# Fast Algorithms for $k$-Shredders and $k$-Node Connectivity Augmentation *

*Joseph Cheriyan* †  *Ramakrishna Thurimella* ‡

16 December 1996

**Abstract**: A *$k$-separator* (*$k$-shredder*) of an undirected graph is a set of $k$ nodes whose removal results in two or more (three or more) connected components. Let the given (undirected) graph be $k$-node connected, and let $n$ denote the number of nodes. Solving an open question, we show that the problem of counting the number of $k$-separators is #P-complete. However, we present an $O(k^2n^2 + k^3n^{1.5})$-time (deterministic) algorithm for finding all the $k$-shredders. This solves an open question: efficiently find a $k$-separator whose removal maximizes the number of connected components. For $k \geq 4$, our running time is within a factor of $k$ of the fastest algorithm known for testing $k$-node connectivity. One application of shredders is in increasing the node connectivity from $k$ to $(k+1)$ by efficiently adding an (approximately) minimum number of new edges. Jordán [JCT(B) 1995] gave an $O(n^5)$-time augmentation algorithm such that the number of new edges is within an additive term of $(k-2)$ from a lower bound. We improve the running time to $O(\min(k, \sqrt{n})k^2n^2 + (\log n)kn^2)$, while achieving the same performance guarantee. For $k \geq 4$, the running time compares favorably with the running time for testing $k$-node connectivity.

## 1 Introduction

Let $G = (V, E)$ be an undirected, simple graph. A *node separator* $S$ of $G$ is an (inclusionwise) minimal subset $S \subset V$ such that $G \backslash S$ is disconnected. Similarly, an *edge separator* is an (inclusionwise) minimal subset $C \subseteq E$ such that $G \backslash C$ is disconnected. One of the differences between edge connectivity and node connectivity is that the deletion of an edge separator always results in two connected components, but the deletion of a node separator results in two or more connected components. Our main contribution is the study of (minimum-cardinality) node separators whose removal results in three or more connected components. We call a separator $S$ of $G$ a *shredder* if $G \backslash S$ has at least three connected components. For example, if $G$ is a tree, each node of degree $\geq 3$ forms a singleton shredder. For another example, if $G$ is the complete bipartite graph $K_{3,3}$, each part of the bipartition forms a shredder. A separator (shredder) of a graph is called a $k$-separator ($k$-shredder) if it has exactly $k$ nodes. A graph $G = (V, E)$ is said to be *$k$-node connected* if $|V| > k$ and the minimum cardinality of a separator is $\geq k$. We focus on the node connectivity, so except for the introduction, "connectivity" means node connectivity. The number of nodes, $|V|$, is denoted by $n$.

*A preliminary version of this paper has appeared in the Proc. of the 28th ACM S.T.O.C. (1996), pp. 37–46.

†Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. Supported in part by NSERC grant no. OGP0138432 (NSERC code OGPIN 007). email: `jcheriyan@dragon.uwaterloo.ca`

‡Department of Mathematics and Computer Science, University of Denver, 2360 S. Gaylord St., Denver CO 80208. Supported in part by NSF Research Initiation Award grant CCR-9210604. email: `ramki@cs.du.edu` URL: `http://www.cs.du.edu/~ramki`

We present an $O(k^2n^2 + k^3n^{1.5})$-time (deterministic) algorithm for finding all the $k$-shredders of a $k$-node connected graph. This solves an open question raised by Jordán [J 95]: efficiently find a $k$-separator of a $k$-node connected graph whose removal maximizes the number of connected components. For $k \geq 4$, our running time is within a factor of $k$ of the running time of the fastest (deterministic) algorithm known for the basic problem of determining whether a graph is $k$-node connected. It may not be possible to find all the $k$-shredders within a time bound that is less than the time bound for testing $k$-node connectivity, though we have no proof of such a lower bound. For $k \leq 3$, linear-time algorithms are known for testing $k$-node connectivity, while for $k \geq 4$, the fastest algorithm runs in time $O(\min[kn^2 + k^4n,\ k^2n^2])$ [HRG 96]. We also describe a dynamic algorithm for maintaining the set of all the $k$-shredders of a $k$-node connected graph over a sequence of edge insertions/deletions. The time per edge update is $O(|E| + (\min(k, \sqrt{n}) + \log n)kn)$.

