[12] K. Simon, *Finding a minimal transitive reduction in a strongly connected digraph with linear time*, manuscript, 1989.

[13] K. Simon, *Personal communication*, March 1990.

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

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

in a dag with $\Theta(n^{1-\epsilon})$ sources and sinks. Computing reachability for these pairs, however, requires $\Omega(n^{2-2\epsilon})$ time.

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, ICSI, 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*, draft, Department of Computer Sciences, Princeton University, NJ, 1991.

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

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

Proof. We prove the lemma by induction on i. It holds for $i = 0$ since the set containing an incoming (essential) edge for each vertex other than x constitutes a forward branching in D_0. Assume inductively that D_i contains a forward branching F rooted at some vertex x and having the properties stated above. Let D_{i+1} be constructed as decribed above. Use y to denote the root of the branching $T(x)$ spanning the gadget for x (see step (4) of algorithm 2'). Let B denote the set containing all redundant edges of D_{i+1} and containing, for each vertex of D that is not a source, an arbitrary incoming edge of D. Exactly as in the proof of theorem 4 one shows that there exists a forward branching rooted at y and containing all edges in B.[]

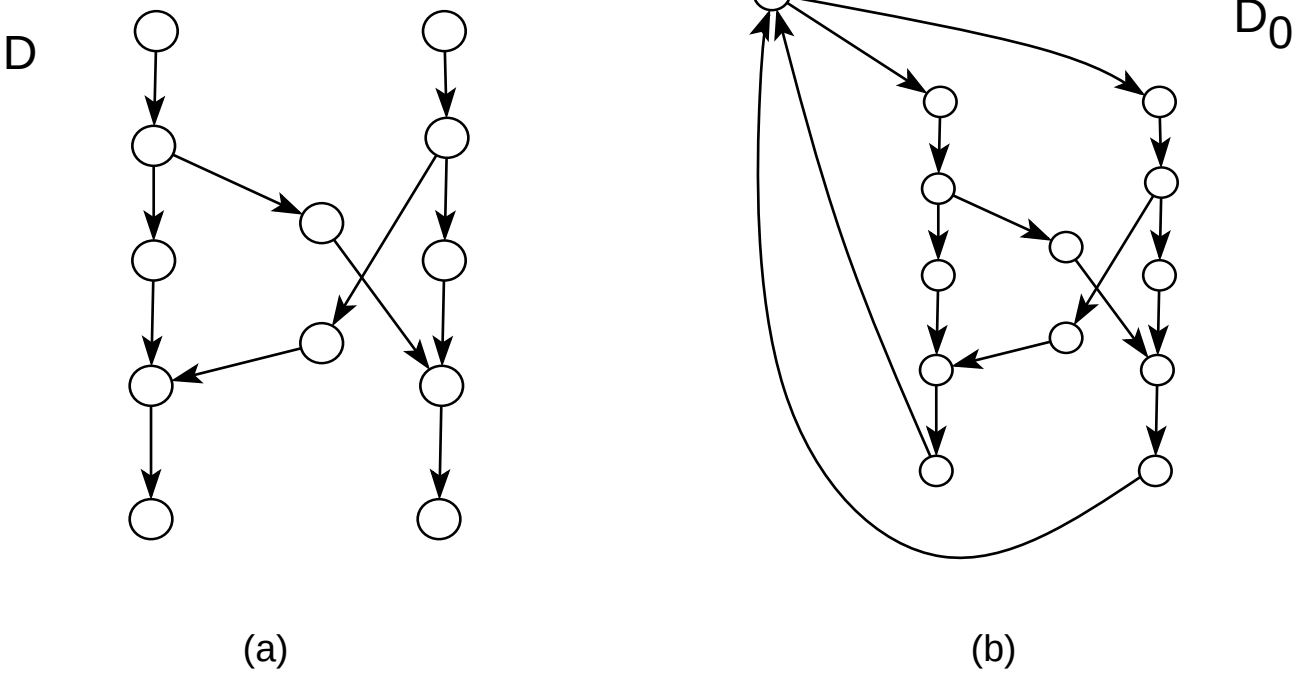
Theorem 11 *Let G_1 be the graph obtained from D_i by doubling external essential edges (step (1) of algorithm 2'). The graph G_1 is obtained by shrinking all essential components in D_{i+1} and contracting all maximal chains in the resulting graph. Furthermore, D_i is of the form $T + A$ where T is an optimal branching in G_1 and A is a minimal augmentation for T in G_1. Hence, D_i may be obtained from D_{i+1} by running one iteration of algorithm 1'c.*

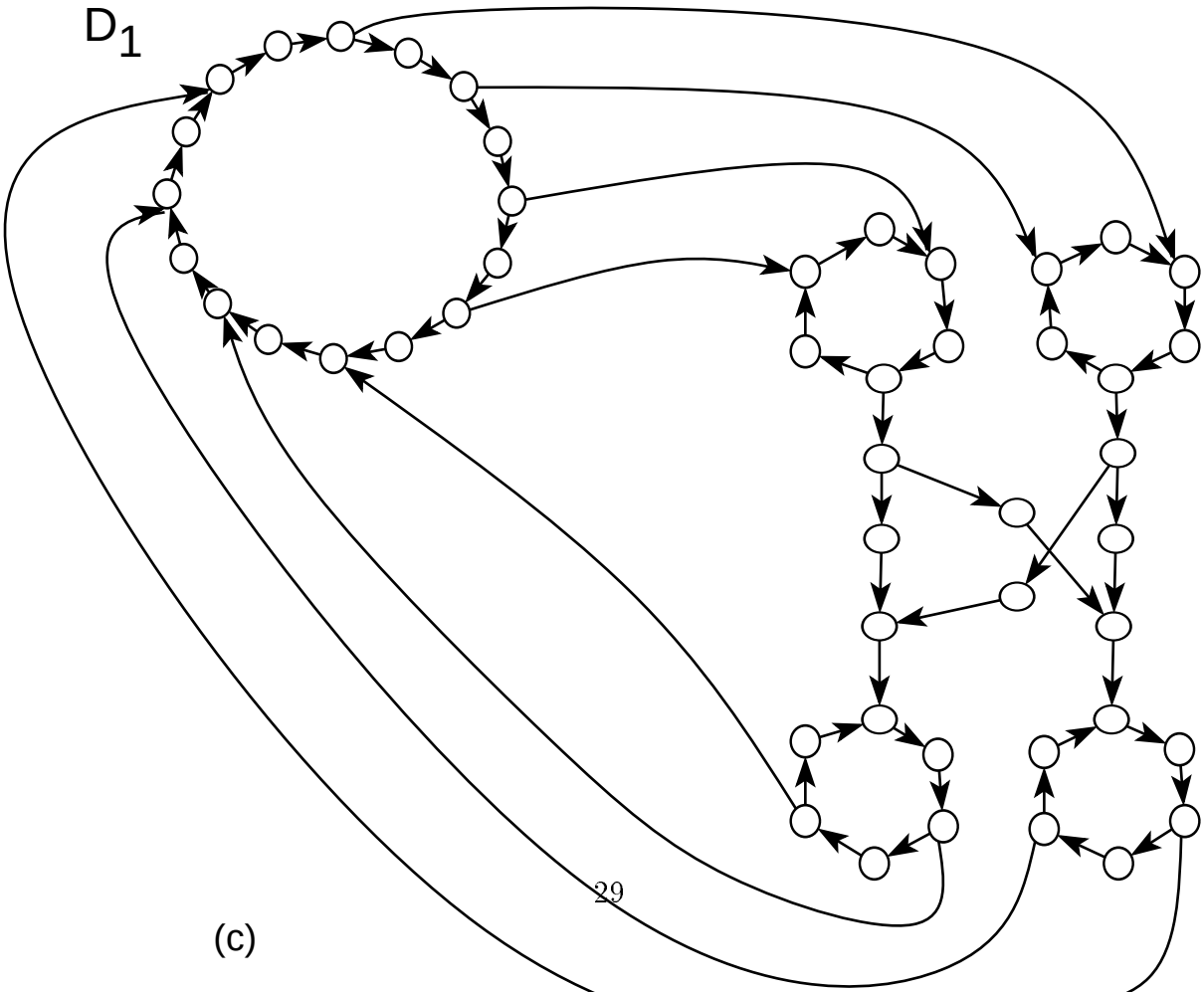
Proof. The essential components in D_{i+1} are exactly the gadgets for the representatives of the external vertices in D_i. Let G be the graph obtained by shrinking these gadgets in D_{i+1}. The only essential edges in G are those in D. By the second property of D, the graph G has no chain of length greater than 2. Hence, contracting chains in G does not change G and $G = G_1$. This proves the first part of the theorem.

