COMPUTING MINIMAL SPANNING SUBGRAPHS IN LINEAR TIME

Xiafeng Han
Pierre Kelsen
Vijaya Ramachandran
Robert Tarjan

CS-TR-356-91

December 1991

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 two related algorithms for this problem and prove their correctness under some rather weak assumptions about P.

We devise a general technique for analyzing the worst-case behavior of these algorithms. By applying the technique to 2-edge-connectivity and biconnectivity, we obtain a tight lower bound of $\Omega(m + n \log n)$ on the worst-case sequential running time of the above algorithms for these properties; this resolves open questions posed earlier with regard to these properties. We then give refinements of the basic algorithms that yield the first linear-time algorithms for finding a minimal 2-edge-connected spanning subgraph and a minimal biconnected spanning subgraph of a graph.

[*]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 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.

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 both in the parallel and sequential 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 ([1], [3]) proved that the problems of finding a minimum k-vertex-connected or k-edge-connected spanning subgraph are NP-hard for any fixed $k \geq 2$. Yannakakis ([18]; see also [11]) 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 and parallel algorithms for the problem.

The problem at hand may be phrased in the very general framework of *independence systems* described by Karp, Upfal, and Wigderson ([7]): 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 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 ([12],[5], [2], [8]). 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 (or *transitive compaction* [4]) and for 2-edge-connectivity and biconnectivity ([9]). For these problems algorithms are given in ([4], [9]) 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 the same 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 section 3.1 of this paper we generalize the high-level algorithm of ([4],[9]) into two general algorithms for finding a minimal spanning subgraph in an undirected graph with a given property. We show that for connectivity requirements the running time of a variant 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 ([16]) and in parallel ([13], [14]), this algorithm provides a useful paradigm for the sequential and parallel determination of minimal spanning subgraphs with respect to connectivity requirements.

In section 3.2 we analyze the worst-case complexity of these algorithms. We give a tight lower bound of $\Omega(\log n)$ on the worst-case number of iterations of the algorithm for 2-edge-connectivity and biconnectivity; this leads to an $\Omega(m+n \log n)$ lower bound on the sequential running time of these algorithms, thus settling open questions posed in [9]. The technique we use to derive these bounds is fairly general and may be applicable to other graph properties. We strengthen the bound for 2-edge-connectivity in section 3.3 by showing that it holds even if we allow certain contractions.

In section 4 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 at each iteration. This result also reduces the work performed by the parallel algorithms for these problems by a logarithmic factor.

In the next section we define the graph-theoretic terms and concepts that are used in this paper.

Note: An extended abstract of the present paper will appear in the Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (Orlando, FL, 1992).

2 Definitions

A *graph* $G = (V, E)$ consists of a set V of *vertices* and a set E of *edges*. We also write $V(G)$ and $E(G)$ for the set of vertices and the set of edges, respectively, of G. We use $n(G)$ and $m(G)$ to denote the number of vertices and edges, respectively, in G. If the edges are ordered pairs (v, w) of distinct vertices, the graph is *directed*: v is called the *tail* and w is called the *head* of the edge. If the edges are unordered pairs of distinct vertices, also denoted by (v, w), the graph is *undirected*. In that case v and w are *incident* with the edge (v, w) and (v, w) *covers* the two vertices v and w. If E is a multiset, i.e., if every edge may occur several times, then G is a *multigraph*. The *degree* of v in G, denoted by $deg_G(v)$, is the number of edges of G covering v.

Fix a multigraph $G = (V, E)$. A *path* P in G is a sequence $< v_0 \ldots v_k >$ of vertices of V such that (v_{i-1}, v_i) $(1 \leq i \leq k)$ is an edge in E; the nodes v_0 and v_k are the *endpoints* of P and $v_1 \ldots v_{k-1}$ are the *internal vertices* of P. The vertices v_i *lie on the path* P and the edges (v_{i-1}, v_i) are the *edges on the path* P. A path is *simple* if $v_0 \ldots v_{k-1}$ are distinct and $v_1 \ldots v_k$ are distinct. A *cycle* in G is a path in G whose endpoints coincide and all of whose edges are distinct. A *simple cycle* is a simple path whose endpoints coincide.

If $G = (V, E)$ and $G' = (V', E')$ are two graphs such that $V' \subseteq V'$ and $E' \subseteq E$, then G' is a *subgraph* of G. The graph G' is the subgraph of G *induced* by the vertices in V' if E' contains exactly the edges of E between vertices of V'. A spanning subgraph of G is a subgraph G' with $V' = V$. If G' is a spanning subgraph of G and $E'' \subseteq E$, then $G' + E''$ denotes the graph with vertex set $V'(= V)$ and edge set $E' \cup E''$ and $G' - E''$ denotes the graph with vertex set V' and edge set $E' - E''$.

An undirected graph G is *connected* if there exists a path between any two distinct vertices in G. A *connected component* in G is a maximal connected subgraph of G (i.e., it is not a proper subgraph of a connected subgraph of G). A *tree* is a connected graph without cycles.

The graph G is *2-edge-connected* if every pair of distinct vertices is connected by two edge-disjoint paths or, equivalently, if every edge lies on a cycle. A *2-edge-connected component* of G is a maximal 2-edge-connected subgraph of G. A *cutedge* in G is an edge in G whose removal increases the number of connected components in G. A graph is *k-edge-connected* $(k > 0)$ if every pair of distinct vertices is connected by at least k edge-disjoint paths. A vertex in G is a *cutpoint* if removing it together with all incident edges increases the number of connected components in G. 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 subset of vertices in G that induces a connected subgraph that has no cutpoint and that is maximal with this property. A graph is k-vertex-connected if it is the complete graph on $k+1$ vertices or it has at least $k+2$ vertices and it cannot be disconnected by removing fewer than k vertices.

Let $G = (V, E)$ be an arbitrary graph. Let $V' \subseteq V$. The operation of *collapsing the vertices of* V' consists of replacing all vertices in V' by a single new vertex v, deleting all edges in G whose two endpoints are in V' and replacing each edge (x, y) with x in V' and y in $V - V'$ by an edge (v, y). In general the resulting graph is a multigraph even if the original graph is not a multigraph.

An *ear decomposition* ([15]) $D = [P_0, P_1, \ldots, P_{r-1}]$ of an undirected graph $G = (V, E)$ is a partition of E into an ordered collection of edge disjoint simple paths $P_0, \ldots P_{r-1}$ such that P_0 is an edge, $P_0 \cup P_1$ is a simple cycle, and each endpoint of P_i, for $i > 1$, is 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*. D is an *open ear decomposition* if none of the P_i is a simple cycle. A *trivial ear* is an ear containing a single edge.

3 Computing a Minimal Spanning Subgraph

3.1 Two General Algorithms

In this section we describe two closely related algorithms for finding a minimal spanning subgraph of a graph for various properties of undirected graphs. Algorithm 1 was first described in [10] while algorithms 2 and 3 are generalizations of algorithms for finding a minimal 2-edge-connected and minimal biconnected spanning subgraph given in [6].

We allow self-loops and multiple edges in our graphs. 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 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 one at a time; remove an edge if it is redundant in the current graph. By observation 1 the resulting subgraph is minimal.

The following algorithm is a generalization of algorithms given in [9] and [4] (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 as 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 G. Therefore, the number of redundant edges decreases by at least one at each iteration and algorithm 1 terminates properly. []

For steps (2) and (2.1) of algorithm 1, the essential and redundant edges of H need to be computed. In general, it is not clear whether computing these edges is easier than the original problem of finding a minimal spanning $P-$subgraph. It turns out, however, that one can avoid this computation by gradually building up a set of essential edges. Algorithm 2 below uses this strategy.

