redundant edges.

in $\{u,v,w\}$ does not belong to H (i.e., it is a vertex of degree 2 on C_B), then we may replace it by one of its neighbors on C_B while maintaining the property that any path from u to w goes through v. If $\{u,v,w\} \subseteq V(H)$, then there exists a path P in H from u to w avoiding v. From P we obtain a path P' in H' from u to w by replacing each edge (z,z') of B on P by a path from z to z' on C_B avoiding v. Clearly P' avoids v. We conclude that H' is biconnected.

Now suppose that H' is biconnected. To show that H is biconnected, we need to prove that for any three distinct vertices u, v and w in H there is a path from u to w in H avoiding v. Since H' is biconnected, there exists a path P from u to w avoiding v in H'. If P contains a subpath from u_i to u_j on C_B, then we can replace each such subpath with a path from u_i to u_j in B avoiding v (since B is biconnected). We thus obtain a path P' from u to w in H that avoids v. We conclude that H is biconnected. $\square$

Corollary 4 *Let H be biconnected and let H' be obtained from H by shrinking an S-block B in H. An edge $e \in E(H') \cap E(H)$ is essential in H' if and only if it is essential in H.*

Proof. The edge e does not belong to B. Thus B is an S-block in $H - e$. The graph $H' - e$ may be obtained from $H - e$ by shrinking S-block B in $H - e$ and contracting at most two chains on the cycle C_B that replaces the block B of $H - e$. (The contractions are necessary if B has more internal vertices in $H - e$ than it has in H. This may happen if e is the only edge in $E(H) - S$ incident on

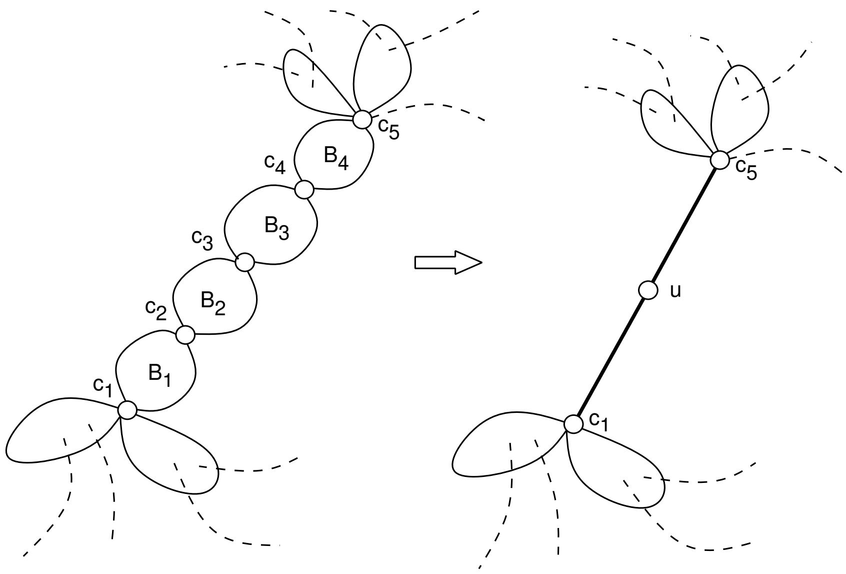

Figure 2: Contracting block chain $c_1B_1c_2B_2c_3B_3c_4B_4c_5$. The dashed lines represent edges in $E(H) - S$.

some vertex in B.) Since the biconnectivity property is closed under contractions (contractions are a special case of contracting block chains where each block is an edge), the previous lemma implies that $H - e$ is biconnected if and only if $H' - e$ is biconnected. $\square$

As before let H be a biconnected graph and $S \subseteq E(H)$. The second operation on biconnected graphs is defined on the block structure of $(V(H), S)$. An *S-block chain* in H is an alternating sequence $c_1B_1 \ldots c_kB_kc_{k+1}$, $k > 1$, of vertices and S-blocks in H with the following properties: (i) each B_i $(1 \leq i \leq k)$ has exactly two cutpoints in $(V(H), S)$, namely c_i and c_{i+1}; (ii) for $1 < i < k$, B_i intersects exactly two blocks, namely B_{i-1} and B_{i+1} in c_i and c_{i+1}, respectively; (iii) no vertex in any B_i except possibly c_1 and c_{k+1} is incident with an edge in $E(H) - S$. A *maximal S-block chain* in H is a block chain in H not properly contained in any other S-block chain of H. If the set S is understood, we may refer to an S-block chain as a block chain. Figure 2 shows a maximal S-block chain $c_1B_1c_2B_2c_3B_3c_4B_4c_5$.

The operation of *contracting the S-block chain* $c_1B_1 \ldots c_kB_kc_{k+1}$ in H consists of deleting in H all vertices in the S-blocks of this sequence except c_1 and c_{k+1}, adding a new vertex z and two new edges (c_1, z) and (z, c_{k+1}) (see figure 2).

Lemma 5 *Let H be a graph that may not be biconnected and $S \subseteq E(H)$. Construct H' by contracting the S-block chain $c_1B_1 \ldots c_kB_kc_{k+1}$ in H. Then H' is biconnected if and only if H is* *biconnected.*

Proof. Assume that H is biconnected. If H' is not biconnected, then there exist distinct $u, v, w \in V(H')$ such that any path from u to w in H' passes through v. Since H is biconnected, there is a path from c_1 to c_{k+1} in H avoiding c_2, yielding a path in H' from c_1 to c_{k+1} avoiding the new vertex z. Therefore $v \neq z$. If $u = z$ or $w = z$, then we can replace either vertex by c_1 or c_{k+1} while maintaining the property that any path from u to w in H' passes through v. We may thus assume that $u, v, w \neq z$ and hence $u, v, w \in V(H) \cap V(H')$. Thus there is a path P in H from u to w avoiding v. If P does not traverse an edge in any B_i, then it is a path in H' from u to w avoiding v. Otherwise P contains a subpath from c_1 to c_{k+1} (c_{k+1} to c_1). We may replace this subpath by the path $< c_1, z, c_{k+1} >$ ($< c_{k+1}, z, c_1 >$) in H', thus obtaining a path in H' from u to w that avoids v. Hence H' is biconnected.

Now let H' be biconnected. If H is not biconnected, then for some distinct vertices u, v, w in H any path from u to w in H goes through v. Since H' is biconnected, there is a path in H' from c_1 to c_{k+1} avoiding z, yielding a path in H from c_1 to c_{k+1} none of whose internal vertices belongs to any B_i. It follows that v is not a vertex in any B_i other than c_1 or c_{k+1}. It also implies that we may assume that neither u nor w is a vertex in any B_i other than c_1 or c_{k+1} while maintaining the property that any path from u to w in H passes through v. Thus the path in H' from u to w avoiding v yields a path in H from u to w avoiding v. Therefore H is biconnected. $\square$

Corollary 5 *Let H be biconnected and let H' be obtained from H by contracting a block chain $c_1 B_1 \ldots c_k B_k c_{k+1}$ in H. An edge $e \in E(H') \cap E(H)$ is essential in H' if and only if it is essential in H.*

Proof. Since e belongs to H and H', it does not belong to any block B_i in the block chain. Thus $c_1 B_1 \ldots c_k B_k c_{k+1}$ is a block chain in $H - e$ and $H' - e$ is obtained from $H - e$ by contracting this block chain in $H - e$. By the previous lemma $H - e$ is biconnected if and only if $H' - e$ is biconnected. The claim follows. $\square$

If the graph Q is obtained from H by first shrinking all S-blocks of H and then contracting all maximal block chains in the resulting graph, we say that Q is a *full contraction* of H with respect to S. The following algorithm is a variation of algorithm 3 in which we replace H by its full contraction at the end of each iteration of the while-loop. It is shown in [13] that the minimal augmentation step (step 2.2) can be done in linear time. We note that the intermediate graph H will always be simple assuming that the input graph G is simple (the usual assumption for biconnected graphs).