By lemma 15 the graph D_i has a forward branching F rooted at some vertex x with the property that it contains all the redundant edges in D_i and also contains, for each vertex of D that is not a source, an incoming edge in D. Since the vertices of D other than the sources of D are the only nodes in G_1 that have an incoming essential edge in G_1, the branching F is an optimal branching in G_1. The second statement of the theorem follows.[]

We call the sequence of graphs obtained at the end of the while-loop in algorithm $1'c$ a *trace* for algorithm 1'c. By theorem 11 the graphs $D_i, D_{i-1}, \ldots, D_0$ form a possible trace for algorithm 1'c. Let p, s, and t denote the number of vertices, sources and sinks, respectively, in D. It can be shown that D_i has size $O(c^i(s+t)+p)$ for a constant c that does not depend on p. By choosing $p = n$, $s + t = n^{1-\epsilon}$, $0 < \epsilon < 1$, and $i = \Theta(\log n)$, we see that the worst case running time of algorithm $1'c$ on the graph D_i having $\Theta(n)$ vertices and $\Theta(n)$ edges is $\Omega(n \log n)$.

By examining the family of graphs D_i constructed above, it appears that what is needed for a linear-time algorithm is an operation that reduces dags of essential edges at each iteration. This seems to require determining the reachability for every (source,sink)-pair


D
z
D_0
(a)
(b)


D_1
(c)

that is contained in H (the intermediate subgraph of the input graph) at each iteration of algorithm 1'c.

Fix a dag (directed acyclic graph) D . We assume that D has the the following three properties. First, the edges of D are essential in any strongly connected graph G that satisfies the following two conditions: G contains D as a subgraph, and no edge of G that is not an edge of D is incident with a vertex of the dag other than a source or a sink. Second, we require that D has no chain of length greater than 2. Third, each source has exactly one outgoing edge and each sink has exactly one incoming edge. One can construct a dag on s sources and t sinks having these 3 properties by taking any dag D' with s sources and t sinks in which each vertex has indegree or outdegree at least 2, subdividing each edge of D' by a new vertex, and adding edges into the sources form s new vertices and adding edges from the sinks to t new vertices.

The graph, D_0, the first graph in the sequence, is defined as follows: the vertices of D_0 are the vertices of the dag D plus one new vertex x. The edges of D_0 are the edges of D (i.e., D is a subgraph of D_0) plus an edge from x to every source of D and an edge from every sink of D to x. See figure 5(a) and 5(b) for an example of D and D_0.

Call an edge of D_i *external* if it is not an edge of D. A vertex of D_i is *external* if it is not a vertex of D or it is a sink or a source in D. We construct D_i from D_{i-1} by running algorithm 2' (see section 2.2) on input D_{i-1} with several modifications. First, we assume that there is a forward branching F in D_{i-1} with the property that it contains all redundant edges of D_{i-1} and contains, for each vertex of D that is not a source, an incoming edge in D. The importance of this assumption will become clear below. In step (1) we only double the external essential edges. In step (2), instead of choosing an arbitrary forward branching T, we choose the branching F described above. Steps (3), (4), and (5) are only performed for external vertices. For each vertex that is not external, we add a unique copy to G'. Step (6) is only done for external edges. For each edge of D in G_1 we proceed as follows: if an endpoint of the edge is neither a source nor a sink in D, then the corresponding endpoint in G' is the copy of that endpoint in G'. If the endpoint is a source or a sink of D (i.e., an external vertex) the corresponding endpoint in G' is defined as for endpoints of external edges (based on edge number at that endpoint). These modifications ensure that the dag D is a subgraph of essential edges in each D_i, a fact that is crucial for the derivation of the lower bound. The graph D_1 constructed from graph D_0 in figure 5(b) is depicted in figure 5(c).

Lemma 15 *In each graph D_i there exists a forward branching containing all redundant edges in D_i and containing, for each vertex of D that is not a source, an incoming edge in D.*

Corollary 14 *Assume the input graph has n vertices and less than n redundant edges. After 7 iterations of the modified algorithm, the number of vertices in H is less than $.82n$.*

Proof. Let H_i be the graph H at the end of the ith iteration of the modified algorithm. It follows from an earlier result that $r(H_{i+1}) < 2/3 \cdot r(H_i)$ for any i. We have $r(H_0) < n$ and hence, $r(H_7) < (2/3)^7 n$. By theorem 10, $n(H_7) < 14r(H_7)$. The claim then follows by checking that $(2/3)^7 \cdot 14 < .82$.[]

By corollary 14 and the fact that there is a linear-time procedure for finding a minimal augmentation ([9]), we conclude that the modified algorithm for biconnectivity runs in linear time.

3.3 Finding a Minimal Strongly Connected Spanning Subgraph

In this section we consider the problem of finding a minimal strongly connected spanning subgraph in a strongly connected digraph. This problem has withstood all of our attempts so far to obtain a linear-time solution. We present evidence below that this problem is indeed significantly harder than the problems for undirected graphs that we have considered earlier.

Fix a strongly connected digraph H. Let $R(H)$ denote the set of redundant edges of H. An *essential component* of H is a strongly connected component of $H - R(H)$. A *chain* of H is a path in H all of whose internal vertices have in-degree and out-degree 1 in H. Thus, all edges of a chain are essential in H.

The operation of *shrinking an essential component* in H consists of collapsing the vertex set of an essential component in H. The operation of *contracting a chain* consists of collapsing the internal vertices of the chain in H.

Consider the following algorithm, referred to as algorithm 1'c: modify algorithm $1'$ (c.f. section 2.2) by shrinking all essential components in H and contracting all maximal chains in the resulting graph at the beginning of each iteration of the while-loop. An edge will be in the output graph if it occurs in the input graph and was essential at some iteration of algorithm 1'c. It can be shown that the output graph of algorithm $1'c$ is indeed a minimal strongly connected spanning subgraph of the input graph.

Algorithm $1'c$ is the natural analog of algorithm $1c$ for directed graphs. Unlike that algorithm, however, it has a nonlinear lower bound on its worst-case sequential running time. We shall construct a sequence of graphs D_i, $i \geq 0$, such that algorithm $1'c$ requires $\Omega(m + n \log n)$ operations in the worst case on a graph in this sequence (as usual, n and m denote the number of vertices and edges, respectively, in the input graph). The key to proving the lower bound is to exhibit an acyclic subgraph of essential edges of size $\Omega(n)$

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 C in $H - S$ contains e and e'. If C 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 C 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 12 *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 A denote the set of edges of H not in H''. Then, $H - A$ is a minimal biconnected spanning subgraph of H.[]*

Corollary 13 *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 A denote the set of redundant edges of H not in H''. Then, $H - A$ is a minimal biconnected spanning subgraph of H.[]*

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

We are now ready to state the main result of this section.