Counting the number of $k$-separators of a $k$-node connected graph is a fundamental problem. For example, the recent approximation scheme of Karger [K 95] for estimating network reliability with respect to edge failures is based on counting (and generating) all the minimum-cardinality edge separators in polynomial time. Karger's work raised the question whether this method extends to approximating network reliability w.r.t. node failures. We show that computing the number of minimum-cardinality node separators is #P-complete, thus resolving an open question in the area. However, we show that the number of $k$-shredders of a $k$-node connected graph is $O(k^2n + n^2)$. We present a key lemma on so-called meshing shredders in Section 4.

One application of shredders is to an important (and, as yet, partially solved) problem in network design. A basic goal in network design is: given a (nonnegative) cost for each edge of the complete graph, construct a subgraph of minimum cost satisfying certain edge/node connectivity requirements. The edge costs may be either zero/one or not. Problems with zero/one costs on the edges are usually regarded as augmentation problems: Given an initial graph (whose edges have zero cost) increase the edge/node connectivity by adding a minimum number of new edges (each new edge costs one). For instance, given a tree, one may want to add the minimum number of new edges to achieve 3-node connectivity. Readers interested in network design with arbitrary edge costs are referred to [GW 95] and [RW 95], and readers interested in edge/node connectivity augmentation problems for both graphs and directed graphs are referred to [F 94].

Let us focus on node connectivity augmentation problems: given a graph, increase the node connectivity to $k'$ by adding the minimum number of new edges. The case $k' = 2$ was solved by Eswaran & Tarjan [ET 76], and later Hsu & Ramachandran [HR 93] gave a linear-time algorithm. The case $k' = 3$ was solved by Watanabe & Nakamura [WN 90], and a linear-time algorithm was given by Hsu & Ramachandran [HR 91]. The case $k' = 4$ was solved by Hsu [H 95] using an $O(|E| + n \log n)$-time algorithm, and earlier Hsu [H 92] gave an almost linear-time algorithm to increase the node connectivity from three to four. Whether there is an efficient algorithm for the node connectivity augmentation problem for arbitrary $k'$ is an outstanding open question.

Jordán [J 95] recently presented an $O(n^5)$-time *approximation* algorithm for the problem of adding the minimum number of new edges to a $k$-node connected graph to make it $(k+1)$-node connected. The difference between the number of new edges added by Jordán's algorithm and a lower bound on the number of new edges is at most $k - 2$. We present an improved version of Jordán's algorithm [J 95] that runs in time $O(\min(k, \sqrt{n})k^2n^2 + (\log n)kn^2)$ and achieves the same performance guarantee. For $k \geq 4$, the running time of our algorithm compares favorably with that of the fastest algorithm known for testing $k$-node connectivity. For $4 \leq k = O(\sqrt{\log n})$, our running time is within a logarithmic factor of the running time for testing $k$-node connectivity, and for larger $k$, our running time is within a factor of $\min(k, \lceil n/k \rceil, \sqrt{n})$ of the running time for testing $k$-node connectivity. The proof of correctness of our algorithm is based on Jordán's proof [J 95], but is simpler. Jordán also has a simpler proof in [J]. Moreover, for $n \geq (2k+1)$, Jordán [J 96] improves the performance guarantee of his algorithm in [J 95] to show a slack of roughly $k/2$.

The rest of the paper is organized as follows. Section 2 has definitions, notation and basic results. Section 3 describes our algorithm for finding all the $k$-shredders of a $k$-node connected graph, and also describes a dynamic algorithm for maintaining the set of all the $k$-shredders over a sequence of edge updates. Section 4 has our results on counting the number of $k$-separators and $k$-shredders in a $k$-node connected graph. Section 5 describes our augmentation algorithm and its proof of correctness.