Algorithm 5 Computing a minimal biconnected spanning subgraph of G.

Input Biconnected graph G.

Output Minimal biconnected spanning subgraph of G.

(1) $H := G$; $S := \emptyset$;

(2) while $E(H) \not\subseteq S$, do:

(2.0) replace H by an open ear decomposition of H; remove edges in trivial ears from H;

(2.1) compute a spanning tree T_H in H with a maximum number of edges of S;

(2.2) compute a minimal $A \subseteq E(H)$ such that $T_H + A$ is biconnected;

(2.3) $H := T_H + A$; $B := \{S\text{-critical edges in } H\}$; $S := S \cup A \cup B$;

(2.4) replace H by its full contraction with respect to $S \cap E(H)$; add new edges to S;

(3) return graph $(V(G), E(G) \cap S)$.

Theorem 11 *Algorithm 5 outputs a minimal biconnected spanning subgraph of G.*

Proof. The proof is very similar to the proof of theorem 9. The termination of the algorithm is established by showing that $|E(H) - S|$ is strictly decreasing exactly as in the proof of theorem 9. Note that newly created edges are added both to $E(H)$ and S and thus do not increase $|E(H) - S|$.

Reusing the notation of that proof we need to show that H_i' is a minimal biconnected spanning subgraph of H_i. The statement (*) that any edge in $E(H_i) \cap S_i$ is essential in H_i is still true. The proof proceeds by induction over j just as in the proof of theorem 9 with the following minor modification: the edges in $E(H_{j+1}) \cap (S_{j+1} - S_j)$ are either essential in $T_j + A_j$ (as before) or they are new edges added by the full contraction. The edges in the first class are essential in H_{j+1} by corollaries 4 and 5 since H_{j+1} is a full contraction of $T_j + A_j$. The edges in the second class (new edges in H_{j+1}) are essential in H_{j+1} since they have an endpoint of degree 2.

One now proves that H_i' is a minimal biconnected spanning subgraph of H_i as in the proof of theorem 9. The main difference is in the inductive argument that H_i' is a minimal biconnected spanning subgraph of H_i: the fact that H_{i+1} is a full contraction of $T_i + A_i$ with respect to some $X \subseteq S_k$ implies that $H_{i+1}' = (V(H_{i+1}), E(H_{i+1}) \cap S_k)$ is obtained from $(V(T_i + A_i), E(T_i + A_i) \cap S_k) = H_i'$ by a full contraction with respect to X possibly followed by contracting chains. With lemma 4 and lemma 5 this implies that H_i' is 2-edge-connected. By corollaries 4 and 5 (and the induction assumption) the edges in $E(H_i') \cap E(H_{i+1}')$ are essential in H_i'. The edges in $E(H_i') - E(H_{i+1}')$

belong either to S_i, in which case they are essential in H_i and hence in H_i' by (*), or they belong to $S_{i+1} - S_i$, in which case they are essential in $T_i + A_i$ and hence in H_i'. $\square$

The remainder of the proof is the same. $\square$

The following result is needed for the analysis of algorithm 5.

Lemma 6 *Let H be biconnected and let S be a proper subset of $E(H)$. If Q is a full contraction of H with respect to S, then $n(Q) < 60|E(H) - S|$.*

Proof. Let C denote the set $E(H) - S$ and let Q' denote the graph $Q - C$. To bound $n(Q)$, we consider the block graph of Q'. Let H' be an arbitrary graph. The *block graph* of H' ([23]), denoted by $blk(H')$, is a bipartite graph whose vertices are the cutpoints and blocks of H'. A block is connected in $blk(H')$ to exactly those cutpoints that it contains in H'. It is known ([23]) that the block graph of H' is a tree for any connected graph H'. Thus the graph $blk(Q')$ is a forest.

Let n_0 and n_1 denote the number of degree 0 and degree 1 nodes in $blk(Q')$. If $n(blk(Q')) = 1$, then the graph $(V(H), S)$ has a unique block and this block contains at least 3 vertices. Hence Q' is a cycle (obtained by shrinking this block) of even length. Every other vertex on this cycle is incident with an edge of $C(\neq \emptyset)$. Hence $n(Q) \leq 4|C|$. If $n(blk(Q')) > 1$, then each node of degree 0 in $blk(Q')$ represents a block in Q' that is incident with at least 2 edges of C in Q and each node of degree 1 in $blk(Q')$ represents a block in Q' that is incident with at least one edge of C in Q. Thus $n_0 \leq |C|$ and $n_1 \leq 2|C|$. Mark a vertex in $blk(Q')$ representing a cutpoint in Q if it is incident in Q with an edge of C and mark a vertex in $blk(Q')$ representing a block of Q' if a vertex in this block other than a cutpoint is incident with an edge of C. From the definition of a full contraction it follows that any chain of unmarked nodes in $blk(Q')$ has length at most 6. By applying lemma 2 we see that $blk(Q')$ has fewer than $|C| + 6(4|C| + 2|C|) = 37|C|$ vertices. Since there are more blocks in Q' than there are cutpoints, Q' has fewer than $19 \cdot |C|$ cutpoints.

We partition the vertices of Q' into 3 classes: class 1 contains the cutpoints in Q', class 2 includes the endpoints of edges of C (that are not cutpoints), and class 3 comprises the new vertices used to subdivide cycles when shrinking S-blocks in H. Let p_1, p_2, and p_3 denote the number of vertices in class 1, class 2, and class 3, respectively. Clearly $p_2 \leq 2|C|$. Above we have shown that $p_1 < 19|C|$. Note that the number of class 3 vertices in any block of Q' is no larger than the number of vertices in that block that belong to class 1 or class 2. The sum of the latter number, taken over all blocks of Q', is an upper bound on p_3. This sum is at most $2|C| + m(blk(Q'))$ and hence $p_3 < 2|C| + 37|C| = 39|C|$. Altogether we find that $p_1 + p_2 + p_3 < 60|C|$ and hence $n(Q') = n(Q) < 60|C|$. $\square$

Corollary 6 *Let k denote the number of iterations of the while-loop of algorithm 5. Let H_i denote the graph H at the start of the ith iteration of the while-loop. Then $m(H_{i+22}) < .77m(H_i)$ for $i + 22 \leq k$.*

Proof. We first argue that

$$m(H_{i+1}) < 4n(H_i) \tag{4}$$

for any $i < k$. Let H' denote the subgraph of H_i consisting of the nontrivial ears in the open ear decomposition found in step 2.0 of algorithm 5. Just as in the proof of corollary 3 we have $m(H') < 2n(H') = 2n(H_i)$. If we shrink a block in $T_i + A_i$, the number of new edges added is bounded by the number of vertices in the block, which is at most the number of edges in the block. It follows that the total increase in the number of edges during the full contraction of $T_i + A_i$ is at most $m(T_i + A_i) \leq m(H')$. We conclude that $m(H_{i+1}) < 4n(H_i)$.

Let q_i denote $|E(H) - S|$ at the start of the *ith* iteration of the while-loop of algorithm 5. Because any new edge created during the full contraction is also added to S, theorem 5 implies that $q_{i+1} < 3q_i/4$ for $i < k$ and hence $q_{i+j} \leq (3/4)^j q_i$ for $i + j \leq k$. With lemma 6 we obtain

$$n(H_{i+j+1}) < 60q_{i+j} \leq 60(3/4)^j m(H_i) \tag{5}$$

for $j > 0$ and $i + j + 1 \leq k$. Combining (4) and (5) yields

$$m(H_{i+j+2}) < 240(3/4)^j m(H_i) \tag{6}$$

for $j > 0$ and $i + j + 2 \leq k$. Substituting $j = 20$ yields the claim of the corollary. □

Theorem 12 *Algorithm 5 finds a minimal biconnected spanning subgraph of any biconnected graph on n vertices and m edges in time and space $O(n + m)$.*

Proof. Consider one iteration of the while-loop. We compute an open ear decomposition in linear time and space using the algorithm of [21]. We compute an optimal tree T_H and S-critical edges in linear time and space as described in the proof of theorem 6. We compute a minimal augmentation in linear time and space using the algorithm of [13]. Finally, the complexity of computing a full contraction is dominated by the complexity of computing blocks, i.e., it can be done in linear time and space ([22]). In summary one iteration of algorithm 5 can be performed in linear time and space. The claim follows with corollary 6. □

As was the case for 2-edge-connectivity the last result yields an improvement by a $\Theta(\log n)$ factor in the work of an efficient NC algorithm described in [13] for computing a minimal biconnected spanning subgraph.