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

```
(1) $H := G$; $Ess := \emptyset$; $C := E(H)$;
(2) While $C \neq \emptyset$, do:
    (2.1) Compute a spanning tree $T_H$ in $H$ with a maximum number of edges of $Ess$;
    (2.2) Compute a minimal $A \subseteq E(H)$ such that $T_H + A$ has property $P$;
    (2.3) $H := T_H + A$; $Ess := Ess \cup A \cup \{cutedges\ in\ H\}$; $C := E(H) - Ess$.
```

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

Proof. An induction on the iteration number shows that H, as computed in step (2.3) of algorithm 2, is a spanning $P-$subgraph of H at any iteration of algorithm 2 and, at the end of any iteration of algorithm 2, all redundant edges of H are contained in C. Thus, upon termination H is a minimal spanning P-subgraph of H. To prove termination, fix an iteration of the while-loop. If not all edges of C belong to T_H (computed in step (2.1)), then those edges of C that lie outside T_H are either added to Ess or excluded from H in step (2.3) of the current iteration. If the edges of C are all part of T_H, then each edge of C is a cutedge in $T_H + A$ since the tree T_H contains a maximum number of edges of Ess. We conclude that in each case $| C |$ decreases by at least 1 at this iteration, thus implying termination. []

By the proofs of theorem 1 and theorem 2 the number of iterations of algorithm 1 and algorithm 2 are both bounded by the number of edges in the input graph. For several graph properties much sharper bounds hold. In this paper we shall look at different connectivity requirements. For the remainder of this section we only consider properties P that imply 2-edge-connectivity. The following theorem shows that for these properties algorithm 1 terminates quickly.

Theorem 3 *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 algorithm 1. An *essential component* in H is a maximal subgraph of H having a spanning tree of essential edges. Let r denote the number of redundant edges in H and c the number of essential components in H. Fix a tree T in H with a maximum number of essential edges of H. 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 in each iteration of algorithm 1. The claim follows. []

Corollary 1 *Algorithm 1 computes a minimal k-vertex-connected or k-edge-connected spanning subgraph in $O(\log n)$ iterations for any $k \geq 2$. []*

The previous bound does not hold for algorithm 2: consider the case where the input graph G is a simple cycle on n vertices and n edges. Each iteration of algorithm 2 finds a tree of $n-1$ edges and adds exactly one edge to Ess. Thus, algorithm 2 may require n iterations of the while-loop.

The following observation points out a way of making algorithm 2 faster: if H is a 2-edge-connected graph and V' is a proper subset of vertices in H, then at least two edges in H connect a vertex in V' to a vertex outside V'. If there are exactly two such edges, then they are both P-essential in H.

Fix a P-graph H and a subset Ess of the set of essential edges in H. An edge of H is called *critical* (with respect to Ess) if it is one of exactly two edges connecting some connected component of $(V(H), Ess)$ with the set of vertices outside this component. By the previous observation any critical edge is essential in H (provided P implies 2-edge-connectivity).

Modify algorithm 2 by adding in step (2.3) to Ess the edges of A as well as the critical edges in H (since H is 2-edge-connected, it does not contain any cutedges). We refer to the modified algorithm as algorithm 3.

Lemma 1 *Let* $\mid C \mid= q$ *at the beginning of an iteration of algorithm 3. Then, at the beginning of the next iteration* $\mid C \mid\leq 3q/4$.

Proof. Fix H at the beginning of an iteration of the while-loop in algorithm 3. The claim certainly holds if the number of edges of C in tree T_H (computed in step (2.1) of the current iteration) is less than $3q/4$. Now assume at least $3q/4$ edges of C belong to T_H. Construct H_c from H by collapsing the vertex sets of the connected components of $(V(H), Ess)$ in H. Let a be the number of vertices of degree 2 in H_c. Let p be the total number of vertices in H_c; note that $p > 3q/4$. We have $\sum_{v\in V(H_c)} deg_{H_c}(v) \geq 2a+3(p-a)$ and $\sum_{v\in V(H_c)} deg_{H_c}(v) \leq 2q$; hence $a \geq 3p-2q$. Since $p > 3q/4$, this implies $a > q/4$. Therefore, the number of critical edges is at least $q/4$. The claim now follows by noting that none of the critical edges in H will belong to C at the beginning of the next iteration of algorithm 3. []

Corollary 2 *If P satisfies $(C1)$ and $(C2)$ and also implies 2-edge-connectivity, then algorithm 3 terminates after $O(\log n)$ iterations of the while-loop. []*

Corollary 3 *Algorithm 3 computes a minimal k-vertex-connected or k-edge-connected spanning subgraph in $O(\log n)$ iterations for any $k \geq 2$. []*

For the special case $k = 2$ the minimal augmentation (step (2.2) of algorithm 3) requires only linear time (see [9]) . Thus, if the number of iterations of algorithm 3 is constant, then algorithm 3 runs in linear time. In the next section we shall rule out this possibility by showing that the $O(\log n)$ bound in corollary 3 is tight for $k = 2$.

Because of its simple structure algorithm 1 will be a more convenient vehicle for proving lower bounds. Fortunately, it turns out that 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 2 *Let H be a P-graph, Ess a subset of the essential edges in H, T an optimal tree in H 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 Ess 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 Ess. 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 with a maximum number of edges of Ess. We have $E(T') \subseteq E(T + A)$. Moreover, the edges of $T + A$ that do not belong to T' form a minimal augmentation for T' in H. []

A *trace* for P-graph H is a sequence $H_0, H_1, \ldots, H_l$ of subgraphs such that $H_0 = H$, $H_i \neq H_{i+1}$ for $i \geq 0$, 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.

Corollary 4 *Algorithm 3 requires at least as many iterations as algorithm 1 in the worst case.*

Proof. Fix a trace $H_0, H_1, \ldots, H_l$ of maximum length for H. Let $E_0, E_1, \ldots, E_l$ be a sequence of sets such that E_i is a set of essential edges in H_i for $0 \leq i \leq l$. By lemma 2 each H_i $(i > 0)$ can be written as $T_i + A_i$ where T_i contains a maximum number of edges of E_{i-1} and A_i is a minimal augmentation for T_i in H_{i-1}. Hence, there is an execution of algorithm 3 that needs at least l iterations on graph H. []

By corollary 4 it suffices to derive lower bounds for the number of iterations of algorithm 1. We shall do so in the next section.

3.2 Worst-Case Analysis

We start out with a few definitions. The *P-complexity* of P-graph H is the maximum length of a trace for H. We denote the P-complexity of P-graph H by $c_P(H)$. We call an infinite sequence of graphs $H_0, H_1, H_2, \dots$ such that $c_P(H_i) \geq i$ for all $i \geq 0$ a *sample* for P.

Note that if we do not impose any additional restrictions on P, there may be no sample. For instance, let P denote connectedness (which satisfies ($C1$) and ($C2$)). In this case every graph has P-complexity 1. We specify two more constraints on P that will guarantee the existence of a sample for P. By analyzing how fast the size of the graphs in the sample grows as a function of the P-complexity, we shall derive lower bounds on the number of iterations required by algorithm 1 for 2-edge-connectivity. Although biconnectivity does not satisfy both constraints, we shall prove that a special property of the sample constructed for 2-edge-connectivity guarantees that it is a sample for biconnectivity as well.

A *contraction of a graph* H (that may not have property P) is obtained from H by collapsing disjoint subsets of vertices of H whose induced subgraphs in H have property P; we refer to a subgraph of H induced by a collapsed subset as a *collapsed subgraph.* Graph H' is an *essential contraction* of a P-graph H if H' is a contraction of H and the edges in the collapsed subgraphs are P-essential in H.

Caveat: If graph H' is obtained from graph H by collapsing disjoint subsets of vertices of H, each edge of H' corresponds to a unique edge of H. To keep the notation compact, we often shall not distinguish between an edge of H' and the corresponding edge of H. For instance for $S \subseteq E(H')$ we use $H - S$ to denote the graph obtained by removing from H those edges corresponding to edges of S.

The following condition states that graph property P is closed under contractions.

> ($C3$) Let H' be a contraction of a graph H. Then, H has property P iff H' has property P.

We note the following important consequence of ($C3$).