## 2 Definitions, notation and preliminaries

For a subset $S'$ of a set $S$, $S \backslash S'$ denotes the set $\{x \in S : x \notin S'\}$. Let $G = (V, E)$ be a (finite, undirected) graph without loops or multiedges. (Since this paper studies node connectivity, multiedges play no role. For example, if we add to $G$ a copy of an existing edge, then $G$ stays the same.) $V(G)$ and $E(G)$ stand for the node set and the edge set of $G$. An edge incident to nodes $v$ and $w$ is denoted by $vw$. An $x \leftrightarrow y$ path refers to a path whose end nodes are $x$ and $y$. We call two paths *openly disjoint* if every node common to both paths is an end node of both paths. Hence, two (distinct) openly disjoint paths have no edges in common, and possibly, have no nodes in common. A set of two or more paths is called openly disjoint if the paths are pairwise openly disjoint. For a subset $V' \subseteq V$, the *induced subgraph* of $V'$, $G[V']$, has node set $V'$ and edge set $\{vw \in E : v, w \in V'\}$. For a subset $S \subseteq V$, $G \backslash S$ denotes $G[V \backslash S]$. We abuse the notation for singleton sets, e.g., we use $v$ for $\{v\}$. By a *component* (or connected component) of a graph, we mean a maximal connected subgraph, as well as the node set of such a subgraph. Hopefully, this will not cause confusion. The number of components of $G$ is denoted by $\#\mathrm{c}(G)$. For a subset $Q \subseteq V$, $N_G(Q)$ or $N(Q)$ denotes the set of neighbors of $Q$ in $V \backslash Q$, $\{w \in V \backslash Q : wv \in E,\ v \in Q\}$. The function $|N(Q)|$ on subsets $Q$ of $V$ is submodular, i.e., for all $Q_1, Q_2 \subseteq V$,

$$|N(Q_1)| + |N(Q_2)| \geq |N(Q_1 \cap Q_2)| + |N(Q_1 \cup Q_2)|.$$

Recall that a *separator* $S$ of $G$ is an (inclusionwise) minimal subset $S \subset V$ such that $G \backslash S$ has at least two components. $S$ is said to *separate* nodes $v$ and $w$ if the two nodes are in different components of $G \backslash S$. Clearly, for each component $D$ of $G \backslash S$, $N(D) = S$, and each $v \in S$ has a neighbor in each component of $G \backslash S$. We call a separator $S$ of $G$ a *shredder* if $G \backslash S$ has at least three components. A pair of separators $S, T$ is called *nonmeshing* if $T$ has a nonempty intersection with at most one component of $G \backslash S$, otherwise, $S$ and $T$ are said to *mesh*. In other words, separators $S$ and $T$ mesh if $T$ has nonempty intersections with at least two components of $G \backslash S$. A family (i.e., set) of separators is called *nonmeshing* if it is pairwise nonmeshing.

Variants of the next lemma have appeared before. Lemma 2.2 of [J 95] implies a special case of the lemma.

**Lemma 2.1** *If $S$ and $T$ are (not necessarily minimum) separators of a (not necessarily $k$-connected) graph $G$ such that $S$ and $T$ mesh, then every component of $G \backslash T$ (or $G \backslash S$) has a node of $S$ (or $T$). Hence, the meshing relation on pairs of separators is symmetric.*

**Proof**: The key point is this:

every component of $G \backslash T$ contains a node of $S$.

To see this, consider a node $v \in V \backslash (S \cup T)$ and suppose that it belongs to a component, say, $D_1$ of $G \backslash S$. (If $V \backslash (S \cup T)$ is empty, then the proof is done.) Focus on a node $t \in T$ that belongs to another component, say, $D_2$ of $G \backslash S$. Such a node exists since $S$ and $T$ mesh. Now focus on the component, say, $D'$ of $G \backslash T$ that contains $v$. Since $T$ is (inclusionwise) minimal, $t$ has a neighbor, say, $t'$ in $D'$, and $D'$ contains a $v \leftrightarrow t'$ path. Since $S$ separates $v$ from $t$, it is clear that this $v \leftrightarrow t'$ path contains a node of $S$ (possibly, $t' \in S$). Hence, $D'$ contains a node of $S$. Our claim follows. Since $G \backslash T$ has at least two components, and each contains a node of $S$, $T$ and $S$ mesh. The lemma follows. $\square$