5 Worst-Case Analysis

In this section we prove that algorithm 3 requires $\Theta(m + n \log n)$ in the worst case for 2-edge-connectivity and biconnectivity. This result justifies the work we invested in the last section to achieve a linear running time. Because of its simple structure algorithm 1 will be a more convenient vehicle for proving lower bounds. Fortunately, any lower bound on the number of iterations of algorithm 1 yields the same lower bound for the number of iterations of algorithm 3.

Lemma 7 *Let H be a P-graph, S a subset of the essential edges in H, T a spanning tree in H with a maximum number of essential edges and A a minimal augmentation for T in H. Then $T + A$ can be rewritten as $T' + A'$ where T' is a spanning tree in H with a maximum number of edges of S and A' is a minimal augmentation for T' in H.*

Proof. An *essential component* of H is a maximal subgraph of H containing a spanning tree of essential edges. Let $C_1, \ldots, C_k$ be the essential components of H. For each i let T_i be a spanning tree of essential edges for C_i with a maximum number of edges of S. Let F be the set of edges of T that are redundant in H. Then the tree T' with edge set $F \cup E(T_1) \cup \ldots \cup E(T_k)$ is a spanning tree in H. The tree T' contains a maximum number of edges of S since the intersection of any other tree with C_i is a forest that contains no more than $|E(T') \cap E(C_i)|$ edges of S. We have $E(T') \subseteq E(T + A)$. Let $A' = E(T + A) - E(T')$. The graph $T' + A'$ $(= T + A)$ has property P. Moreover, each edge in A' is essential in $T' + A'$ since T' contains all edges of T that are redundant in H. We conclude that A' is a minimal augmentation for T' in H. □

A *trace of algorithm 1* for P-graph G is a sequence of graphs representing H at the start of successive iterations of algorithm 1. Formally, $H_0, H_1, \ldots, H_l$ is a trace of algorithm 1 for P-graph G if $H_0 = G$, $H_i \neq H_{i+1}$ for $0 \leq i < l$, and H_i $(0 < i \leq l)$ is of the form $T + A$ where T is an optimal tree in H_{i-1} and A is a minimal augmentation for T in H_{i-1}. The integer l is the *length* of the trace. Similarly, we define a *trace* of algorithm 3 for P-graph G to be a sequence of graphs representing H at the start of successive iterations of algorithm 3. The sequence $H_0, H_1, \ldots, H_l$ is a trace of algorithm 3 for P-graph G if $H_0 = G$, $H_i \neq H_{i+1}$ for $0 \leq i < l$, and, for any sequence of sets $E_0, E_1, \ldots, E_{l-1}$ such that E_i is a set of essential edges in H_i for $0 \leq i \leq l-1$, we can write H_i as $T_{i-1} + A_{i-1}$ where T_{i-1} is a spanning tree in H_{i-1} containing a maximum number of edges of E_{i-1} and A_{i-1} is a minimal augmentation for T_{i-1} in H_{i-1}. The following result is an immediate corollary of lemma 7.

Corollary 7 *A trace of algorithm 1 for P-graph G is also a trace of algorithm 3 for G.* □

By corollary 7 a lower bound on the number of iterations of algorithm 1 implies the same lower bound for the number of iterations of algorithm 3. Below we make use of this fact: we derive a tight $\Theta(\log n)$ lower bound on the (worst-case) number of iterations of algorithm 1 that also applies to algorithm 3.

To capture the worst-case behavior of algorithm 1, we define the *P-complexity* of a graph. Informally, the P-complexity of a P-graph H is the maximum number of iterations that algorithm 1 may need in order to compute a minimal spanning P-subgraph of H. Formally, we define the P-complexity of H to be the maximum length of a trace of algorithm 1 for H. If the property P is clear, we shall use the term "complexity" instead of "P-complexity". We denote the P-complexity of graph H by $c_P(H)$, or $c(H)$ if property P is understood. Note that the notion of P-complexity corresponds to an adversary choosing, for a given input graph, an optimal tree and a minimal augmentation in each iteration of the while-loop of algorithm 1 with the goal of maximizing the number of such iterations. Hence we consider the worst-case behavior of algorithm 1 not only with respect to different input instances but also with respect to different choices of the algorithm (i.e., sequences of trees and augmentations). We call an infinite sequence of graphs $H_0, H_1, H_2, \ldots$ such that $c_P(H_i) \geq i$ for all $i \geq 0$ a *sample* for P.

We construct a sample for 2-edge-connectivity inductively: F_0 has two vertices with two parallel edges between them. We construct F_i from F_{i-1} as follows: we double each essential edge in F_{i-1}; we call the resulting graph F_i'. For each vertex u of degree d in F_i', we add d new vertices $u_1, u_2, \ldots, u_d$ to F_i and connect them in a cycle. We number the edges incident on each vertex in F_i' as $0, 1, \ldots$ and, for each edge $e = (u, v)$ in F_i' that is the ith edge incident on u and the jth edge incident on v, we add an edge (u_i, v_j) to F_i. Finally, we subdivide each edge of F_i' connecting two vertices u_i, u_j corresponding to the same vertex u in F_i' with a new vertex. Note that F_i is uniquely defined up to isomorphism. Figure 3 shows the first three graphs in this sequence.

The proof that the F_i's form a sample for 2-edge-connectivity relies on the following property of 2-edge-connectivity. Let us say that H' is an essential contraction of H if H' is obtained from H by shrinking (vertex-disjoint) S-components in H' where S is a set of essential edges in H. (The operation of shrinking an S-component is defined in section 4.1.)

Lemma 8 *Let H' be an essential contraction of a 2-edge-connected graph H and let $H_0', H_1', \ldots, H_k'$ be a trace of algorithm 1 for H'. Then, there is a trace $H_0, H_1, \ldots, H_k$ of algorithm 1 for H such that H_i' is an essential contraction of H_i for $0 \leq i \leq k$. Hence $c(H') \leq c(H)$.*

Proof. Fix a 2-edge-connected graph H. Let H' be an essential contraction of H obtained from H by shrinking S-components in H. By lemma 1 H' is 2-edge-connected and its complexity is indeed well-defined. Fix a trace $H'_0, H'_1, \ldots, H'_k$ of algorithm 1 for H'. We prove the lemma by induction on k. The base case $k = 0$ is clear: the graph H constitutes a trace of length 0 for H and H' is an essential contraction of H.

Assume that the claim holds if the trace for H' has length at most l. Let $H'_0, H'_1, \ldots, H'_{l+1}$ be a trace for H'. Let $H'_1 = T' + A'$ where T' is an optimal tree in H' and A' is a minimal augmentation for T' in H'. We may combine T' with spanning trees for the S-components in H to form a spanning tree T of H. By the optimality of T and corollary 2 the tree T contains a maximum number of essential edges in H that do not belong to S. Since T contains a spanning tree for each S-component, it also contains a maximum number of essential edges in S. Thus, T is an optimal tree in H. Let $A = A' \cup (S - E(T))$. By corollary 2 and the fact that all edges in S are essential it follows that A is a minimal augmentation for T in H. The graph $T' + A'$ is an essential contraction of $T + A$. By the induction assumption there exists a trace $H_1, H_2, \ldots, H_{l+1}$ of algorithm 1 for $T + A$ such that H'_i is an essential contraction of H_i for $1 \leq i \leq l + 1$. The claim of the lemma follows. □

Theorem 13 *The graphs $F_0, F_1, \ldots$ form a sample for 2-edge-connectivity.*

Proof. The graph F'_i obtained from F_{i-1} by doubling the essential edges is certainly 2-edge-connected if F_{i-1} is 2-edge-connected. Moreover F'_i is an essential contraction of F_i. Thus F_i is 2-edge-connected if F_{i-1} is 2-edge-connected. Since F_0 is 2-edge-connected all F_i's are 2-edge-connected. Hence the complexity of F_i is well defined. We prove by induction on i the following two statements: (1) $c(F_i) \geq i$; (2) F_i has a spanning tree containing all redundant edges of F_i.