Lemma 3 *If H' is an essential contraction of P-graph H, then $c_P(H') \leq c_P(H)$.*

Proof. : Fix a P-graph H and an essential contraction H' of H. First, note that ($C3$) implies that H' has property P; therefore the P-complexity of H' is well-defined. We prove the lemma by induction on the P-complexity of H'. If $c_P(H') = 0$, then certainly $c_P(H) \geq c_P(H')$.

Let $c_P(H') = k > 0$. Let $T' + A'$ be a graph of P-complexity $k - 1$ where T' is an optimal tree in H' and A' is a minimal augmentation for T' in H'. Since each collapsed subgraph in H is connected (condition ($C2$)), we can combine T' with spanning trees for the collapsed subgraphs to form a spanning tree T of H; the tree T is an optimal tree in H. Let A be the edges of A' plus the collapsed edges of H that are not in T. The graph $T'+A'$ is a contraction of $T+A$. By $(C3)$ $T+A$ is a P-graph and $T'+A'$ is indeed an essential contraction of $T+A$. By the induction assumption $c_P(T+A) \geq k-1$. Moreover, by $(C3)$ and the definition of H', the edges of A are essential in $T+A$. We conclude that $c_P(H) \geq k$. []

The last constraint is rather technical.

> $(C4)$ For any nonempty and finite set S there exists a P-graph $G_S = (S \cup S', E)$ such that the edges of G_S are essential in any P-graph $G' = (V', E')$ with $S \cup S' \subseteq V'$, $E \subseteq E'$ (i.e., G' contains G_S as a subgraph), and such that no edge of $E' - E$ is incident with a vertex in S'.

We call the graph G_S a *gadget for* S. As an example, consider 2-edge-connectivity. Let $S = \{v_1, \dots v_s\}$ and let $S' = \{v'_1, \dots, v'_s\}$. The cycle alternating between the vertices of S and those of S' is a gadget for S. To see this, note that if G' is a 2-edge-connected graph containing this cycle as a subgraph and such that no edges other than those of the cycle are incident with vertices of S', then the nodes of S' all have degree 2 in G'; hence, all edges of the cycle are essential in G'.

Algorithm 4 below provides a means of generating a sample for any graph property P satisfying conditions $(C1)$-$(C4)$. A *linear graph* is a P-graph whose edge set can be partitioned into a spanning tree and a set of essential edges.

Algorithm 4: *Increasing the P-complexity of a graph.*

Input Linear graph G of P-complexity k.

Output Linear graph G' of P-complexity $\geq k+1$.

(1) Add edges to G so that all edges in the resulting graph G_1 are P-redundant (e.g., by doubling all essential edges).

(2) For each vertex v number the incident edges in G_1 as $e_1, \dots, e_d$ (d=degree of v in G_1).

(3) Construct G' from G_1 as follows: for each v in G_1 of degree d create d new vertices $v_1, \dots, v_d$ in G'; we call these vertices the *representatives for* v. For each edge (u, v) in G_1 that is the *ith* edge incident on u and the *jth* edge incident on v (in G_1), add an edge (u_i, v_j) to G'. Finally, for each vertex v of G_1, add to G' a gadget for the

representatives of v whose vertex set is the set of representatives plus a collection of new vertices (these collections are disjoint for different gadgets).

Theorem 4 *G' is a linear graph of P-complexity at least $k+1$ provided P satisfies conditions $(C1)$-$(C4)$.*

Proof. Since input graph G is linear, it is of the form $T + A$ where T is a spanning tree in G and A is a set of essential edges in G. Since every edge of G_1 is redundant, T is an optimal tree in G_1. Hence $c_P(G_1) \geq k+1$.

Graph G_1 is a contraction of G'. Hence, by $(C3)$, G' is a P-graph. By condition $(C4)$ and the definition of a gadget, G_1 is an essential contraction of G'. Since $c_P(G_1) \geq k+1$, lemma 3 gives us $c_P(G') \geq k+1$. To see that G' is linear, note that we obtain a spanning tree for G' by combining the edges of G' corresponding to edges of G_1 with a subset of the edges in the gadgets. By $(C4)$ all remaining edges are essential in G'. []

We shall now apply the above results to a concrete graph property: 2-edge-connectivity.

Lemma 4 *2-edge-connectivity satisfies conditions $(C1)$-$(C4)$.*

Proof. Conditions $(C1)$ and $(C2)$ are immediate from the definition of 2-edge-connectivity. For $(C3)$, let H' be a contraction of H. First, note that H is connected iff H' is connected (with condition $(C2)$). Call an edge occurring both in H and H' an *external edge.* If H is 2-edge-connected, every external edge lies on a cycle in H. This cycle translates into a cycle in H' containing the same external edge. Therefore, H' is 2-edge-connected. Conversely, if H' is 2-edge-connected any external edge is on a cycle in H' and that cycle yields a cycle in H containing the same external edge. Since each collapsed subgraph is 2-edge-connected, every edge of H is on a cycle and hence H is 2-edge-connected. For $(C4)$ fix a set $S = \{v_1, \ldots, v_s\}$. Let $v'_1 \ldots v'_s$ be s new vertices. The cycle $v_1 v'_1 v_2 v'_2 \ldots v_s v'_s v_1$ is a gadget for S. []
By theorem 4 and lemma 4, algorithm 4 may be used to construct a sample for 2-edge-connectivity. We fill in the details for steps (1) and (3) of algorithm 4. We start the construction of the sample with graph F_0 consisting of two vertices with two parallel edges between them. The graph F_0 is shown in figure 1. Assume inductively that we have constructed F_{i-1}. In step (1) of algorithm 2 we double the essential edges in F_{i-1}. In step (3) we

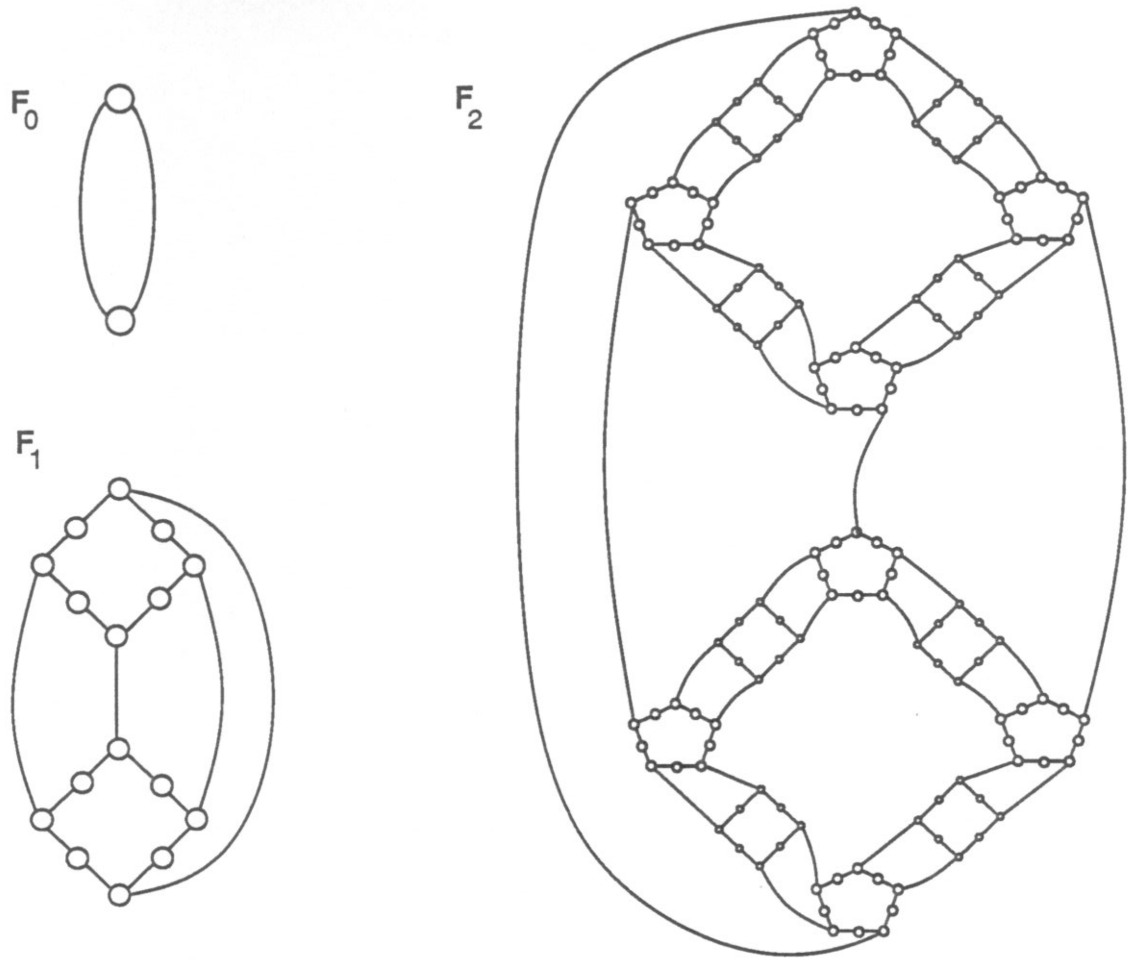

Figure 1: F_0, F_1, and F_2.