A separator (shredder) of a graph is called a $k$-separator ($k$-shredder) if it has exactly $k$ nodes. A graph $G$ is said to be *$k$-node connected* ($k$-connected) if $|V(G)| \geq k+1$, and $G$ has no separators of cardinality $\leq (k-1)$ (i.e., the deletion of any set of $< k$ nodes results in a connected graph). An edge $vw$ of a $k$-connected graph $G$ is called *critical* (w.r.t. $k$-connectivity) if $G \backslash vw$ is not $k$-connected (i.e., $G \backslash vw$ has a $(k-1)$-separator).

A *tight set* of a $k$-node connected graph $G = (V, E)$ is a node set $Q$ such that $|N(Q)| = k$ and $|V \backslash Q| \geq (k+1)$. In other words, a tight set is either a component obtained by deleting a $k$-separator $S$ from $G$, or the union of two or more (but not all) components of $G \backslash S$. See Section 5.1 for examples and an application. The next lemma on tight sets is used often in Section 5. The proof follows from the submodularity of $|N(Q)|$ over $Q \subseteq V$. Also, see [J 95, Lemma 1.2].

**Lemma 2.2** *Given a $k$-connected graph $G = (V, E)$, and tight sets $X, Y$ with $X \cap Y \neq \emptyset$ and $|V \backslash (X \cup Y)| \geq k$, the set $X \cap Y$ is tight, and there is no edge with one end in $X \backslash Y$ (or $Y \backslash X$) and the other end in $Y \backslash (X \cup N(X \cap Y))$ (or $X \backslash (Y \cup N(X \cap Y))$). Moreover, if $|V \backslash (X \cup Y)| \geq k+1$, then the set $X \cup Y$ is tight.*

## 3 A fast algorithm for finding all $k$-shredders

This section presents an efficient algorithm for finding all the $k$-shredders of a $k$-connected graph. For ease of description, we assume that the input graph is $k$-connected, but it is straightforward to modify the algorithm to include a test for $k$-connectivity. The algorithm is based on the next result. See Figure 1 for an illustration of the algorithm.

**Proposition 3.1** *Let $G$ be a $k$-connected graph and let $v, r$ be a pair of nodes. The number of $k$-shredders separating $v$ and $r$ is at most $n$, and the family of $k$-shredders separating $v$ and $r$ is nonmeshing.*

**Proof**: Let $P_1, \ldots, P_k$ be an arbitrary set of $k$ openly disjoint $v \leftrightarrow r$ paths. Every $k$-separator $S$ separating $v$ and $r$ has exactly one (distinct) node from each of the paths $P_1, \ldots, P_k$. Let $Q$ denote $V(P_1) \cup \ldots \cup V(P_k)$. If $S$ is a $k$-shredder, then $G \backslash S$ has at least three components $D_1, D_2, D_3, \ldots$. Suppose that $v \in V(D_1)$ and $r \in V(D_2)$. The key point is:

> $D_3$ stays connected, even after removing all nodes of $P_1, \ldots, P_k$ (i.e., $D_3$ is a component of $G \backslash Q$), because $D_3$ has no node of $Q$.

The bound on the number of $k$-shredders separating $v$ and $r$ follows, since there is a distinct component in $G \backslash Q$ for each distinct $k$-shredder separating $v$ and $r$. Suppose that two of the $k$-shredders separating $v$ and $r$, say, $S$ and $T$, mesh. Then, by Lemma 2.1, every component $D_1, D_2, D_3, \ldots$ of $G \backslash S$ contains a node of $T$. Hence, $Q$ has at most $(k-1)$ nodes of $T$. We have the desired contradiction, since at least one of the $v \leftrightarrow r$ paths $P_1, \ldots, P_k$ "survives" in $G \backslash T$. $\square$

The above result gives an $O(n^3)$ bound on the number of $k$-shredders. This bound can be improved somewhat, see Algorithm 1 in the box on page 6.