Theorem 10 *If Q is obtained from H by first shrinking all essential blocks of H and then contracting all maximal block chains in the resulting graph, then $n(Q) < 14r(H)$.*

Proof. To bound $n(Q)$, we consider $blk(Q)$, the block graph of Q. Let l denote the number of leaves in $blk(Q)$. From the way Q is constructed, it follows that any proper chain in $blk(Q)$ has length at most 4; thus, an arbitrary chain in $blk(Q)$ has length at most 6. Each leaf of $blk(Q)$ is incident with at least 2 redundant edges in $\hat{Q}$; hence, $l \leq r(H)$. Mark each node of $blk(Q)$ that is incident with a redundant edge of H. By applying lemma 14 we see that $blk(Q)$ has less than $6(2r(H) + 2r(H)) = 24r(H)$ vertices. Hence, Q^e has less than $12 \cdot r(H)$ cutpoints. Each vertex of Q^e is either a cutpoint in Q^e or it is incident with a redundant edge of H. Thus, $n(Q) < 12r(H) + 2r(H) = 14r(H)$ as claimed.[]

If the graph Q is obtained from H as described in theorem 10, we say that Q is a *full contraction* of H. Consider a variation of algorithm 1 in which we replace H by its full contraction at the end of each iteration of the while-loop.

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 A' denote the set of edges of H_k that are not in H''. By corollary 10 $H_k - A'$ 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 $A' = A$. Thus, the induction assumption applied to H and H_k shows that $H - A$ is a minimal biconnected spanning subgraph of H, as required.[]

The second operation on biconnected graphs is defined on the block structure of H^e. A *block chain* in a biconnected graph H is an alternating sequence $c_1 B_1 \ldots c_k B_k c_{k+1}$ of vertices and essential blocks in H with the following properties: (i) each B_i $(1 \leq i \leq k)$ has exactly two cutpoints in H^e, 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 on any B_i except possibly c_1 and c_{k+1} are incident with a redundant edge of H. 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. The *block graph* of H ([14]), 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 a biconnected graph H. Define a mapping h from the vertices of H to the vertices of $blk(H^e)$ as follows: for any vertex v that is a cutpoint in H^e, $h(v) = v$; if v is not a cutpoint then $h(v)$ is the block of H^e containing v. The *condensation* of H, denoted by $\hat{H}$, is the graph $blk(H^e) + \{(h(u), h(v)) : (u, v)$ is a redundant edge in $H\}$.

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^e. 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 $c_1 B_1 \ldots c_k B_k c_{k+1}$ in the biconnected graph H. For any subset S of edges of H, $H - S$ is biconnected iff $H' - S$ is biconnected.*

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 10 *Let H' be obtained from a biconnected graph H by shrinking an essential block in H. If H'' is a minimal biconnected spanning subgraph of H' and A is the subset of redundant edges of H that are not in H'', then $H - A$ 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, $H'' = H' - A$ and theorem 8 implies the lemma.[]

Corollary 11 *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 A is the set of edges of H that are not in H'', then $H - A$ 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

pointed out earlier condition $(C3)$ does not hold for biconnectivity, i.e., by shrinking an essential component in a biconnected graph we may loose some information that is necessary to compute a minimal biconnected spanning subgraph. The new operation we define for biconnected graphs is similar, though, to the shrinking of essential components: instead of collapsing an essential component into a single vertex, it replaces it by a simple cycle. We now define these two operations formally.

Fix a biconnected graph H for which the subgraph of essential edges, denoted by H^e, is not biconnected. An *essential block* of H is a block of H^e. The graph H^e need not be connected. Thus, an essential block of H is either a single vertex, a single edge, or a maximal biconnected subgraph of H^e (with at least 3 vertices).

Let B be an essential 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^e nor is it incident with a redundant edge; we write $I(B)$ for the set of internal vertices of B. The operation of *shrinking the essential 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 $C_B = \{(u_1, v_1), (v_1, u_2), \ldots, (u_{k-1}, v_{k-1}), (v_{k-1}, u_k), (u_k, v_k), (v_k, u_1)\}$.

Theorem 8 *Let H be a biconnected graph and let H' be obtained from H by shrinking an essential block B in H. Then, for any $S \subseteq R(H)$, $H - S$ 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,

at least $l+2p+3q$ and $n(F') = l+p+q$, we find that $q \leq l-2$ and hence $q < l$. Since $p \leq r$, we have $n(F') < 2l + r$. By noting that $n(F) < kn(F')$, the claim of the lemma follows.[]

Theorem 7 *If Q is obtained from H by shrinking the essential components in H and contracting all maximal chains in the resulting graph, then $n(Q) < 8r(H)$.*

Proof. Let Q^e denote the graph $Q - R(Q)$. The graph Q^e is a forest. Each leaf in Q^e is covered by at least two redundant edges of H. Thus, Q^e has at most $r(H)$ leaves. Mark each endpoint of a redundant edge of H in Q. Each unmarked chain in Q^e has length at most 2. By applying lemma 14 to Q^e, we get $n(Q^e) < 2(2r(H) + 2r(H))$ and hence $n(Q) < 8r(H)$ as claimed.[]

Corollary 8 *Assume the input graph has n vertices and less than n redundant edges. After 6 iterations of algorithm 1c, the number of vertices in H is less than $.71n$.*

Proof. Let H_i be the graph H at the end of the *ith* iteration of algorithm 1c. It follows from an result in [9] that $r(H_{i+1}) < 2/3 \cdot r(H_i)$. We have $r(H_0) < n$ and hence $r(H_6) < (2/3)^6 n$. By theorem 7, $n(H_6) < 8r(H_6)$. The claim then follows by verifying that $(2/3)^6 \cdot 8 < .71$.[]

Corollary 9 *Algorithm 1c 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 1c 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.[]

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

3.2 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 essential components and contracting chains. We have shown that an algorithm that performs only one of these operations has a worst case time complexity of $\Omega(m + n \log n)$ while the incorporation of both operations leads to an algorithm that runs in time $O(n + m)$. Here, n and m denote the number of vertices and edges of the input graph, respectively.

In this section we exhibit a pair of operations on biconnected graphs with similar properties. One of the operations, the contraction of block chains, is similar to the corresponding operation on 2-edge-connected graphs. The other operation is more complicated. As we

Lemma 13 *No 2-edge-connected spanning subgraph of F_i has a chain of length greater than 6.*

Proof. A proof by induction on i shows that the maximum degree in any F_i is 3 (as observed earlier). Thus, each gadget in F_i has at most 6 edges. Let F be an arbitrary 2-edge-connected spanning subgraph of F_i. The graph F must completely contain every gadget of F_i. Moreover, each gadget is incident with at least two edges connecting it to vertices outside the gadget. The claim follows.[]

By lemma 13 there is an execution for algorithm $1b$ in which there are $\Omega(\log n)$ intermediate graphs (at the end of the while-loop) each of which has size $\Omega(n)$, thus implying the $\Omega(m + n \log n)$ lower bound for the worst-case sequential time.

The logical next step is to consider a variation of algorithm 1 that shrinks essential components and contracts essential chains at the beginning of the while-loop. We refer to the resulting algorithm as algorithm 1c. In the next section we show that algorithm $1c$ runs in linear time on any input graph.

3 Linear Time Algorithms for Finding Minimal Subgraphs

3.1 Finding a Minimal 2-Edge-Connected Spanning Subgraph

In this section we show that algorithm 1c, defined at the end of the last section, computes a minimal 2-edge-connected spanning subgraph of an arbitrary graph in linear time.