choose as gadget for $\{v_1, \ldots, v_d\}$ the cycle $v_1, v_1', v_2, v_2', \ldots, v_d, v_d', v_1$, where $\{v_1', v_2', \ldots, v_d'\}$ is a set of d new vertices. The first three graphs F_0, F_1, and F_2 in a possible sample are given in figure 1. Note that we may obtain different samples for different edge numberings (step (2) of algorithm 4) but this does not affect the the size of the graphs in the sample. In the sequel F_i denotes a graph in a fixed sample for 2-edge-connectivity constructed according to the above rules.

Lemma 5 *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 $i > 0$. []

Corollary 5 *For 2-edge-connectivity, there exists a function $f(n) = \Omega(\log n)$ such that there is a graph on n vertices of P-complexity $f(n)$ for any $n \geq 1$.*

Proof. To construct a graph of P-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 by repeatedly subdividing an essential edge. []

Let us now turn our attention to biconnectivity. Unfortunately, biconnectivity does not satisfy condition $(C3)$. We do however have the following well-known result.

Lemma 6 *If G is a graph with at least three vertices in which each vertex has degree ≤ 3, then G is biconnected iff G is 2-edge-connected.*

Proof. The 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. []

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

Corollary 6 *If a graph H is a $3-$ graph, then $c_P(H) \leq c_{P'}(H)$.*

Proof. By induction on $c_P(H)$. The induction base is clear with lemma 6. 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 6, an edge in H is P-redundant iff 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 $c_{P'}(H) \geq k$. []
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 $i \geq 1$ with respect to biconnectivity. Thus, we get the following result.

Corollary 7 *For biconnectivity, there exists a function $f(n) = \Omega(\log n)$ such that there is a graph on n vertices with P-complexity $f(n)$ for any $n \geq 3$.*

Proof. Similar to proof of corollary 5. []

Corollaries 5 and 7 establish that Algorithm 1 takes $\Omega(\log n)$ iterations for 2-edge-connectivity and biconnectivity in the worst-case. By corollary 4 the same bound holds for algorithm 3. Since each iteration of these algorithms takes $\Omega(n)$ time we get an $\Omega(m+n\log n)$ lower bound for the sequential running time of these algorithms. A similar lower bound is given in [10] for an algorithm from [4] that finds a minimal strongly connected spanning subgraph in a directed graph.

3.3 Avoiding Cycles

When constructing graph F_{i+1} from graph F_i in the sample for 2-edge-connectivity, algorithm 4 expands each node of F_i into a 2-edge-connected subgraph of essential edges. Thus, one may attempt to speed up algorithm 3 (for P=2-edge-connectivity) by collapsing 2-edge-connected subgraphs consisting of essential edges. Below we show that this modification does not improve the asymptotic worst-case running time.

Let H be a 2-edge-connected graph and let $E(H) = Ess \cup C$ where Ess is a set of essential edges in H. An *Ess-component of H* is a 2-edge-connected component of $(V(H), Ess)$. The operation of *shrinking an Ess-component* in H consists of collapsing in H the vertex set of this component.

Modify algorithm 3 (for P=2-edge-connectivity) as follows: at the end of the while-loop collapse the Ess-components of H. The modified algorithm, referred to as algorithm 3′, is given below.

Algorithm 3': *Computing a minimal 2-edge-connected spanning subgraph of G.*
Input P-graph G.
Output Minimal 2-edge-connected spanning subgraph of G.

```
(1) $H := G$; $Ess := \emptyset$; $C := E(H)$; $R := \emptyset$;
(2) While $C \neq \emptyset$, do:
    (2.1) Compute a spanning tree $T_H$ in $H$ with a maximum number of edges of $Ess$;
    (2.2) Compute a minimal $A \subseteq E(H)$ such that $T_H + A$ is 2-edge-connected.
    (2.3) $R := R \cup (E(H) - E(T_H + A))$;
```

```
        (2.4) $H := T_H + A$; $Ess := Ess \cup A \cup$ {critical edges in $H$}; $C := E(H) - Ess$;
        (2.5) $R := R \cup$ { edges in $C$ whose endpoints belong to same $Ess$-component };
        (2.6) Shrink the $Ess$-components in $H$; call the new graph $H$. Remove those edges
              from $Ess$ and $C$ that are not in $H$.
    (3) Return graph $G - R$.
```

We now establish the correctness of this algorithm.

Lemma 7 *Assume that H' is constructed by shrinking some Ess-components in H. If H'' is a minimal 2-edge-connected spanning subgraph of H' and S is the set of redundant edges of H that do not correspond to edges in H'', then $H-S$ is a minimal 2-edge-connected spanning subgraph of H. []*

Proof. Observe that $H'' = H' - S$ is a contraction of $H - S$. Since 2-edge-connectivity satisfies ($C3$) (see lemma 4), the graph $H - S$ is 2-edge-connected. To prove that $H - S$ is minimal, fix an edge $e \in E(H) - S$. If e is essential in H, then e is also essential in $H - S$ by observation 1. Otherwise e is an edge in H''. Since $H'' - e$ is a contraction of $H - S - e$ and is not 2-edge-connected, it follows that $H - S - e$ is not 2-edge-connected and hence e is essential in $H - S$. Thus, $H - S$ is a minimal 2-edge-connected spanning subgraph of H. []

Theorem 5 *Algorithm 3′ returns a minimal 2-edge-connected spanning subgraph of the input graph G.*

Proof. We first show that the following two statements hold at the beginning of any iteration of algorithm 3′:

(1) $H = Ess \cup C$;

(2) The edges of Ess are essential in H.

Note that both (1) and (2) hold before the first iteration of the while-loop. Assume inductively that they hold at the beginning of iteration i. Statement (1) holds after step (2.4) of iteration i. By the definition of step (2.6) it also holds at the beginning of iteration $i+1$.

Let Ess_i and H_i denote the set Ess and the graph H at the beginning of iteration i. Let T_i and A_i denote the tree and augmentation found during iteration i of algorithm 3′. Any edge of $e \in Ess_{i+1} \cap Ess_i$ is essential in H_i and hence (by the inductive assumption) in $T_i + A_i$. The graph H_{i+1} is a contraction of $T_i + A_i$. By condition (C3) e is essential in H_{i+1}. If $e \in Ess_{i+1} - Ess_i$, then e is essential in $T_i + A_i$ and, by the previous argument, it is essential in H_{i+1}. Thus, we have established that the edges of Ess_{i+1} are essential in H_{i+1}.