Statement (1) trivially holds for $i = 0$. Statement (2) holds for $i = 0$ since all edges in F_0 are essential. Assume inductively that the claim holds for $i = j - 1$. Recall that each vertex in F_{j-1} corresponds to a cycle in F_j; we refer to such a cycle in F_j as an F_j-cycle. All edges in F_j-cycles are essential in F_j since they have an endpoint of degree 2. All other edges in F_j are redundant in F'_j. Since F'_j is an essential contraction of F_j, corollary 2 implies that these edges are redundant in F_j as well. By the construction of F_j no two redundant edges are incident on the same vertex. Hence there is a spanning tree in F_j containing all redundant edges of F_j, establishing statement (2).

To prove (1), fix a spanning tree T_{j-1} in F_{j-1} containing all redundant edges in F_{j-1}. Thus, $F_{j-1} = T_{j-1} + A_{j-1}$ where A_{j-1} is a minimal augmentation for T_{j-1} in F_{j-1}. We can combine

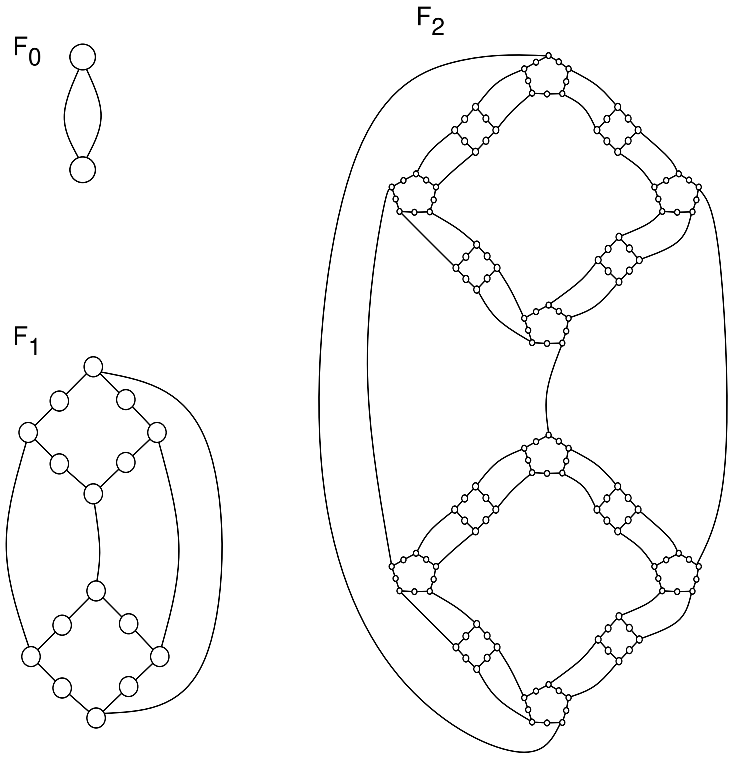

Figure 3: F_0, F_1, and F_2.

T_{j-1} with spanning trees for the F_j-cycles (obtained by deleting an edge in each F_j-cycle) to form a spanning tree T_j of F_j. The tree T_j is an optimal tree in F_j since it contains a maximum number of edges in F_j-cycles. Let A_j be the edges in A_{j-1} as well as the edges in the F_j-cycles that are not in T_j. With corollary 2 we see that A_j is a minimal augmentation for T_j in F_j. The graph F_{j-1} is an essential contraction of $T_j + A_j$. By the induction assumption and the previous lemma $c(T_j + A_j) \geq j - 1$. We conclude that $c(F_j) \geq j$. $\square$

Lemma 9 *Let n_i, m_i, and e_i denote the number of vertices, edges, and essential edges, respectively, in F_i ($i \geq 0$). These quantities satisfy the following recurrence relations:*

$$\begin{aligned} n_{i+1} &= 4(m_i + n_i), \\ m_{i+1} &= m_i + e_i + n_{i+1}, \\ e_{i+1} &= n_{i+1}, \end{aligned}$$

with initial conditions $n_0 = m_0 = e_0 = 2$. Thus, $n_i = 4 \cdot 9^{i-1}$ and $m_i = 5 \cdot 9^{i-1}$ for $i > 0$. $\square$

Corollary 8 *For 2-edge-connectivity, there exists a function $f(n) = \Omega(\log n)$ such that there is a graph on n vertices of complexity $f(n)$ for any $n \geq 1$.*

Proof. To construct a graph of complexity $\Omega(\log n)$ with exactly n vertices, start with F_i where i is the maximum integer such that $n_i \leq n$ and increase the number of vertices in F_i to n by repeatedly subdividing an essential edge. The resulting graph has the same complexity as F_i, namely $\Omega(\log n)$. $\Box$

Let us now turn our attention to biconnectivity. Unfortunately, biconnectivity does not satisfy a condition similar to lemma 8. We do however have the following result.

Lemma 10 *If G is a graph with at least three vertices in which each vertex has degree ≤ 3, then G is biconnected if and only if G is 2-edge-connected.*

Proof. The only-if-part is clear. Assume that G is 2-edge-connected and let u be a cutpoint in G. Hence, at least 2 blocks share u. Both of these blocks are 2-edge-connected and hence the degree of u is at least 4, contradicting the assumption that each vertex in G has degree ≤ 3. $\Box$

Let us call a graph in which each vertex has degree ≤ 3 a $3-graph$. Let P="2-edge-connectivity" and P'="biconnectivity".

Corollary 9 *If a graph H is a $3-graph$, then $c_P(H) = c_{P'}(H)$.*

Proof. By induction on $c_P(H)$. The induction base is clear with lemma 10. Assume that the claim holds for $c_P(H) \leq k-1$. Fix an H with $c_P(H) = k$. Thus, $H = T+A$ with $c_P(T+A) = k-1$ where T is an optimal tree in H with respect to P and A is a minimal augmentation for T in H (with respect to P). By lemma 10, an edge in H is P-redundant if and only if it is P'-redundant. Hence, T is an optimal tree in H with respect to P' and A is a minimal augmentation for T in H with respect to P'. Since $T+A$ is a $3-graph$ the induction hypothesis gives us $c_{P'}(T+A) \geq k-1$ and hence $c_{P'}(H) \geq k = c_P(H)$. Similarly, one proves $c_P(H) \geq c_{P'}(H)$. We conclude that $c_P(H) = c_{P'}(H)$. $\Box$

Each F_i is a 3-graph. Since $n(F_i) \geq 3$ for $i \geq 1$, we have $c_{P'}(F_i) \geq i$ for $i \geq 1$ and the sequence $F_1, F_2, \ldots$ is a sample for biconnectivity. Thus, we get the following result.

Corollary 10 *For biconnectivity, there exists a function $g(n) = \Omega(\log n)$ such that there is a graph on n vertices with complexity $g(n)$ for any $n \geq 3$.*

Proof. Similar to proof of corollary 8. $\Box$

Corollaries 8 and 10 establish that algorithm 1 takes $\Omega(\log n)$ iterations for 2-edge-connectivity and biconnectivity in the worst case. Since each iteration of these algorithms takes $\Omega(n)$ time they require $\Omega(m + n \log n)$ time in the worst case. By corollary 7 the same bound applies to algorithm 3. Because of theorem 7 this bound is tight for algorithm 3. In [13] it is shown that redundant edges can be determined for 2-edge-connectivity and biconnectivity in linear time by modifying the linear-time triconnectivity algorithm of [21]. Thus the bound is tight for algorithm 1 as well.