For any graph H, we denote by $n(H)$ and $m(H)$ the number of vertices and edges in H, respectively. If H is 2-edge-connected (biconnected, strongly connected), $R(H)$ denotes the set of redundant edges in H and $r(H)$ the number of these edges. A *chain* in a graph is a path whose internal vertices have degree 2. The following lemma is needed for the analysis of algorithm 1c.

Lemma 14 *Let F be a forest on n 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 q 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 lefthand side is

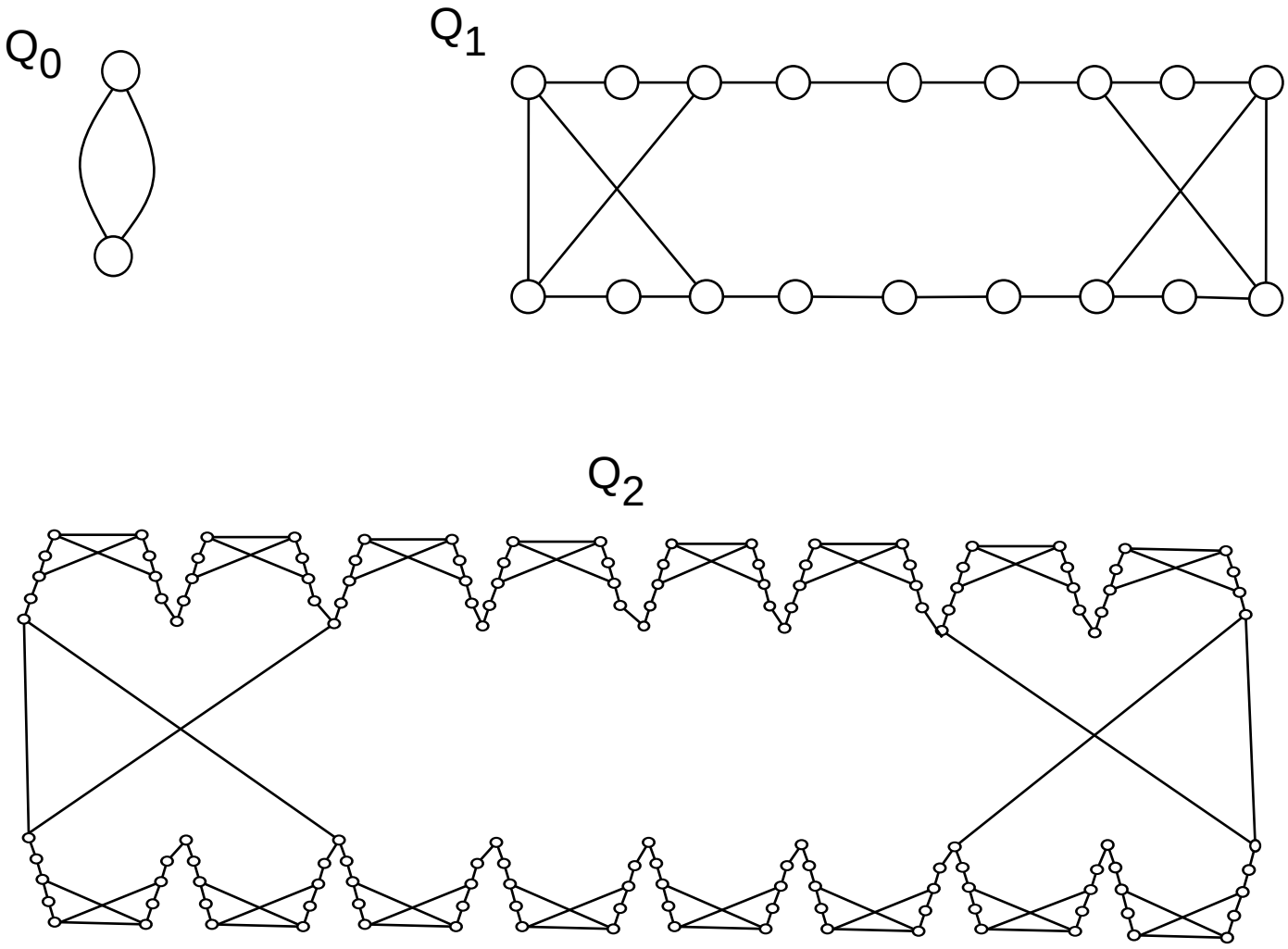


Figure 4: Q_0, Q_1, and Q_2.

tracting a collection of edge-disjoint chains. Then, H' is 2-edge-connected iff H is 2-edge-connected.[]

Corollary 6 *Assume that H' is constructed by contracting a collection of 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.[]*

For a 2-edge-connected graph H let $c^*(H)$ be the maximum integer k for which there are graphs $H_0 \ldots H_k$ with the following properties: $H_0 = H$, H_k is a minimal 2-edge-connected spanning subgraph of H, 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 $i > 0$. The following result is another corollary of lemma 11

Lemma 12 *If H' is obtained from H by contracting (edge-disjoint) chains, then $c^*(H) = c^*(H')$.[]*

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 $v_1^e, v_2^e, v_3^e, v_4^e$; 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 4 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 12 and the induction assumption we find that $c^*(Q) \geq i$ and hence $c^*(Q_{i+1}) \geq i + 1$.[]

Corollary 7 *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 $n \geq 1$.[]*

Let algorithm 1*b* be a variant of algorithm 1 in which all maximal chains are contracted at the beginning of each while loop. The following lemma implies that algorithm 1*b* may perform poorly on the sample for 2-edge-connectivity, i.e., it may require $\Omega(m + n \log n)$ time.

connectivity. We describe a refinement of algorithm 1 that takes into account property $(C3)$ satisfied by 2-edge-connectivity.

Let H be a 2-edge-connected graph and let $R(H)$ be the set of redundant edges in H. An *essential component of* H is a 2-edge-connected component of $H - R(H)$. The operation of *shrinking an essential component* in H consists of collapsing in H the vertex set of this component.

Theorem 5 *Assume that H' is constructed by shrinking some essential 2-edge-connected 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.[]*

Modify algorithm 1 (for finding a minimal spanning 2-edge-connected subgraph) as follows: at the beginning of the while-loop find the essential components of H and shrink them into single vertices. The ouput of this algorithm is a spanning subgraph of the input graph consisting of exactly those edges in the input graph that were essential at some iteration. We refer to the modified algorithm as algorithm 1a.

With theorem 5 one can show that algorithm $1a$ does indeed produce a minimal 2-edge-connected spanning subgraph of the input graph. Let us examine the running time of algorithm $1a$ on graphs F_i of the sample for 2-edge-connectivity. The worst-case analysis of algorithm 1 on input graphs from the sample hinged on the fact that, by running one iteration of algorithm 1 on graph F_i, we may obtain a graph F such that F_{i-1} is an essential contraction of F. Now consider what happens if algorithm 1a chooses the optimal trees and the minimal augmentations described in the proof of theorem 2 on input F_i. After shrinking essential components in the resulting graph, algorithm $1a$ is left with graph F_{i-1}. It then follows from lemma 3 that for this particular choice of optimal trees and minimal augmentations algorithm 1a, although still requiring $\Omega(\log n)$ iterations, computes a minimal 2-edge-connected spanning subgraph of F_i in time linear in the size of F_i.

The obvious question is: does algorithm $1a$ run in linear time on any input graph. We answer this question negatively by constructing graphs with the property that if we run algorithm $1a$ on them, no cycle of essential edges will be created at any intermediate stage.

A *chain in* H is a path of edges 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 11 *Let H' be obtained from a graph H that need not be 2-edge-connected by con-*

observation is the following: let $H' = T' + A'$ where T' is an optimal branching in H' rooted at x and A' is a minimum augmentation for T' in H'. Let T be an optimal branching in H obtained by augmenting T' with forward branchings for the collapsed subgraphs in H. Then the set A containing the edges in the collapsed subgraphs in H that are not in T together with the edges of H that correspond to edges of A' forms a minimum augmentation for T in H.[]

Recall that F_i' denotes a graph in a fixed sample for strong connectivity constructed by algorithm 2'. Let P denote strong connectivity.