The termination of algorithm 3′ follows from the proof of theorem 2. We shall now prove that $G - R$, returned as output in step (3), is a minimal 2-edge-connected spanning subgraph of G. Denote by R_i the set of edges that are added to R during iteration i or later. Assume algorithm 3′ terminates after k iterations of the while-loop, i.e, at the beginning of iteration $k+1$ we have $C = \emptyset$. We prove by induction on $k+1-i$ that $H_i - R_i$ is a minimal 2-edge-connected spanning subgraph of H_i for $0 \leq i \leq k+1$. The base case $i = k+1$ holds since statements (1) and (2) above imply that every edge of H_{k+1} is in Ess_{k+1} and is therefore essential in H_{k+1}. Assume that the claim holds for $i > j$. Let $R^{(1)}$ and $R^{(2)}$ denote the sets of edges added to R at step (2.3) and (2.5), respectively, of iteration $j+1$ of algorithm 3′. By the induction assumption $H_{j+1} - R_{j+1}$ is a minimal 2-edge-connected spanning subgraph of H_{j+1}. Since H_{j+1} is a contraction of $T_j + A_j$ (and the edges in $R^{(2)} \cup R_{j+1}$ are redundant in $T_j + A_j$), we infer with lemma 7 that $T_j + A_j - R^{(2)} - R_{j+1}$ is a minimal 2-edge-connected spanning subgraph of $T_j + A_j$ and hence of H_j. By noting that $T_j + A_j = H_j - R^{(1)}$ we conclude that $H_j - R_j$ is a minimal 2-edge-connected spanning subgraph of H_j as required. By setting $j = 0$ we see that $G - R$ is a minimal 2-edge-connected spanning subgraph of G. []

The obvious question is: does algorithm 3′ run in linear time on any input graph. We answer this question negatively by constructing graphs with the property that if we run algorithm 3′ on them, no cycle of essential edges (and hence no cycle of edges in Ess) will be created at any intermediate stage. To describe the construction, we need the concept of a *chain*.

A *chain in* H is a path in H 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 cannot be extended. The length of the chain is the number of edges it contains. The operation of *contracting a chain in* H consists of collapsing the set of internal nodes of the chain in H.

Lemma 8 *Let H' be obtained from a graph H that need not be 2-edge-connected by contract-*

ing edge-disjoint chains. Then, H' is 2-edge-connected iff H is 2-edge-connected.

Proof. Every cycle D in H yields a cycle D' in H' containing those edges in H' that correspond to edges of D. Conversely, a cycle D' in H' yields a cycle D in H containing the edges of H corresponding to edges in D'; moreover, if D' includes a collapsed vertex, then D contains all edges on the corresponding chain in H. The claim follows.[]

Corollary 8 *Assume that H' is constructed by contracting edge-disjoint chains in H. If H'' is a minimal 2-edge-connected spanning subgraph of H' and S is the set of redundant edges of H that are not in H'', then $H - S$ is a minimal 2-edge-connected spanning subgraph of H.*

Proof. The graph H'' can be obtained by contracting essential chains in $H - S$. Thus, by lemma 8 $H - S$ is 2-edge-connected. Moreover, for any redundant edge $e \in E(H) - S$ the graph $H'' - e$ can be constructed by contracting essential chains in $H - S - e$. Since H'' is a minimal 2-edge-connected spanning subgraph of H', $H'' - e$ is not 2-edge-connected and hence, by lemma 8, $H - S - e$ is not 2-edge-connected. We conclude that $H - S$ is a minimal 2-edge-connected spanning subgraph of H.[]

For a 2-edge-connected graph H let $c^*(H)$ be the maximum integer k for which there are graphs $H_0 \dots H_k$ with the following properties: $H_0 = H$ and H_i $(0 < i \leq k)$ 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}, and finally, H_i does not contain an essential cycle for $i < k$. The following result is another corollary of lemma 8.

Lemma 9 *If H' is obtained from H by contracting (edge-disjoint) chains, then $c^*(H) = c^*(H')$.*

Proof. We prove $c^*(H) \leq c^*(H')$ by induction on $c^*(H)$. With the proof of lemma 8 we see that the claim holds for the case $c^*(H) = 0$: in this case both H and H' contain an essential cycle. Assume that the claim holds for $c^*(H) < i$. Let $c^*(H) = i$. Hence, H contains an optimal spanning tree T and a minimal augmentation A for T such that $c^*(T + A) = i - 1$. Since H has no essential cycle, T contains all essential edges of H. From T we can construct a spanning tree T' in H' that contains all essential edges of H' and hence is optimal in H'. Moreover, the set A' of edges of H' corresponding to the edges in A forms a minimal augmentation for T' in H'. The graph $T' + A'$ can be obtained by contracting essential chains in $T + A$. By the inductive assumption $c^*(T' + A') \geq i - 1$ and hence $c^*(H') \geq i$. A similar argument shows that $c^*(H) \geq c^*(H')$ whenever H' is constructed from H by contracting chains.[]

We construct a sequence of graphs $Q_0, Q_1, \ldots$ such that $c^*(Q_i) \geq i$ as follows: Q_0 is the graph consisting of two parallel edges (actually, any other minimal 2-edge-connected graph could be chosen instead). Q_{i+1} is constructed from Q_i as follows: for each essential edge $e = (u, v)$ in Q_i we create new vertices $u_1^e, u_2^e, u_3^e, u_4^e$ and $v_1^e, v_2^e, v_3^e, v_4^e$; we replace the edge (u, v) in Q_i with the edges $(u, u_1^e), (u_1^e, u_2^e), (u_2^e, u_3^e), (u_3^e, u_4^e)$, $(v, v_1^e), (v_1^e, v_2^e), (v_2^e, v_3^e), (v_3^e, v_4^e)$, and $(u_4^e, v_2^e), (u_2^e, v_4^e), (u_4^e, v_4^e)$. We denote the resulting graph by Q_{i+1}. Figure 2 shows Q_0, Q_1, and Q_2.

Theorem 6 *For all $i \geq 0$, $c^*(Q_i) \geq i$.*

Proof. By induction on i. The claim trivially holds for $i = 0$. Assume that it holds for any $j \leq i$. We claim that $c^*(Q_{i+1}) \geq i+1$. The graph Q obtained by removing, for each essential edge $e = (u, v)$ in Q_i, the edges (u_2^e, v_4^e) and (u_4^e, v_2^e) from Q_{i+1} is of the form $T + A$ where T is an optimal tree in Q_{i+1} and A is a minimal augmentation for T in Q_{i+1}. We further note that Q_i can be obtained from Q by contracting edge-disjoint chains. With lemma 9 and the induction assumption we find that $c^*(Q) \geq i$ and hence $c^*(Q_{i+1}) \geq i + 1$. []

Corollary 9 *There exists a function $f(n) = \Omega(\log n)$ such that there is a graph G_n on n vertices with $c^*(G_n) = f(n)$ for any $n \geq 1$.*

Proof. First note that there is a constant $c > 0$ such that $n(Q_i) < c^i$ for any $i \geq 0$. Now fix n. Let $i = max\{j : n(Q_j) \leq n\}$. Construct G_n from Q_i by subdividing essential edges in Q_i. By lemma 9 $c^*(G_n) = c^*(Q_i)$ and with theorem 6 we see that $f(n) = c^*(G_n) \geq i$. Since $n(Q_{i+1}) > n$ we have $c^{i+1} > n$ and hence $f(n) = \Omega(logn)$.[]

With lemma 2 it follows from corollary 9 that algorithm 3′ requires $\Omega(m + n \log n)$ time in the worst case (for 2-edge-connectivity). In the next section we shall show that a variant of algorithm 3 that shrinks Ess-components in H *and* contracts chains runs in linear time.