Theorem 14 *Algorithm 1 and algorithm 3 require $\Theta(m + n \log n)$ operations in the worst case, both for 2-edge-connectivity and biconnectivity.* □

6 Computing Minimal Spanning Subgraphs in Directed Graphs

In this section we generalize the algorithms for undirected graphs to directed graphs. We provide general algorithms that compute a minimal spanning P-subgraph in a directed graph for any property P that is monotone and implies strong connectivity. For the special case of strong connectivity we obtain an algorithm that computes a minimal strongly connected spanning subgraph in time $O(m+n \log n)$. This algorithm is simpler (both in the sequential and parallel implementation) than an earlier algorithm of [8] for this problem because it avoids the explicit computation of redundant edges. We prove that our algorithms require $\Theta(m + n \log n)$ time in the worst case for strong connectivity. By adapting the analysis we also get a $\Theta(m + n \log n)$ tight bound on the worst-case running time of an algorithm of [8] for finding a minimal strongly connected spanning subgraph. This answers an open question of [8].

6.1 High-Level Algorithm

Let G be a directed graph. A *forward branching* ([8]) rooted at a vertex x is a spanning subgraph of G whose underlying graph is a tree and in which x has in-degree zero and all other vertices have in-degree one. Throughout this section we shall assume that all forward branchings are rooted at a fixed vertex x of G.

It turns out that the development carried out in section 3 for undirected graph properties carries over to digraphs with some modifications. The definitions of digraph property, P-graph, P-subgraph, and P-redundant and P-essential edges carry over from the undirected case without change. As for undirected graph properties we shall always assume that the property P is decidable.

The following conditions $D1$ and $D2$ correspond to conditions $C1$ and $C2$ in the undirected case (see section 3). Note that condition $D1$ (monotonicity of P) is identical with $C1$.

($D1$) P is monotone, i.e., adding edges preserves property P.

($D2$) Any P-graph is strongly connected.

Algorithm 6 computes a minimal spanning P-subgraph of digraph G, i.e., a spanning P-subgraph of G in which all edges are P-essential. It has a structure similar to that of algorithm 1; instead of computing a spanning tree in step (2.1), it computes a forward branching T_H rooted at a fixed vertex x of H (and containing a maximum number of P-essential edges). We call T_H an *optimal branching* in H (rooted at x) and A (see step (2.2) of algorithm 6), as before, a *minimal augmentation* for T_H in H.

Algorithm 6 Computing a minimal spanning P-subgraph of G.
Input Digraph G with property P.
Output Minimal spanning P-subgraph H of G.

(1) $H := G$.

(2) While H has P-redundant edges, do:

(2.1) compute a forward branching T_H in H, rooted at x, with a maximum number of P-essential edges;

(2.2) compute a minimal subset A of edges in H such that $T_H + A$ has property P;

(2.3) $H := T_H + A$.

The following result is the analog of theorem 1. The proof is similar.

Theorem 15 *Algorithm 6 computes a minimal spanning P-subgraph of G for any digraph property P satisfying $D1$ and $D2$.* $\square$

The following lemma will be needed to establish a logarithmic upper bound on the number of iterations of algorithm 6. Let H be a digraph that contains a forward branching rooted at x. We say that an edge e of H is *f-redundant* in H if $H - e$ contains a forward branching rooted at x; an edge of H that is not f-redundant is *f-essential* in H.

Lemma 11 *Let B be a set of f-redundant edges in H. There exists a forward branching rooted at x that contains at most half of the edges in B.*

Proof. Double the f-essential edges in H to obtain a graph H'. Let $X \subseteq V(H)$ with $x \in X$ and $X \neq V(H)$. Then there are at least two edges in H' from X to $V(H) - X$. By Edmonds' branching theorem ([6]) H' contains two edge-disjoint forward branchings rooted at x. Since no edge in B has been doubled, at least one of these two forward branchings contains at most half of the edges in B, yielding a forward branching in H rooted at x with the same property. □

Theorem 16 *If P satisfies $D1$ and $D2$, then algorithm 6 terminates after $O(\log n)$ iterations of the while-loop.*

Proof. Consider the start of an iteration of the while-loop. Let B denote the set of redundant edges in H. Each edge $e \in B$ is f-redundant in H since $H - e$ is strongly connected. By lemma 11 there exists a forward branching rooted at x containing at most half of the edges in B. Thus an optimal branching in H contains at most half of the redundant edges in H. Therefore at least half of the redundant edges in H are discarded or become essential in this iteration of the while-loop. The claim follows. □

As for undirected graph properties one can avoid the explicit computation of the P-essential (or P-redundant) edges by gradually building up a set of essential edges. The following algorithm uses this idea. We use m to denote the number of edges in G.

Algorithm 7 Computing a minimal spanning P-subgraph of G.
Input Digraph G with property P.
Output Minimal spanning P-subgraph H of G.

(1) $H := G$; $S := \emptyset$;

(2) for $i = 1$ to $\lceil \log m \rceil + 1$ do:

(2.1) compute forward branching T_H in H rooted at x with maximum number of edges of S;

(2.2) compute a minimal $A \subseteq E(H)$ such that $T_H + A$ has property P;

(2.3) $H := T_H + A$; $S := S \cup A$;

Theorem 17 *Algorithm 7 computes a minimal spanning P-subgraph of a digraph G with property P provided P satisfies D1 and D2.*

Proof. A straightforward induction over the iteration number shows that at the end of each iteration of the while-loop H is a spanning P-subgraph of G and the edges of S are essential in H.

We now show that H is minimal upon termination. Fix an iteration i of the while-loop. Let H_i (S_i) denote the graph H (the set S) at the beginning of iteration i and let R_i denote the set of edges in $E(H_i) - S_i$ that are f-redundant in H_i. To prove that the final graph H is minimal, it suffices to show that $|R_{i+1}| \leq \frac{1}{2}|R_i|$. Indeed, after $j = \lceil \log m \rceil + 1$ iterations we then have $R_j = \emptyset$, implying that all edges in $E(H_j) - S_j$ are f-essential in H_j and thus essential in H_j. Since all edges in S_j are essential as well, the graph H_j is minimal.

Any forward branching T in H_i must contain all edges of $E(H_i) - S_i$ that are f-essential in H. Thus a spanning tree T contains a maximum number of edges of S if and only if it contains a minimum number of edges in R_i. By lemma 11 (with $B = R_i$) there exists a forward branching in H_i that contains at most half of the edges in R_i. We conclude that at least half of the edges in R_i are discarded or added to S in iteration i. Hence $|R_{i+1}| \leq \frac{1}{2}|R_i|$. $\square$

6.2 Computing a Minimal Strongly Connected Spanning Subgraph

In this subsection we adapt algorithm 7 to strong connectivity. Before we do this we review the algorithm of [8] for computing a minimal strongly connected spanning subgraph (called *transitive compaction* in [8]).

The following definitions from [8] will be needed. Let G be a directed graph and $x \in V(G)$. An *inverse branching* rooted at x is a spanning subgraph of G whose underlying graph is a tree and in which x has out-degree zero and all other vertices have out-degree one. A *branching* is either a forward or an inverse branching. We assume that all branchings are rooted at a fixed vertex x of G. Let H be a subgraph of G. An *H-philic (H-phobic)* branching in G is one that has the greatest (smallest) number of edges in common with H over all branchings (rooted at x) in G.

In [8] the following algorithm is given for finding a minimal strongly connected spanning subgraph in a strongly connected digraph.

Algorithm 8 Computing a minimal strongly connected spanning subgraph H of G.
Input Strongly connected digraph G.
Output Minimal strongly connected spanning subgraph H of G.