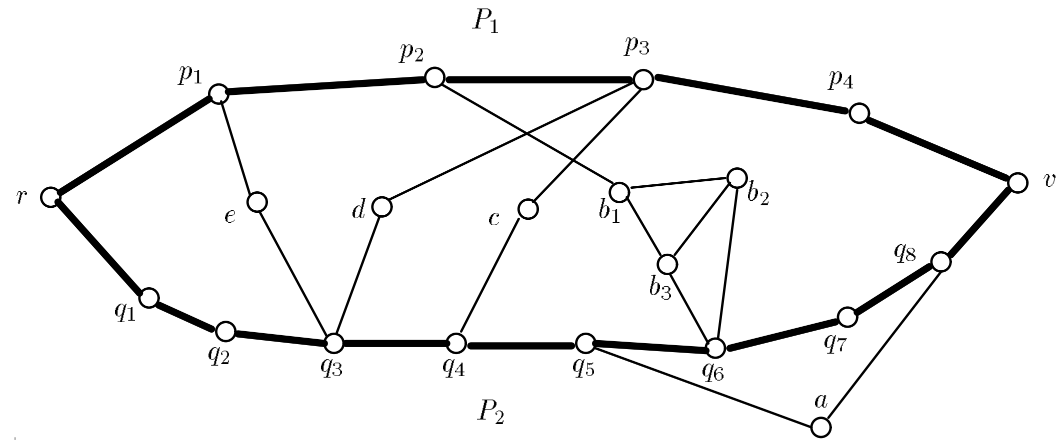


**Figure 1:** Illustrating algorithm *Shredders*$(r,v)$, using $k = 2$. $P_1$ and $P_2$ are two openly disjoint $r \leftrightarrow v$ paths. The components of $G \backslash (V(P_1) \cup V(P_2))$ are $D_1 = \{e\}$, $D_2 = \{d\}$, $D_3 = \{c\}$, $D_4 = \{b_1, b_2, b_3\}$, and $D_5 = \{a\}$. The candidate shredders are $N(D_1) = \{p_1, q_3\}$, $N(D_2) = \{p_3, q_3\}$, $N(D_3) = \{p_3, q_4\}$, and $N(D_4) = \{p_2, q_6\}$. Step 5 finds that $N(D_2)$ and $N(D_4)$ are incomparable, and discards both. The remaining candidate shredders are lexicographically ordered as $S_1 = N(D_1)$ and $S_2 = N(D_3)$. There are 5 bridges of $P_1 \cup P_2$, given by $D_1, \ldots, D_5$ and their open intervals are: $(1,1)$, $(1,2)$, $(2,2)$, $(1,3)$, and $(2,3)$. The union of the open intervals is $(1,3)$. Step 6 discards $S_2 = N(D_3)$, since the index of $S_2$ is in $(1,3)$. There is only one 2-shredder separating $r$ and $v$: $S_1 = N(D_1) = \{p_1, q_3\}$.

**Corollary 3.2** *The number of $k$-shredders in a $k$-connected graph is $O(k^2 n + n^2)$*

Algorithm *All-$k$-shredders* (see Algorithm 1 in the box on page 6) outputs all the $k$-shredders of a $k$-connected graph. The main subroutine *Shredders(r,v)* finds all the $k$-shredders separating two specified nodes $v$ and $r$. Let $y_1, \ldots, y_k$ be $k$ arbitrary nodes. A $k$-shredder either separates some $y_i$ from some $y_j$, $1 \leq i \neq j \leq k$, or separates $\{y_1, \ldots, y_k\}$ from some node $v \in V \backslash \{y_1, \ldots, y_k\}$. To handle the second possibility, our algorithm adds a new *root node* $z$ and the edges $zy_1, \ldots, zy_k$ (cf. [G 80]), and then finds all the $k$-shredders separating $z$ and $v$, for each node $v \in V \backslash \{z, y_1, \ldots, y_k\}$.