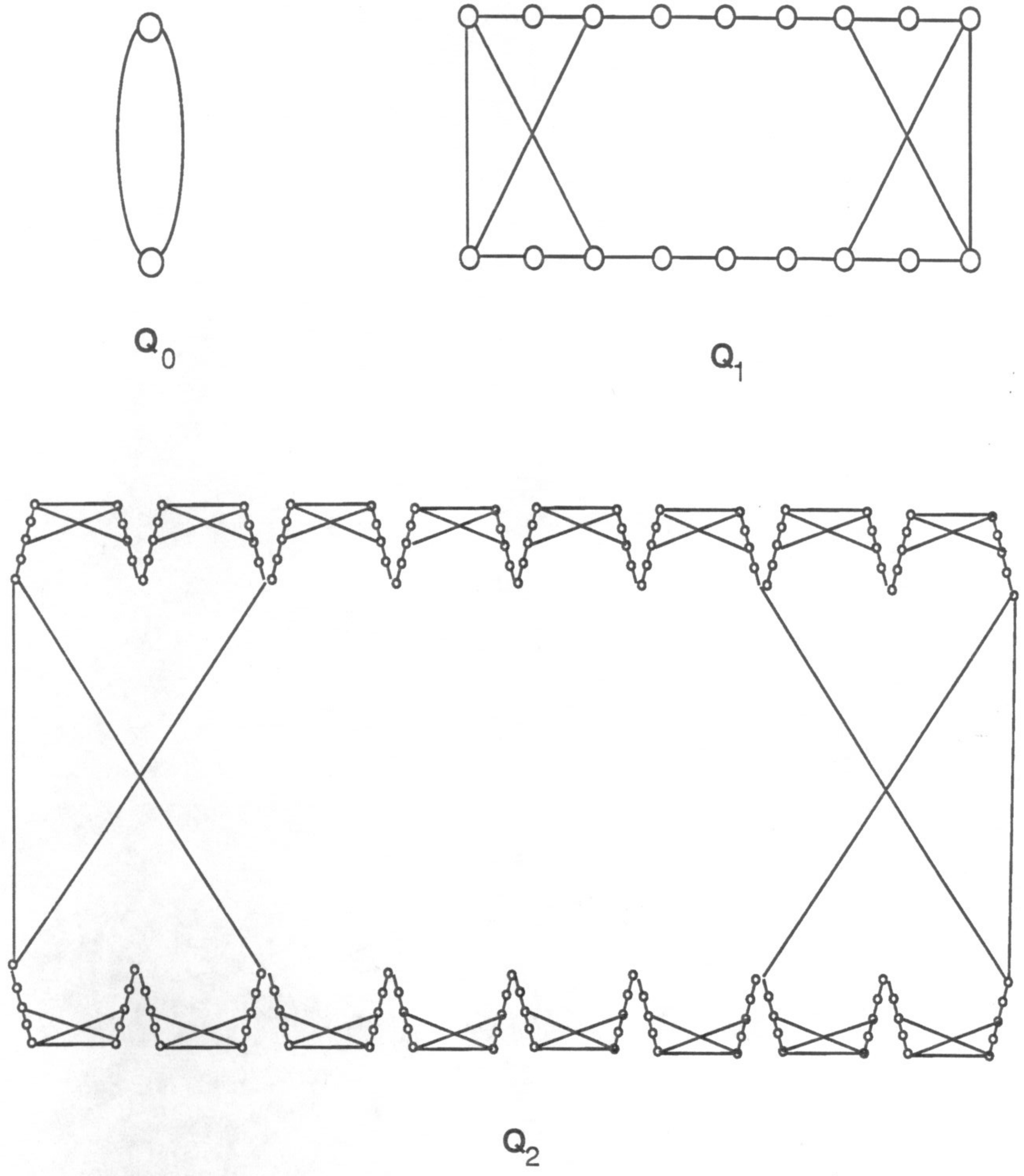

Figure 2: Q_0, Q_1, and Q_2.

4 Linear Time Algorithms for Finding Minimal Subgraphs

4.1 Finding a Minimal 2-Edge-Connected Spanning Subgraph

In the sequel we assume that H is a 2-edge-connected graph and $E(H) = Ess \cup C$ where Ess is a set of essential edges in H. If graph Q is obtained from graph H by shrinking the Ess-components in H and contracting all maximal chains in the resulting graph, we say that Q is a *full contraction of* H. 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 5.