(1) $H := G$;

(2) while H has redundant edges, do:

(2.1) $R :=$ set of redundant edges in H;

(2.2) $F :=$ R-phobic forward branching in H;

(2.3) $I :=$ F-philic inverse branching in H;

(2.4) $H := F \cup I$.

One iteration of the while-loop of algorithm 8 can be performed in linear time (see [8] for details). Algorithm 8 is a special case of algorithm 6 in the following sense: first, F (computed in step 2.2) is an optimal branching in H; second, $F \cup I = F \cup (I - F)$ and $I - F$ is a minimal augmentation for F in H. Thus, theorem 16 (or a similar result of [8]) implies that algorithm 8 runs in $O(m + n \log n)$ time.

We now adapt algorithm 7 for strong connectivity so that it runs in $O(m+n \log n)$ time as well. A spanning tree T_H with a maximum number of edges of S is an S-philic forward branching. It can be computed in linear time using Edmonds' minimum weight branching algorithm as explained in [8]. To compute a minimal augmentation A for T_H, we compute a T_H-philic inverse branching I and set $A = E(I) - E(T_H)$. This can again be done in linear time using Edmonds' minimum weight branching algorithm ([8]). Thus, algorithm 7 computes a minimal strongly connected spanning subgraph in time $\Theta(m + n \log n)$.

This implementation of algorithm 7 is simpler than algorithm 8 since it does not require the redundant edges to be computed at each step (this is a fairly involved procedure). These simplifications are even more apparent in the parallel implementation of algorithm 8. Indeed, most of [8] is concerned with developing a fairly involved parallel algorithm for computing the redundant edges. Algorithm 7 can be parallelized just as algorithm 8. It does not require redundant edges to be computed (resulting in a much simpler implementation) while achieving the same parallel complexity.

As pointed out in [8] it is conceivable that algorithm 8 terminates in a constant number of iterations of the while-loop, resulting in a linear worst-case running time. It would then be asymptotically faster than algorithm 7 which always runs in time $\Theta(m + n \log n)$. In the remainder of this section we shall rule out this possibility by showing that algorithm 8 requires $\Theta(m + n \log n)$ time in the worst case, thus answering a question of [8]. Because algorithm 8 is a special case of algorithm 6 this will imply a similar result for the worst-case time complexity of algorithm 6 (for strong connectivity).

To analyze the worst-case time complexity of algorithm 8, we first observe that this algorithm chooses a *minimum* augmentation for F at each step, i.e., a minimal augmentation of smallest size.

Lemma 12 *Fix an iteration of the while-loop of algorithm 8. The edges of I that are not in F form a minimum augmentation for F in H. Conversely, if A is a minimum augmentation for an R-phobic forward branching F in H, then $F + A$ is of the form $F \cup I$ where I is an F-philic inverse branching in H.*

Proof. The lemma follows routinely from the following two facts: $F + A$ is strongly connected if and only if it contains an inverse branching (rooted at x) and all branchings in H have the same number of edges. □

The previous lemma motivates the following definition. For a strongly connected digraph H a *minimum augmentation trace for H with respect to x* is a sequence $H_0, H_1, \ldots, H_k$ such that $H_0 = H$, $H_i \neq H_{i+1}$ for $0 \leq i < k$, and H_i $(0 < i \leq k)$ is of the form $T + A$ where T is an optimal branching in H_{i-1} (i.e., a forward branching with a minimum number of redundant edges) rooted at x and A is a *minimum* augmentation for T in H_{i-1}. The integer k is the *length* of the trace. Let $\hat{c}(H, x)$ denote the maximum length of a minimum augmentation trace for H with respect to x and let $\hat{c}(H)$ stand for $max\{\hat{c}(H, x) : x \in V(H)\}$. Note that $\hat{c}(H)$ is the worst-case number of iterations of algorithm 8 if we let an adversary choose a root and choose in each iteration an R-phobic forward branching F and an F-philic inverse branching I. A minimum augmentation sample (for strong connectivity) is a sequence of digraphs $F'_0, F'_1, \ldots$ such that $\hat{c}(F'_i) \geq i$.

The procedure for constructing a minimum augmentation sample for strong connectivity is similar to that for constructing a sample for 2-edge-connectivity although it is more complicated. Let F'_0 be a directed cycle of length 2. We use x to denote an arbitrary fixed vertex in F'_i. We construct F'_{i+1} from F'_i as follows:

Algorithm 9 Computing a minimum augmentation sample for strong connectivity.
Input Digraph F'_i.
Output Digraph F'_{i+1}.

(1) Double the essential edges in F'_i. Call the resulting graph G.

(2) Let $\hat{T}$ be a forward branching in G rooted at x.

(3) For each vertex v in G of degree d, we add d new vertices $v_1, \ldots, v_d$ – called the *representatives of v* – to F'_{i+1} and connect them into a (directed) cycle; subdivide each edge on this cycle with a new vertex.

(4) For each vertex v in G of degree d, number the representatives of v as follows: fix a path $P(v)$ that spans the cycle constructed for v in step 3 and that is rooted at a representative v_i of v. Assign labels $1, 2, \ldots, d$ to the representatives of v in increasing order of their distance from v_i on $P(v)$. (Hence, v_i is labeled 1.)

(5) For a vertex v in G of indegree p and outdegree q, label the $p+q$ edges incident on v as follows: if $v \neq x$, assign label 1 to the unique incoming edge that belongs to $\hat{T}$ (constructed in step 2) ; the outgoing edges receive labels $2 \ldots 1+q$ and the labels $2+q \ldots p+q$ are assigned to the other incoming edges. If $v = x$, then assign the labels $1 \ldots q$ to the outgoing edges and the labels $1+q \ldots p+q$ to the incoming edges.

(6) For each edge (u, v) in G that has label i at u and label j at v (with respect to the labeling of step 5), add an edge from the representative of u with label i to the representative of v with label j in F'_{i+1} (with respect to the labeling defined in step 4).

Figure 4 illustrates steps 3-6 of algorithm 7. Figure 5 shows the first three graphs in a possible sample. Note that these graphs are orientations of the corresponding graphs in the sample for 2-edge-connectivity (see figure 3).

The next two results will be needed to prove that the graphs F'_i form a minimal augmentation sample for strong connectivity.

Lemma 13 *Let H be a directed graph and $S \subseteq E(H)$. Let H' be obtained from H by collapsing in H the vertex sets of the strong components of $(V(H), S)$. Then H is strongly connected if and only if H' is strongly connected. Furthermore, if both graphs are strongly connected, then an edge $e \in E(H')$ is essential in H' if and only if it is essential in H.*

Proof. A straightforward adaptation of the proofs of the corresponding claims for shrinking S-components in a 2-edge-connected graph (see lemma 1 and corollary 2). $\square$

We say that H' is an *essential contraction* of a strongly connected graph H (with respect to a subset S of $E(H)$) if H' is obtained from H by collapsing in H the vertex sets of the strong components of $(V(H), S)$ and the edges in S are essential in H. If $v \in V(H)$ belongs to a strong component that is collapsed into a new vertex z, then we say that v is collapsed into z. If $v \in V(H) \cap V(H')$ then we say that v is collapsed into v.