Lemma 10 *$\hat{c}_P(F_i') \geq i$ for any $i \geq 0$.*

Proof. We prove the following stronger statement: for each $i \geq 0$ there is a vertex x in F_i' such that $\hat{c}_P(H, x) \geq i$ and F_i' is linear with respect to x. We show this by induction on i.

The base case clearly holds. Assume the statement holds for F_i'. Let G_1 be the graph constructed in step (1) of algorithm 2′ by doubling the essential edges in F_i'. 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 x and A is a set of essential edges in F_i'. Since every edge of G_1 is redundant, T is an optimal branching in G_1. Furthermore, A is a minimum augmentation for T in G_1. To see this, fix an essential edge e in F_i' that is not in T. Let e' be the edge parallel to e in G_1. By the definition of G_1 and the fact that e is essential in F_i', any minimal augmentation for T in G_1 has a nonempty intersection with $\{e, e'\}$. Thus, the set of essential edges in F_i' that are not in T constitutes a minimum augmentation for T in G_1. Hence, $\hat{c}_P(G_1) \geq i + 1$. By lemma 9 we have $\hat{c}_P(F_{i+1}', y) \geq i + 1$ for any vertex y in the subgraph of F_{i+1}' collapsed into x. Exactly as in the proof of lemma 4 one now shows that there is one such y with respect to which F_{i+1}' is linear. This completes the proof of our claim.[]

Corollary 5 *There exists a function $f(n) = \Omega(\log n)$ such that for any $n \geq 1$, there is a graph G on n vertices with $\hat{c}_P(G) = f(n)$ (for P = strong connectivity).[]*

Corollary 5 resolves the open question posed in [4].

2.3 Avoiding Cycles or Subdivisions

So far we have considered the number of iterations that algorithm 1 requires in the worst case. In this section we take the analysis one step further by examining the sequential running time of various natural variants of algorithm 1.

We restrict our attention to 2-edge-connectivity. We first note that algorithm 1 has a worst-case running time of $\Theta(m + n \log n)$ on a graph in the sample $F_0, F_1, \ldots$ for 2-edge-

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

The authors of [4] leave as an open question whether algorithm 3 runs in a constant number of iterations. Lemma 7 fails to resolve this question because the set of edges in $I - F$ turns out to be a *minimum* augmentation for F in H, i.e., a minimal augmentation of smallest cardinality. It can be shown, however, that even though it chooses a minimum augmentation at each step, algorithm 3 may need at least i iterations on input graph F'_i. Here are the details of the argument.

Lemma 8 *Fix an iteration of the while-loop of algorithm 3. 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 sketch. The lemma follows from the following two facts: $F + A$ is strongly connected iff it contains an inverse branching (rooted at x) and all branchings in H have the same number of edges.[]

Let P be a digraph property satisfying $(C1')$ and $(C2')$. For a P-graph 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_k is a minimal strongly connected spanning subgraph of H, 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 *minimum* augmentation for T in H_{i-1}. The integer k is the *length* of the trace. Let $\hat{c}_P(H, x)$ denote the maximum length of a trace of H with respect to x and let $\hat{c}_P(H)$ stand for $max\{\hat{c}_P(H, x) : x \in V(H)\}$.

Lemma 9 *If P satisfies conditions $(C1')$, $(C2')$, $(C3')$ and if H' is an essential contraction of P-graph H, then $\hat{c}_P(H', x) \leq \hat{c}_P(H, y)$ for any vertex x in H' and any node y of H collapsed into x.*

Proof sketch. The proof is very similar to the proof of lemma 5: one shows that if there is a minimum augmentation trace t' for H' with respect to x, then there is a minimum augmentation trace t, at least as long as t', for H with respect to any vertex collapsed into x. As in the proof of lemma 5 this can be shown by induction on the length of t'. The key

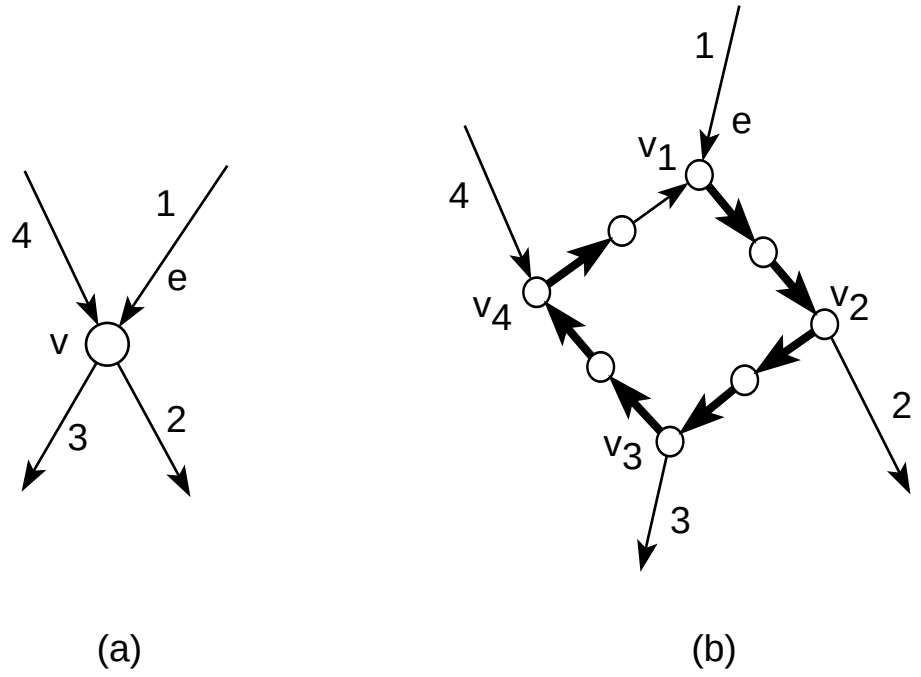


Figure 3: F_0', F_1', and F_2'.

Proof. The sizes of the graphs F_i' in a sample for strong connectivity constructed as described above are the same as those for the graphs F_i in a sample for 2-edge-connectivity (see lemma 3). As in the proof of corollary 1 we obtain a graph on n vertices having the same asymptotic size bound for arbitrary n by subdividing edges in the graph F_i' with the largest number of vertices that is less than n.[]

Lemma 7 is closely related to an open question posed in [4]. The following definitions from [4] will be needed. Let G be a directed graph. An *inverse branching* rooted at x is a spanning tree of G 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 [GKRST] the following algorithm is given for finding a minimal strongly connected spanning subgraph in a graph.

Algorithm 3. *Computing a minimal strongly connected spanning subgraph H of G.*
Input Strongly connected graph G.
Output Minimal strongly connected spanning subgraph H of G

Let us call a path in G' all of whose edges are in $B \cup B'$ a *good path*. We denote the root of $T(v)$ by $r(v)$ for any v in G_1 (see step (4) of algorithm 2' for the definition of $T(v)$). We first show that there is a good path from y to the root $r(v)$ of $T(v)$ in G' for any v in G_1. We show this by induction on the depth of v in T (see step (2) of algorithm 2'). If $depth(v) = 0$, then $v = x$ and $r(v) = y$; hence, the base case holds. Assume inductively that the claim holds for any v of depth k. Consider a vertex v of depth $k+1$. Let u be the father of v in T. By the induction assumption there is a good path from y to the root $r(u)$ of $T(u)$. Let $(w, r(v))$ be the edge in G' corresponding to edge (u, v) of T. From steps (4), (5), and (6) of algorithm 2' it follows that there is a good path in $T(u)$ 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 $r(v)$.