Algorithm 5: *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) $H := G$; $Ess := \emptyset$; $C := E(H)$; $R := \emptyset$;
(2) While $C \neq \emptyset$, do:
    (2.0) Replace $H$ by an ear decomposition of $H$; add the edges in the trivial ears to $R$
          and discard them from $H$.
    (2.1) Compute a spanning tree $T_H$ in $H$ with a maximum number of edges of $Ess$;
    (2.2) Compute a minimal $A \subseteq E(H)$ such that $T_H + A$ is 2-edge-connected.
    (2.3) $R := R \cup (E(H) - E(T_H + A))$;
    (2.4) $H := T_H + A$; $Ess := Ess \cup A \cup \{$critical edges in $H\}$; $C := E(H) - Ess$;
    (2.5) $R := R \cup \{$ edges in $C$ whose endpoints belong to same $Ess$-component $\}$;
    (2.6) Replace $H$ by its full contraction and remove those edges from $Ess$ and $C$ that
          are not in $H$.
(3) Return graph $G - R$.
```

By combining the proof of theorem 5 with corollary 8 one can show that algorithm 5 does indeed produce a minimal 2-edge-connected spanning subgraph of the input graph. The following lemma is needed for the analysis of algorithm 5.

Lemma 10 *Let F be a forest on l leaves in which r nodes are marked. If every chain in F that does not contain a marked node as an internal vertex has length at most k, then $n(F) < k(2l+r)$.*

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. Let l, p, and p' denote the number of nodes of degree 1, degree 2, and degree ≥ 3, respectively, in F'. We know that $\sum_{v \in V(F')} deg(v) \leq 2n(F') - 2$. Since the left-hand side is at least $l + 2p + 3p'$ and $n(F') = l + p + p'$, we find that $p' \leq l - 2$ and hence $p' < l$. Since $p \leq r$, we have $n(F') < 2l + r$. By noting that $n(F) \leq n(F') + (k-1)m(F')$ and hence $n(F) < kn(F')$, the claim of the lemma follows. []

Theorem 7 *Let $E(H) = Ess \cup C$ where Ess is a set of essential edges. Let Q be a full contraction of H. Then $n(Q) < 12 \mid C \mid$.*

Proof. Let Q' denote the graph $Q - C$. The graph Q' is a forest. Each leaf in Q' is covered by at least one edge of C. Thus, Q' has at most $2 \mid C \mid$ leaves. Mark each endpoint of an edge of C in Q. Each unmarked chain in Q' has length at most 2. By applying lemma 10 to Q', we get $n(Q') < 2(4 \mid C \mid + 2 \mid C \mid)$ and hence $n(Q) = n(Q') < 12 \mid C \mid$ as claimed. []

Corollary 10 *Fix a graph H at the end of an iteration of algorithm 5. Let H_{17} denote H after 17 iterations of algorithm 5. Then $n(H_{17}) + m(H_{17}) < .97(n(H) + m(H))$.*

Proof. Let H_i denote the graph H after i iterations of the while-loop. First, we claim that $m(H_{i+1}) < 2n(H_i)$ for any i. 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) and let q denote the number of those ears. Then $m(H') < n(H') + q$ and $q < n(H')$. Since $n(H') = n(H_i)$ and $m(H_{i+1}) \leq m(H')$ the claim follows.

Thus

$$m(H_{i+1}) + n(H_{i+1}) < 3n(H_i). \tag{1}$$

Let q_i denote $\mid C \mid$ after i iterations of algorithm 5. By lemma 1 we have $q_{i+1} < 3q_i/4$. Since $q_1 < 2n(H)$, we obtain $q_i < 2(3/4)^{i-1}n(H)$ and hence, with theorem 7,

$$n(H_i) < 24(3/4)^{i-1}n(H). \tag{2}$$

Combining (1) and (2) yields

$$m(H_{i+1}) + n(H_{i+1}) < 72(3/4)^{i-1}n(H). \tag{3}$$

Substituting $i = 16$ in (3) we find that

$$m(H_{17}) + n(H_{17}) < .97n(H),$$

implying the claim of the corollary. []

Corollary 11 *Algorithm* 5 *finds a minimal 2-edge-connected spanning subgraph of any 2-edge-connected graph on n vertices and m edges in time $O(n + m)$.*

Proof. The time required by one iteration of algorithm 5 is dominated by the time to find a minimal augmentation for a spanning tree. By a result of [9] this can be done in linear time. The claim follows with corollary 10. []

An efficient NC algorithm for this problem is given in [9]. With corollary 11 the work of this algorithm can be reduced by a factor of $\Theta(\log n)$ using standard techniques.

4.2 Finding a Minimal Biconnected Spanning Subgraph

The basic ingredients for the linear-time algorithm to find a minimal 2-edge-connected spanning subgraph are the two operations of shrinking Ess-components and contracting chains. In this section we exhibit a pair of operations on biconnected graphs with similar properties although they are more complicated.

Fix a biconnected graph H with $E(H) = Ess \cup C$ where Ess is a set of essential edges in H. An *Ess-block* of H is a block of $H - C$. The graph $H - C$ need not be connected. Thus, an Ess-block of H is either an isolated vertex in $H - C$, a cutedge in $H - C$, or a maximal biconnected subgraph of $H - C$ (with at least 3 vertices).

Let B be an Ess-block of H with at least 3 vertices. An *internal vertex* of B is a vertex in B that is neither a cutpoint in $H - C$ nor is it incident in H with an edge of C; we write $I(B)$ for the set of internal vertices of B. The operation of *shrinking the Ess-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 simple 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 following results establish that we can compute a minimal biconnected spanning subgraph of the input graph even if we shrink Ess-blocks at each iteration.

Theorem 8 *Let H be a biconnected graph with $E(H) = Ess \cup C$ and let H' be obtained from H by shrinking an Ess-block B in H. Then, for any set S of redundant edges in H, $H - S$ is biconnected iff $H' - S$ is biconnected.*

Proof. Assume that $H - S$ is biconnected. Consider two adjacent edges $e = (u, v)$ and $e' = (v, w)$ in $H - S$ that do not belong to B. There is a simple cycle in $H - S$ containing e and e'. This cycle yields a simple path p in $H - S$ with the following properties: it contains e and e', no internal endpoint of p except possibly v is a vertex of B, and the endpoints of p both belong to B or they are adjacent in $H - S$. The path p is also a simple path in $H' - S$ and both endpoints are distinct in $H' - S$. If both endpoints do not belong to B, then they are adjacent in $H - S$ and e and e' lie on a simple cycle in $H' - S$. If both endpoints of p belong to B, then they represent distinct vertices on C_B. Hence, we can make p into a simple cycle of $H' - S$ by adding a portion of C_B between the endpoints of p that does not contain v. Thus, any two adjacent edges not in B share a block of $H' - S$.

Now consider two adjacent edges $e = (u, v)$ and $e' = (v, w)$ in $H - S$ such that e is an edge of B but e' is not. Again, a simple cycle of $H - S$ includes both edges. From this cycle we get a simple path p in $H - S$ with the following properties: no edge of p is in B, no internal endpoint of p belongs to B and both endpoints of p are nodes of B. The path p is also a simple path in $H' - S$ whose distinct endpoints lie on C_B. Using edges of C_B we complete p into a simple cycle containing both edges of C_B and edges not in B. Since all edges of C_B lie in a single block of $H' - S$, we conclude that $H' - S$ is biconnected.

To prove the "if-part", we proceed in a similar fashion. Assume that $H'-S$ is biconnected. We first consider two adjacent edges $e = (u, v)$ and $e' = (v, w)$ in $H - S$ that are not in B. Some simple cycle in $H' - S$ contains both edges. From this cycle we get a simple path in $H' - S$ containing e and e' and having the following properties: no internal vertex of p except possibly v lies on C_B and the endpoints of B either both belong to C_B or they are adjacent in $H' - S$. The path p is also a simple path in $H - S$ between the same endpoints. If it is not the case that both endpoints of p belong to C_B, then they are adjacent in $H' - S$ and hence, they are adjacent in $H - S$. In this case e and e' lie on a simple cycle in $H - S$. If both endpoints of p belong to C_B, they are nodes of B in $H - S$. We can join them by a path completely contained in B and avoiding v. Again, we find that e and e' lie on a simple cycle in $H - S$.

Now we take two adjacent edges $e = (u, v)$ and $e' = (v, w)$ such that e is an edge of C_B and e' is not. Since $H' - S$ is biconnected, some simple cycle in $H' - S$ contains e and e'. This cycle yields a path p in $H - S$ with the following properties: no edge of p is an edge of B, the endpoints of p are two distinct nodes of B, and no internal vertex of p belongs to B. Thus, by adding a path completely contained in B between the two endpoints of p, we get a simple cycle containing both edges in B and edge not in B. Thus, $H - S$ is biconnected. []

Corollary 12 *Let H' be obtained from a biconnected graph H by shrinking an Ess-block in H. If H'' is a minimal biconnected spanning subgraph of H' and S is the set of redundant edges of H that are not in H'', then $H - S$ is a minimal biconnected spanning subgraph of H.*

Proof. By construction all edges of H' that are not in H are essential in H'. Thus, $H'' = H' - S$ and theorem 8 implies the claim. []

Corollary 13 *Let H' be obtained from H by shrinking any number of essential blocks in H. If H'' is a minimal biconnected spanning subgraph of H' and S is the set of redundant edges of H that are not in H'', then $H - S$ is a minimal biconnected spanning subgraph of H.*

Proof. By induction on the number k of essential blocks shrunk in H. The induction base $k = 0$ is trivial. Assume that the claim holds if H' is obtained by shrinking at most k blocks in H. Now consider the case where H' is obtained from H by shrinking $k+1$ essential blocks of H. The key observation is that H' can be obtained by shrinking a single essential block in a graph H_k that is obtained from H by shrinking k essential blocks in H. Let S' denote the set of edges of H_k that are not in H''. By corollary 12 $H_k - S'$ is a minimal biconnected spanning subgraph of H_k. Since H_k contains all redundant edges of H and any edge of H_k not in H'' is redundant in H, we have $S' = S$. Thus, the induction assumption applied to H and H_k shows that $H - S$ is a minimal biconnected spanning subgraph of H, as required. []

The results in section 3.3 together with the observation that the graphs Q_i are 3-graphs imply that shrinking Ess-blocks is not sufficient for improving the $\Theta(m+n \log n)$ time bound for algorithm 3.

The second operation on biconnected graphs is defined on the block structure of $H - C$. A *block chain* in biconnected graph H is an alternating sequence $c_1 B_1 \ldots c_k B_k c_{k+1}$ of vertices and Ess-blocks in H with the following properties: (i) each B_i ($1 \leq i \leq k$) has exactly two cutpoints in $H - C$, 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 of C. A *maximal block chain* in H is a block chain in H not properly contained in any other block chain of H.

It is helpful to interpret block chains in an auxiliary graph which we shall now define. Let H' be an arbitrary graph. The *block graph* of H' ([17]), 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 that the block graph of H' is a tree for any connected graph H'.

Now consider biconnected graph H. Define a mapping h from the vertices of H to the vertices of $blk(H - C)$ as follows: for any vertex v that is a cutpoint in $H - C$, $h(v) = v$; if v is not a cutpoint then $h(v)$ is the block of $H - C$ containing v. The *condensation* of H, denoted by $\hat{H}$, is the graph $blk(H - C) + \{(h(u), h(v)) : (u, v) \in C\}$.

A chain in $\hat{H}$ is a path of vertices in $\hat{H}$ all of whose internal vertices have degree 2. A chain in $\hat{H}$ is *proper* if its terminal vertices are cutpoints in $H - C$. A *maximal proper chain* is a proper chain of $\hat{H}$ not properly contained in another proper chain of $\hat{H}$.

Observation 2 *The sequence $c_1 B_1 \ldots c_k B_k c_{k+1}$ is a maximal block chain in H iff it is a maximal proper chain in $\hat{H}$.*

The operation of *contracting the block chain* $c_1 B_1 \ldots c_k B_k c_{k+1}$ in H consists of deleting in H all vertices in the blocks of this sequence except c_1 and c_{k+1}, adding a new vertex u and two new edges (u, c_1) and (u, c_{k+1}).

Theorem 9 *Let H' be obtained by contracting the block chain $c_1B_1 \dots c_kB_kc_{k+1}$ in the biconnected graph H. For any subset S of redundant edges of H, $H - S$ is biconnected iff $H' - S$ is biconnected.*

Proof. We refer to the edges in the B_i's as the *internal edges of* H. Assume that $H - S$ is biconnected. Fix two adjacent edges $e = (u, v)$ and $e' = (u', v')$ that are not internal in H. Some simple cycle D in $H - S$ contains e and e'. If D does not contain an internal edge, it is also a simple cycle in $H' - S$ containing e and e'. Otherwise, it contains both c_1 and c_2 and we get a simple cycle in $H' - S$ containing e and e' by replacing the portion of D consisting of internal edges by the path c_1, u, c_2 in $H' - S$.

Now consider two adjacent edges e and e' in $H - S$ such that only e is internal. A simple cycle containing e and e' in $H - S$ gives a simple path in $H' - S$ between c_1 and c_2 and not including u. Hence, there is a simple cycle in $H' - S$ containing both non-internal edges and the edges (c_1, u) and (c_2, u). We conclude that $H' - S$ is biconnected.

The if-part of the theorem is proved similarly. []

Corollary 14 *Let H' be obtained from H by contracting a block chain in H. Let H'' be a minimal biconnected spanning subgraph of H' and let S denote the set of redundant edges of H that are not in H''. Then, $H - S$ is a minimal biconnected spanning subgraph of H.*

Proof. Immediate with theorem 9. []

Corollary 15 *Let H' be obtained from H by contracting any number of block chains in H. Let H'' be a minimal biconnected spanning subgraph of H' and let S denote the set of redundant edges of H not in H''. Then, $H - S$ is a minimal biconnected spanning subgraph of H.[]*

Proof. By induction on the number of block chains contracted in H. []

If the graph Q is obtained from H by first shrinking all Ess-blocks of H and then contracting all maximal block chains in the resulting graph, we say that Q is a *full contraction* of H. Consider a variation of algorithm 3 in which we replace H by its full contraction at the end of each iteration of the while-loop - denote the modified algorithm by algorithm 6.

Algorithm 6: *Computing a minimal biconnected spanning subgraph of G.*

Input Biconnected graph G.

Output Minimal biconnected spanning subgraph of G.