Lemma 14 *Let H' be an essential contraction of a strongly connected graph H and let $H'_0, H'_1, \ldots, H'_k$ be a minimum augmentation trace for H' with respect to a vertex x. Then, there is a minimum*

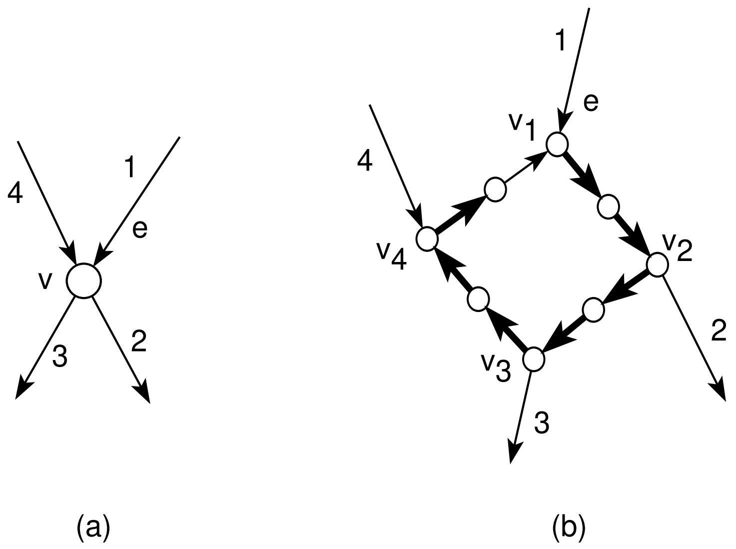

Figure 4: (a) Vertex $v \neq x$ in G of degree 4; the edge e belongs to branching $\hat{T}$. The labels on the edges are those assigned in step 5. (b) The cycle through the representatives of v together with the edges in F'_{i+1} corresponding to the edges labeled $1-4$ in (a). The thick edges are those in the path $P(v)$ (rooted at v_1).

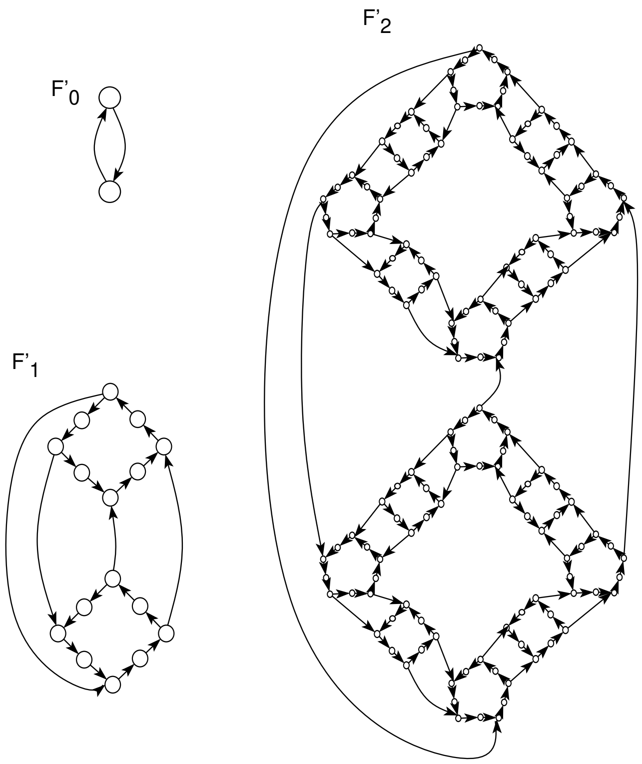

Figure 5: F'_0, F'_1, and F'_2.

augmentation trace $H_0, H_1, \ldots, H_k$ for H with respect to any vertex y that is collapsed into x such that H_i' is an essential contraction of H_i for $0 \le i \le k$. Hence $\hat{c}(H') \le \hat{c}(H)$.

Proof. Fix a strongly connected graph H. Let H' be an essential contraction of H with respect to a set S of essential edges in H. By lemma 13 H' is strongly connected. We prove the lemma by induction on k, the length of the trace for H'. If $k = 0$, then the minimum augmentation trace for H' consists of H' itself. The graph H by itself constitutes a minimum augmentation trace of length 0 for H (with respect to an arbitrary vertex). The base case follows.

Let $H_0', H_1', \ldots, H_k'$ be a minimum augmentation trace for H' with respect to a vertex x with $k > 0$. Let $H_1' = T' + A'$ where T' is an optimal branching in $H_0' = H'$ rooted at x and A' is a minimum augmentation for T' in H'. We may combine T' with forward branchings for the strong components of $(V(H), S)$ to form a forward branching T of H rooted at an arbitrary vertex y of H collapsed into x. By the optimality of T' and lemma 13 the forward branching T contains a maximum number of essential edges in H that do not belong to S (among all forward branchings in H rooted at y). Since T contains a forward branching for each strong component of $(V(H), S)$, T also contains a maximum number of edges of S (which are essential in H). Thus, T is an optimal branching in H rooted at y. Let $A = A' \cup (S - E(T))$. By lemma 13 and the fact that all edges in S are essential it follows that A is a minimum augmentation for T in H. The graph $T' + A'$ is an essential contraction of $T + A$. By the induction assumption there is a minimum augmentation trace $H_1, H_2, \ldots, H_k$ for $T + A$ with respect to any vertex z that is collapsed into x such that H_i' is an essential contraction of H_i for $1 \le i \le k$. We prefix this minimum augmentation trace with H to get a minimum augmentation trace $H, H_1, \ldots, H_k$ for H that has the claimed property. □

Theorem 18 *We have $\hat{c}(F_i') \ge i$ for all $i \ge 0$.*

Proof. We prove the following stronger statement: for each $i \ge 0$ there is a vertex x in F_i' such that $\hat{c}(F_i', x) \ge i$ and the edge set of F_i' can be partitioned into a forward branching rooted at x and a set of essential edges. We show this by induction on i.

The base case clearly holds. Assume the statement holds for F_i'. Let G be the digraph constructed by doubling the essential edges in F_i' (step 1 of algorithm 9). By the induction assumption the graph F_i' is of the form $T + A$ where T is a forward branching in F_i' rooted at a vertex x and A is a set of essential edges in F_i'. Since every edge of G is redundant, T is an optimal branching in G rooted at x. Furthermore, A is a minimum augmentation for T in G. To see this, fix an edge e in A. Let e' be the edge parallel to e in G. By the definition of G and the fact that e is essential in

F_i', it follows that any minimal augmentation for T in G has a nonempty intersection with $\{e, e'\}$. Thus, the set A constitutes a minimum augmentation for T in G. Hence, $\hat{c}(G, x) \geq i + 1$. Note that G is an essential contraction of F_{i+1}'. By lemma 14 we have $\hat{c}(F_{i+1}', y) \geq i + 1$ for any vertex y in the subgraph of F_{i+1}' collapsed into x.

We now show that the edge set of F_{i+1}' can be partitioned into a forward branching rooted at one such vertex y and a set of essential edges. Let y be the root of the path $P(x)$ used in step 4 of algorithm 9, i.e., y is that representative of x that is labeled 1 in step 4 of algorithm 9. Let B denote the set of edges in F_{i+1}' that also belong to G. By lemma 13 these edges are exactly the redundant edges in F_{i+1}'. It suffices to show that there exists a forward branching in F_{i+1}' rooted at y and containing all the edges of B.

For any vertex $w \neq y$ in F_{i+1}' that has no incoming edge in B, define $e(w)$ to be the unique edge in $P(v)$ whose head is w. Let B' be the set $\{e(w) : w \in V(F_{i+1}'),\ w \neq y$ and w has no incoming edge in $B\}$. We shall now prove that every vertex in F_{i+1}' is reachable from y by edges in $B \cup B'$. Since each vertex in F_{i+1}' other than y has exactly one incoming edge in $B \cup B'$ and y has none, this implies that the edges in $B \cup B'$ form a forward branching in F_{i+1}' rooted at y; hence, F_{i+1}' can be partitioned into a forward branching rooted at y and a set of essential edges.