Now consider the case where a vertex w in G' is contained in the gadget of some vertex v but is different from the root of $T(v)$. Vertex w is reachable in $T(v)$, using only edges of B', either from y or from a vertex w' 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 whose head is w'. As above one argues that w'' is reachable from the root of its gadget using only edges of B'. We conclude that there is a good path from y to w.[]

We now consider an important property of digraphs: strong connectivity.

Lemma 6 *Strong connectivity satisfies properties* $(C1')$-$(C4')$.

Proof. Similar to proof of lemma 2 (use characterization of a strongly connected graph as a connected graph each of whose edges lies on a directed cycle).[]

To construct a sample for strong connectivity, start with the graph F_0' consisting of a directed cycle of length 2. Construct F_i by running algorithm 2' on input F_{i-1} for $i = 1, 2, \ldots$. In step (1) of algorithm 2' double each redundant edge in F_{i-1}. Choose as the gadget for the representatives $v_1, \ldots, v_d$ of v the directed cycle $v_1, v_1', v_2, v_2', \ldots, v_d, v_d', v_1$ consisting of the representatives of v and a set $\{v_1', \ldots, v_d'\}$ of new vertices. Note that at this point many different samples could be constructed by choosing different numberings in algorithm 2' but this will not affect the size of the graphs in the sample. The first three graphs F_0', F_1', and F_2' in a possible sample for strong connectivity are depicted in figure 3. In the remainder of this paper the sequence $F_0', F_1', \ldots$ stands for an arbitrary but fixed sample for strong connectivity constructed according to the rules we have just described.

Lemma 7 *There exists a function* $f(n) = \Omega(\log n)$ *such that there is a graph on* n *nodes of complexity* $f(n)$ *with respect to strong connectivity for any* $n \geq 1$.

(4) For each v in G_1 of degree d, number the representatives of v as follows: fix a forward branching $T(v)$ that spans the gadget constructed for v in step (3) and that is rooted at a representative w for v. Assign numbers $1, 2, \ldots, d$ to the representatives of v in nondecreasing order of their distance from w in $T(v)$ (hence, w is numbered 1).

(5) For a vertex v in G_1 of indegree p and outdegree q, number the $p+q$ edges incident on v as follows: if $v \neq x$, assign number 1 to the unique incoming edge marked in step (2); the outgoing edges receive numbers $2 \ldots 2+q-1$ and the numbers $2+q \ldots p+q$ are assigned to the other incoming edges. If $v = x$, then assign the numbers $1 \ldots p$ to the outgoing edges and the numbers $p+1 \ldots p+q$ to the incoming edges.

(6) For each edge (u, v) in G_1 that is the ith edge incident on u and the jth edge incident on v, add an edge from the ith representative of u to the jth representative of v in G' (with respect to the numbering defined in step (4)).

Theorem 4 *The graph G' has complexity at least $k+1$ with respect to some vertex y and is linear with respect to y, provided that P satisfies conditions $(C1')$-$(C4')$.*

Proof. Fix a trace t for G with respect to vertex x of length k. Since G is linear with respect to x, it is of the form $T + A$ where T is a forward branching rooted at x and A is a set of essential edges in G. Hence, we can extend trace t to a trace t' for G_1 with respect to x having length $k+1$. Thus, $c_P(G_1, x) \geq k+1$.

The graph G_1 is a contraction of G'. Hence, by condition $(C3')$ the graph G' is a P-graph. By condition $(C4')$, the definition of a gadget, and algorithm 2', G_1 is an essential contraction of G'. Since $c(G_1, x) \geq k+1$, lemma 5 gives us $c_P(G', y) \geq k+1$ for any vertex y collapsed into x. It remains to be shown that G' is linear with respect to at least one such vertex y. Let y be the root of forward branching $T(x)$ spanning the gadget for x in G' (see step (4) of algorithm 2'). Let B denote the set of edges in G' corresponding to edges in G_1. Note that these edges are exactly the redundant edges in G'. We shall establish that G' is linear with respect to y by proving that there exists a forward branching in G' rooted at y and containing all edges of B.

For any vertex $w \neq y$ in G' belonging to the gadget of some vertex v and having no incoming edge in B, define $e(w)$ to be the unique edge in $T(v)$ whose head is w. Let B' be the set $\{e(w) : w \in V(G'), w \neq y$ and w has no incoming edge in $B\}$. We shall now prove that every vertex in G' is reachable from y by edges in $B \cup B'$. Since each vertex in G' 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 G' rooted at y; hence, G' is linear with respect to y.

branching in H' rooted at x, A' is a minimal augmentation for T' in H', and there is a trace for $T' + A'$ with respect to x having length k. Since each collapsed subgraph in H is strongly connected (condition $(C2')$), we can combine T' with forward branchings for the collapsed subgraphs to form a forward branching T in H rooted at an arbitrary vertex y collapsed into x. Let A be the edges of A' plus the edges in the collapsed subgraphs 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 there exists a trace for $T + A$ with respect to y having length at least k. By condition $(C3')$ an edge of H' is redundant iff the corresponding edge of H is redundant. Moreover, all edges in the collapsed subgraphs in H are essential. Therefore, the forward branching T is an optimal branching rooted at y and A is a minimal augmentation for T in H. Hence, there is a trace for H with respect to y having length at least $k + 1$.[]

Corollary 4 *If H' is an essential contraction of H, then $c_P(H') \leq c_P(H)$.[]*

Condition $(C4')$ is identical with condition $(C4)$. As in the undirected case, we call the graph G_S a *gadget* for S. To illustrate the notion of a gadget, consider strong connectivity. Let $\mid S \mid = \mid S' \mid$ and $S \cap S' = \emptyset$. The directed cycle alternating between vertices of S and those of S' is a gadget for S.

It turns out that conditions $(C1')$-$(C4')$ guarantee the existence of a sample for a digraph property. Algorithm 2' below provides a means of constructing such a sample for any digraph property P satisfying these conditions. It is similar to algorithm 2 although more involved. A P-graph H is *linear* with respect to vertex x if its edge set can be partitioned into a forward branching rooted at x and a set of essential edges.

Algorithm 2': *Increasing the complexity of a digraph.*
Input P-graph G of complexity k with repect to x and linear with respect to x.
Output P-graph G' of complexity $\geq k + 1$ with respect to some vertex y and linear with respect to y.

(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) Let T be a forward branching in G_1 rooted at x. Mark all edges of G_1 that are in T.

(3) For each v in G_1 of degree d, do the following: create d new vertices $v_1, \ldots, v_d$ in G' which we call the *representatives* of v. 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 vertices of G_1).

forward branching T_H rooted at a fixed vertex x of H (and containing a minimum number of P-redundant edges). We call T_H an *optimal branching* in H rooted at x and A (see algorithm 1), as before, a *minimal augmentation* for T_H in H.

The following result is the analog of theorem 1 for the directed case.

Theorem 3 *Algorithm* $1'$ *computes a minimal spanning* P*-subgraph of* G *for any digraph property* P *satisfying* $(C1')$ *and* $(C2')$.

Proof. Similar to proof of theorem 1.[]

Given a digraph property P (satisfying $(C1')$ and $(C2')$), a *trace* for P-graph H *with respect to root* x is a sequence $H_0, \ldots, H_k$ of subgraphs of H such that $H_0 = H$, H_k is a minimal spanning P-subgraph of H, 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}. We call k the *length* of the trace. The *P-complexity* of H with respect to x is the maximum length of a trace for H with respect to vertex x; it is denoted by $c_P(H, x)$. We define the *P-complexity* of H, denoted by $c_P(H)$, as $max\{c_P(H, x) : x \in V(H)\}$. As in the undirected case, we denote the P-complexity of H by $c(H)$ if the property P is understood, and we call an infinite sequence $H_0, H_1, \ldots$ of digraphs such that $c_P(H_i) \geq i$ $(i \geq 0)$ a *sample for* P.