```
(1) $H := G$; $Ess := \emptyset$; $C := E(H)$; $R := \emptyset$;
(2) While $C \neq \emptyset$, do:
    (2.0) Replace $H$ by an open ear decomposition of $H$; add the edges in the trivial ears
          to $R$ and remove them from $H$.
    (2.1) Compute a spanning tree $T_H$ in $H$ with a maximum number of edges of $Ess$;
    (2.2) Compute a minimal $A \subseteq E(H)$ such that $T_H + A$ is biconnected.
    (2.3) $R := R \cup (E(H) - E(T_H + A))$;
    (2.4) $H := T_H + A$; $Ess := Ess \cup A \cup \{\text{critical edges in } H\}$; $C := E(H) - Ess$;
    (2.5) Shrink all $Ess$-blocks in $H$ and contract all maximal block chains in the resulting
          graph. Remove those edges from $Ess$ that are not in $H$ and add to $Ess$ the newly
          created edges.
(3) Return graph $G - R$.
```

The correctness of algorithm 6 can be shown by combining the proof of theorem 5 with corollaries 13 and 15.

We now state the main result of this section.

Theorem 10 *Let H be biconnected and $E(H) = Ess \cup C$ where Ess is a set of essential edges in H. If Q is a full contraction of H, then $n(Q) < 58 \mid C \mid$.*

Proof. Let Q' denote the graph $Q - C$. To bound $n(Q)$, we consider $blk(Q')$, the block graph of Q', and $\hat{Q}'$, the condensation of Q'. From the way Q is constructed, it follows that any proper chain in $\hat{Q}'$ has length at most 4; thus, an arbitrary chain in $\hat{Q}'$ has length at most 6. Mark each node of $blk(Q')$ that is incident in $\hat{Q}'$ with an edge of C. Thus, any unmarked chain in $blk(Q')$ has length at most 6. The graph $blk(Q')$ is a forest. Let l denote the number of leaves in $blk(Q')$. Since each leaf of $blk(Q')$ is incident in $\hat{Q}'$ with at least one edge of C, we have $l \leq 2 \mid C \mid$. By applying lemma 10 we see that $blk(Q')$ has less than $6(4 \mid C \mid +2 \mid C \mid) = 36 \mid C \mid$ vertices. Since there are more blocks in Q' than there are cutpoints, Q' has less than $18 \cdot \mid C \mid$ cutpoints.

We partition the vertices of Q' into 3 classes: class 1 contains the cutpoints, class 2 includes the endpoints of edges of C (that are not cutpoints), and class 3 comprises those vertices used to subdivide cycles when shrinking Ess-blocks in H. Let n_1, n_2, and n_3 denote the number of vertices in class 1, class 2, and class 3, respectively. Clearly $n_2 \leq 2 \mid C \mid$. Above we have shown that $n_1 < 18 \mid C \mid$. Note that the number of class 3 vertices in any block 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, is an upper bound on n_3. This sum is at most $2 \mid C \mid + m(blk(Q'))$ and hence $n_3 < 2 \mid C \mid + 36 \mid C \mid = 38 \mid C \mid$. Altogether we find that $n_1 + n_2 + n_3 < 58 \mid C \mid$ and hence $n(Q') = n(Q) < 58 \mid C \mid$. []

Corollary 16 *Fix a graph H at the end of an iteration of algorithm 6. Let H_{23} denote H after 23 iterations of algorithm 6. Then $n(H_{23}) + m(H_{23}) < .83(n(H) + m(H))$.*

Proof. Let H_i denote the graph H after i iterations of the while-loop. As in the proof of corollary 10, one argues that $m(H_{i+1}) < 2n(H_i)$ for any i.

Thus

$$m(H_{i+1}) + n(H_{i+1}) < 3n(H_i). \tag{4}$$

Let q_i denote the $\mid C \mid$ after i iterations of algorithm 5. By lemma 1 we have $q_{i+1} < 3q_i/4$. Since $q_1 < 2n(H)$, we obtain $q_i < 2(3/4)^{i-1}n(H)$ and hence, with theorem 7,

$$n(H_i) < 116(3/4)^{i-1}n(H). \tag{5}$$

Combining (4) and (5) yields

$$m(H_{i+1}) + n(H_{i+1}) < 348(3/4)^{i-1}n(H). \tag{6}$$

Substituting $i = 22$ in (6) we find that

$$m(H_{23}) + n(H_{23}) < .83n(H),$$

implying the claim of the corollary. []

Corollary 17 *Algorithm 6 finds a minimal biconnected spanning subgraph of any biconnected graph on n vertices and m edges in time $O(n + m)$.*

Proof. The time required by one iteration of algorithm 6 is dominated by the time to find a minimal biconnectivity augmentation for a spanning tree (step (2.2) of algorithm 6). By a result of [9] this can be done in linear time. The claim follows with corollary 16. []

References

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

[2] E. Dahlhaus, M. Karpinski, *An efficient algorithm for the 3MIS problem,* Technical Report TR-89-052, September 1989, International Computer Science Institute, Berkeley, CA.

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

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

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

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

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

[8] P. Kelsen, *An efficient parallel algorithm for finding a maximal independent set in hypergraphs of dimension 3,* manuscript, Department of Computer Sciences, University of Texas, Austin, TX, January 1990.

[9] P. Kelsen, V. Ramachandran, *On finding minimal 2-connected subgraphs*, Tech. Report TR-90-16. June 1990, Department of Computer Sciences, University of Texas, Austin, TX 78712; extended abstract in *Proceedings of the Second ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1991, pp. 178-187.

[10] P. Kelsen, 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.

[11] J. Lewis, M. Yannakakis, *The node-deletion problem for hereditary properties is NP-complete*, J. C. S. S., vol. 20, v1980, pp.219-230.

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

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

[14] 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.

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

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

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

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