Let us call a path in F_{i+1}' all of whose edges are in $B \cup B'$ a *good path*. We denote the root of $P(v)$ by $r(v)$ for any v in G (see step 4 of algorithm 9). We first show that there is a good path from y to the root $r(v)$ of $P(v)$ in F_{i+1}' for any v in G. Let $depth(v)$ denote the depth of v in $\hat{T}$ (constructed in step 2 of algorithm 9), i.e., the number of edges on the unique path from y to v in $\hat{T}$. We prove the claim by induction on $depth(v)$. If $depth(v) = 0$, then $v = x$ and $r(v) = y$; hence, the base case holds. Assume inductively that the claim holds if $depth(v) = k$. Now let $depth(v) = k + 1$. Let u be the father of v in $\hat{T}$. By the induction assumption there is a good path from y to the root $r(u)$ of $P(u)$. Let $(w, r(v))$ be the edge in F_{i+1}' corresponding to edge (u, v) of $\hat{T}$. From steps 4, 5, and 6 of algorithm 9 it follows that there is a good path in $P(u)$ from $r(u)$ to w. We can assemble the good paths from y to $r(u)$ and from $r(u)$ to w together with the edge $(w, r(v))$ into a good path from y to $r(v)$ in F_{i+1}'.

Now consider the case where a vertex w in F_{i+1}' lies on the path $P(v)$ of some vertex v in G but is different from the root of $P(v)$. Vertex w is reachable on $P(v)$, using only edges of B', either from the root of $P(v)$ or from a vertex w' on $P(v)$ that has an incoming edge in B. In the former case we are done. In the latter case let w'' be the tail of the edge of B in F_{i+1}' whose head is w'. With the labeling in step 5 of algorithm 9 it follows that w'' is reachable from the root of its path using only edges of B'. We conclude that there is a good path from y to w. $\Box$

Theorem 19 *Algorithm 8 requires $\Omega(\log n)$ iterations in the worst case. Thus, its worst-case time complexity is $\Theta(m + n \log n)$.*

Proof. The graphs F_i' are orientations of the graphs F_i in the sample for 2-edge-connectivity and thus have the same size. With lemma 9 and theorem 18 we have $\hat{c}(F_i') = \Omega(\log n(F_i'))$. One can construct for each $n > 1$ a digraph G on n vertices with $\hat{c}(G) = \Omega(\log n)$ by subdividing edges in the graph F_i' where i is the largest integer with $n(F_i') \leq n$. Since each iteration of algorithm 8 requires time $\Omega(n)$, we infer that algorithm 8 requires $\Omega(m + n \log n)$ time in the worst case. This bound is tight because it is shown in [8] that the algorithm terminates in time $O(m + n \log n)$. $\Box$

We now turn our attention to algorithm 6. For a strongly connected digraph H a *trace for H with respect to x* is a sequence $H_0, H_1, \ldots, H_k$ such that $H_0 = H$, $H_i \neq H_{i+1}$ for $0 \leq i < k$, and H_i $(0 < i \leq k)$ is of the form $T + A$ where T is an optimal branching in H_{i-1} rooted at x and A is a minimal augmentation for T in H_{i-1}. Let $c(H)$ denote the maximum length of a trace for H (maximum taken over all vertices). Since each minimum augmentation trace for H is also a trace for H with respect to the same vertex, theorem 18 implies that $c(F_i') \geq i$. The following theorem is an easy corollary of this fact.

Theorem 20 *Algorithm 6 requires $\Omega(\log n)$ iterations in the worst case. Thus, its worst-case time complexity is $\Theta(m + n \log n)$.* $\Box$

7 Concluding Remarks

In this paper we have given linear-time algorithms for computing a minimal biconnected spanning subgraph and a minimal 2-edge-connected spanning subgraph. We have also provided a general framework for computing minimal spanning subgraphs with respect to various graph properties. These results should be useful in deriving algorithms for computing minimal spanning subgraphs for other graph properties, e.g., k-vertex- or k-edge-connectivity for $k > 2$.

In the context of directed graphs we leave open the question whether there is a linear time algorithm for computing a minimal strongly connected spanning subgraph. We have shown in this paper that several natural algorithms for this problem achieve a worst-case running time of $\Theta(m + n \log n)$. The results in this paper suggest that it may be possible to achieve linear time by combining the basic algorithms with various contraction operations (this approach has worked for

2-edge-connectivity and biconnectivity). Unfortunately we proved in [14] (using a fairly involved construction) that this approach will not improve the worst-case running time if we only collapse cycles and contract chains. A new approach seems necessary.

References

[1] A.V. Aho, J.E. Hopcroft AND J.D. Ullman, *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[2] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications,* North-Holland, 1976.

[3] F. Chung AND R.L. Graham, *Private communication*, 1977; cited in [7].

[4] R. Cole AND U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree, and graph problems,* Proc. 27th Ann. IEEE Symp. on Foundations of Comp. Sci., 1986, pp.478-491.

[5] E. Dahlhaus, M. Karpinski AND P. Kelsen, *An efficient parallel algorithm for finding a maximal independent set in a hypergraph of dimension 3,* Inf. Proc. Letters, vol. 42, pp. 309-313, 1992.

[6] J. Edmonds, *Edge-disjoint branchings*, in "Combinatorial Algorithms", pp.91-96, Algorithmic Press, New York, 1973.

[7] M.R. Garey AND D.S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness,* Freeman, San Francisco, CA, 1979.

[8] P. Gibbons, R.M. Karp, V. Ramachandran, D. Soroker, AND R. Tarjan, *Transitive compaction in parallel via branchings,* J. Algorithms, vol. 12, 1991, pp. 110-125.

[9] M. Goldberg AND T. Spencer, *A new parallel algorithm for the maximal independent set problem,* SIAM J. Computing, vol. 18, 1989, pp.419-427.

[10] X. Han, *An algorithmic approach to extremal graph problems*, Ph.D. Thesis, June 1991, Department of Computer Sciences, Princeton University, Princeton, NJ.

[11] R.M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines,* in J. van Leeuwen, editor, Handbook of Theoretical Computer Science, Vol. A, pp. 869-941, MIT Press/Elsevier, 1990.

[12] R.M. Karp, E. Upfal, AND A. Wigderson, *The complexity of parallel search,* J.C.S.S., vol. 36, 1988, pp.225-253.

[13] P. Kelsen AND V. Ramachandran, *On finding minimal 2-connected subgraphs*, Journal of Algorithms, to appear; extended abstract in *Proceedings of the Second ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1991, pp. 178-187.

[14] P. Kelsen AND V. Ramachandran, *The complexity of finding minimal spanning subgraphs*, Tech. Report TR-91-17, 1991, Department of Computer Sciences, University of Texas, Austin, TX 78712.

[15] J.M. Lewis AND M. Yannakakis, *The node-deletion problem for hereditary properties is NP-complete*, J. C. S. S., vol. 20, 1980, pp.219-230.

[16] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Computing, vol. 15, 1986, pp. 1036-1053.

[17] G. Miller AND J.H. Reif, *Parallel tree contraction and its applications*, Proc. 26th Ann. Symp. on Foundations of Comp. Sci., pp.478-489, 1985.

[18] H. Nagamochi AND T. Ibaraki, *Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph,* Algorithmica, vol. 7, 1992, pp. 583-596.

[19] M. D. Plummer, *On minimal blocks*, Trans. Amer. Math. Soc., vol. 134, 1968, pp.85-94.

[20] V. Ramachandran, *Fast parallel algorithms for reducible flow graphs*, Concurrent Computations: Algorithms, Architecture and Technology, S.K. Tewksbury, B.W. Dickinson and S.C. Schwartz, ed., Plenum press, New York, NY, 1988, pp.117-138; see also *Fast and processor-efficient parallel algorithms for reducible flow graphs*, Tech. Report ACT-103, November 1988, Coordinated Science Laboratory, University of Illinois, Urbana, Illinois, IL 61801.

[21] V. Ramachandran, *Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity*, in *Synthesis of Parallel Algorithms,* J. Reif, ed., Morgan-Kaufmann, 1993.

[22] R. Tarjan, *Depth first search and linear graph algorithms*, SIAM J. Computing, vol. 1, 1972, pp.146-160.

[23] W. Tutte, *Graph theory*, Addison-Wesley, 1984.

[24] M. Yannakakis, *Node- and edge-deletion NP-complete problems*, Proc. 10th Ann. ACM Symp. on Theory of Computing, New York, 1978, pp. 253-264.