The notions of *contraction* and *essential contraction* of a graph H are defined as for undirected graphs. If H' is a contraction of H, then there is a natural correspondence between the vertices of H and those of H'; the correspondence is 1-1 for those nodes in H not belonging to a collapsed subgraph. We say that a node in H is *collapsed* into the corresponding node in H'.

Condition $(C3')$ is identical with condition $(C3)$. The following result is reminiscent of lemma 1.

Lemma 5 *If* H' *is an essential contraction of* P*-graph* H*, then the complexity of* H' *with respect to a vertex* x *is not larger than the complexity of* H *with respect to any vertex* y *collapsed into* x.

Proof. By condition $(C3')$ digraph H' has property P and $c_P(H')$ is indeed well-defined. We prove the following: if H' is an essential contraction of H and there is a trace t' for H' with respect to some root x, then there is a trace t for H at least as long as t' with respect to any vertex belonging to the set of vertices of H collapsed into x. We show this by induction on the length of t'.

The statement trivially holds if the length of t' is 0. Assume it holds if the length of t' is $k \geq 0$. Suppose that there is a trace t' for H' with respect to x having length $k + 1$. Hence, H' is of the form $T' + A'$ where T' and A' have the following properties: T' is an optimal

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

Corollary 2 *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 4. 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 4, 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 and hence $F_0, F_1, \ldots$ is a sample for biconnectivity as well. Thus, we get the following result.

Corollary 3 *For biconnectivity, there exists a function $f(n) = \Omega(\log n)$ such that there is a graph on n vertices with complexity $f(n)$ for any $n \geq 1$.*

Proof. Similar to proof of corollary 1.[]

Corollaries 1 and 3 establish that Algorithm 1 takes $\Omega(\log n)$ iterations for 2-edge-connectivity and biconnectivity.

2.2 Properties of Digraphs

Let G be a directed graph. A *forward branching* ([4]) rooted at x is a spanning tree of G in which x has in-degree zero and all other vertices have in-degree one. We assume that all forward branchings are rooted at a fixed vertex x of G.

It turns out that the development carried out above for undirected graph properties carries over to digraphs with some modifications. We distinguish conditions and algorithms for the directed case from the corresponding conditions and algorithms for the undirected case by adding a single quote, e.g., algorithm $1'$ computes a minimal spanning subgraph with property P.

The definitions of digraph property, P-graph, P-subgraph, and P-redundant and P-essential edges carry over from the undirected case without change. Condition $(C1')$ is identical with condition $(C1)$. Observation 1 holds for any digraph property satisfying $(C1')$. For condition $(C2')$ we require that the graph be strongly connected.

Algorithm $1'$ finds a minimal spanning P-subgraph of digraph G. It has a structure similar to that of algorithm 1; instead of computing a spanning tree in step (2.1), it computes a

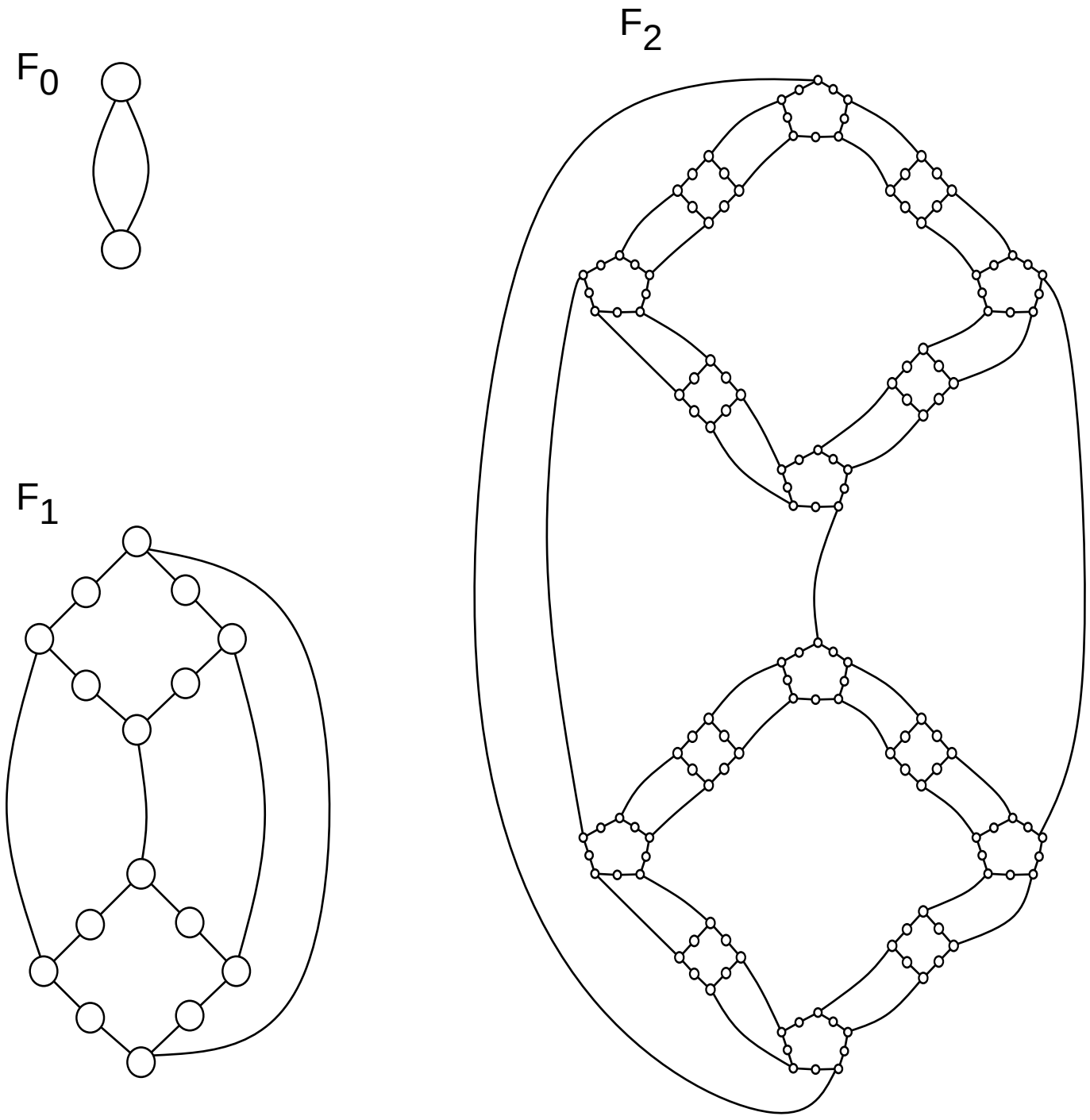


Figure 2: F_0, F_1, and F_2.

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 (see also Figure 1).[]

By theorem 2 and lemma 2, algorithm 2 may be used to construct a sample for 2-edge-connectivity. We fill in the details for steps (1) and (3) of algorithm 2. 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 2. 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 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 2. Note that we may obtain different samples for different edge numberings (step (2) of algorithm 2) 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 3 *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 1 *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 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 4 *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.[]*

Algorithm 2 below provides a method to compute 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 2: *Increasing the complexity of a graph.*
Input Linear graph G of complexity k.
Output Linear graph G' of 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 $e_1, \ldots, e_d$ (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 $v_1, \ldots, v_d$ 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 2 *G' is a linear graph of 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 $c(G_1) \geq k+1$.

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 $c(G_1) \geq k+1$, lemma 1 gives us $c(G') \geq k+1$. 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 2 *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

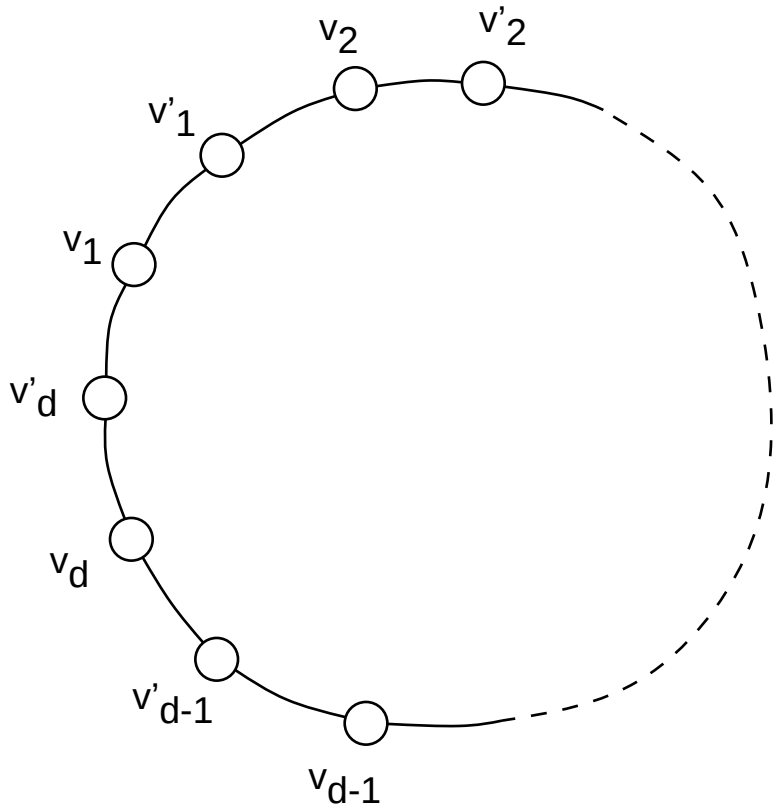


Figure 1: A gadget for 2-edge-connectivity

collapsing disjoint subsets of vertices of H whose induced subgraphs in H have property P (into single vertices); 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. Condition $(C3)$ implies that 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 1 *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 complexity of H' is well-defined. We prove the lemma by induction on the complexity of H'. If $c(H') = 0$, then certainly $c(H) \geq c(H')$.

Let $c(H') = k > 0$. Let $T' + A'$ be a graph of 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 $c(T+A) \geq k-1$. Moreover, by $(C3)$ and the definition of H', the edges of A are essential in $T+A$. We conclude that $c(H) \geq k$.[]

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. To illustrate the notion of a gadget, consider 2-edge-connectivity. Let $S = \{v_1, \ldots v_d\}$ and let $S' = \{v'_1, \ldots, v'_d\}$. The cycle alternating between the vertices of S and those of S' is a gadget for S (see Figure 1). 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'.

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.[]

By the proof of theorem 1 and observation 1 the number of iterations of algorithm 1 is bounded from above by the number of P-redundant edges in G. In many cases a much stronger bound holds. In the sequel, n denotes the number of vertices of the input graph G. In [KR] it is shown that $O(\log n)$ iterations of algorithm 1 yield a minimal 2-edge-connected spanning subgraph of G or a minimal biconnected spanning subgraph of G. Below we show that these bounds are tight, i.e., the number of iterations of algorithm 1 on a graph with n nodes is $\Theta(\log n)$ in the worst case for the above properties. Instead of deriving lower bounds specifically for 2-edge-connectivity and biconnectivity, we proceed within the framework of general graph properties; accordingly, the results we shall derive will be applicable to other graph properties.

To capture the worst-case behavior of algorithm 1, we introduce the concept of 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. More precisely, we define a *trace* for P-graph H to be a sequence $H_0, H_1, \ldots, H_k$ of subgraphs such that $H_0 = H$, H_k is a minimal spanning P-subgraph of 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}. The integer k is the *length* of the trace. The P-complexity of H is the maximum length of a trace 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. 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.

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 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 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

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. We restrict our attention to 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 P-graph;
$(C2)$ any P-graph is connected.

As an immediate consequence of condition $(C1)$ we note the following 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 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 minimum number of P-redundant 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 $(C2)$.*

linear-time algorithm for finding a minimal spanning strongly connected subgraph is given in [12]; however, this algorithm is incorrect ([13]).

In this paper we generalize the high-level algorithm of ([4], [9]) into a general algorithm for finding a minimal spanning subgraph with a given property and analyze its worst-case complexity. In section 2 we give a tight lower bound of $\Omega(\log n)$ on the worst-case number of iterations of the algorithm for 2-edge-connectivity, biconnectivity, and strong connectivity; this leads to an $\Omega(m+n \log n)$ lower bound on the sequential running time of these algorithms. We strengthen this bound by showing that the lower bound on the running time holds if we allow various types of graph contractions. The method we describe for constructing worst-case graphs is fairly general and may be applicable to other graph properties.

In section 3 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 we perform certain contractions and transformations on the current graph so that its size goes down by a constant factor with each iteration. This result also reduces the work performed by the parallel algorithms for these problems by a logarithmic factor. Finally, we provide some strong evidence that a similar strategy will not lead to a linear-time algorithm for the strong connectivity property.

Note: We have recently learned that Han and Tarjan ([6]) have independently discovered linear time algorithms for finding a minimal 2-edge-connected spanning subgraph and for finding a minimal 2-connected spanning subgraph (see section 3).

2 Worst-Case Behavior of Algorithms for Finding Minimal Subgraphs

2.1 Properties of Undirected Graphs

In this section we describe an algorithm for finding a minimal spanning subgraph of a graph for various properties of undirected graphs. We examine the number of iterations the algorithm requires in the worst case. We describe a fairly general technique for constructing worst-case graphs. We apply the technique to 2-edge-connectivity and show that it works for biconnectivity as well. In the next section we generalize the development to directed graphs.

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.

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.

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 and k-edge-connected spanning subgraph are NP-hard for any fixed $k \geq 2$. Yannakakis ([15]; see also [10]) 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 we 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 ([11], [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.

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. We also note that a

The Complexity of Finding Minimal Spanning Subgraphs *

Pierre Kelsen*
Vijaya Ramachandran*

Department of Computer Sciences
University of Texas, Austin, TX 78712

February 6, 1991

Abstract

Let P be a property of graphs (directed or undirected). 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 an algorithm for this problem and prove that it is correct under some rather weak assumptions about P. We then analyze the number of iterations of this algorithm. By suitably restricting the graph properties, we devise a general technique to construct graphs for which the algorithm requires a large number of iterations.

We apply the above technique to three concrete graph properties: 2-edge-connectivity, biconnectivity, and strong connectivity. We obtain a tight lower bound of $\Omega(\log n)$ on the number of iterations of the algorithm for finding minimal spanning subgraphs with these properties; this resolves open questions posed earlier with regard to these properties. This also implies that the worst case sequential running time of the algorithm for these three properties is $\Omega(m+n\log n)$. We then give refinements of the basic algorithm that yield the first linear-time algorithms for finding a minimal 2-edge-connected and a minimal biconnected spanning subgraph of a graph. Finally, we provide evidence that the problem of refining the algorithm to find a minimal strongly connected spanning subgraph in linear time is more difficult.

o

*This work was supported in part by NSF grant CCR89-10707.

E-mail addresses for the authors: kelsen@cs.utexas.edu (Pierre Kelsen) and vlr@cs.utexas.edu (Vijaya Ramachandran).