Focus on subroutine *Shredders(r,v)* (see Algorithm 2 in the box on page 6). We construct $k$ openly disjoint $v \leftrightarrow r$ paths $P_1, \ldots, P_k$. For $1 \leq i \leq k$, by an $r \to v$ path $P_i$ we mean the path $P_i$ oriented from $r$ to $v$. Let $Q$ denote the set of nodes of the paths $P_1, \ldots, P_k$, and let $D_1, \ldots, D_c$ denote the components of $G \backslash Q$. By a *candidate shredder* we mean the neighbor set $N(D_g)$ of a component $D_g$ of $G \backslash Q$ $(1 \leq g \leq c)$ such that $|N(D_g)| = k$, $N(D_g)$ has exactly one node from each path $P_1, \ldots, P_k$, and neither $r$ nor $v$ is in $N(D_g)$. We take each candidate shredder $S = N(D_g)$ to be a $k$-tuple by ordering the nodes in $S$ according to their occurrence in $P_1, \ldots, P_k$. A $k$-tuple $\langle u_1, u_2, \ldots, u_k \rangle$ is said to *precede* another $k$-tuple $\langle w_1, w_2, \ldots, w_k \rangle$ if for each $i$, $1 \leq i \leq k$, $u_i$ precedes $w_i$ on the $r \to v$ path $P_i$. If two $k$-tuples are incomparable (i.e., neither $k$-tuple precedes the other), then neither of the two corresponding candidate shredders is a $k$-shredder separating $r$ and $v$. In more detail, if the $k$-tuple $\langle u_1, u_2, \ldots, u_k \rangle$ for $S = N(D_s)$ and the $k$-tuple $\langle w_1, w_2, \ldots, w_k \rangle$ for $T = N(D_t)$ are incomparable, then there exist $i$ and $j$, $1 \leq i, j \leq k$, such that $u_i$ strictly precedes $w_i$ on the $r \to v$ path $P_i$ but $u_j$ strictly follows $w_j$ on the $r \to v$ path $P_j$. Hence, in $G \backslash T$, there is an $r \leftrightarrow v$ path via node $u_i$, $D_s$ and node $u_j$. Similarly, in $G \backslash S$ there is an $r \leftrightarrow v$ path via $w_j$, $D_t$ and $w_i$. Consequently, whenever *Shredders(r,v)* finds a pair of candidate shredders whose $k$-tuples are incomparable, it discards both candidate shredders. After this round of elimination, we are left

**Algorithm 1** *All-$k$-shredders*

**Input**: A $k$-connected graph $G = (V, E)$.
**Output**: The family of $k$-shredders of $G$, stored in $\mathcal{L}$.

(1) $\mathcal{L} = \emptyset$.

(2) Choose (arbitrarily) $k$ nodes $y_1, \ldots, y_k$.

(3) **For** each pair $y_i, y_j$, $1 \leq i < j \leq k$ **do**

$\mathcal{L}' =$*Shredders*$(y_i,\, y_j)$;
$\mathcal{L} = \mathcal{L}' \bigcup \mathcal{L}$.

(4) Add a new node $z$, and add the edges $zy_1, \ldots, zy_k$.

(5) **For** each $v \in V \backslash \{z, y_1, \ldots, y_k\}$ **do**

$\mathcal{L}' =$ *Shredders*$(z,\, v)$;
$\mathcal{L} = \mathcal{L}' \bigcup \mathcal{L}$.

(6) If $\{y_1, \ldots, y_k\} \in \mathcal{L}$, then remove $\{y_1, \ldots, y_k\}$ from $\mathcal{L}$ if $G \backslash \{z, y_1, \ldots, y_k\}$ has two components.

**End**

**Algorithm 2** *Shredders(r, v)*

**Input**: A $k$-connected graph $G$ and a node pair $r, v \in V(G)$.
**Output**: The family of $k$-shredders of $G$ that separate $r$ and $v$.

(1) Find $k$ openly disjoint $r \leftrightarrow v$ paths $P_1, \ldots, P_k$ in $G$. Let $Q$ denote the set of nodes of the paths $P_1, \ldots, P_k$.

(2) For each path $P_j$, $1 \leq j \leq k$, number the nodes $0, 1, 2, \ldots, (|P_j| - 1), \infty$, where $num(r) = 0$ and $num(v) = \infty$.

(3) Find the components $D_1, D_2, \ldots, D_c$ of $G \backslash Q$.

(4) Examine the components of $G \backslash Q$ to obtain a list of candidate shredders. Represent each candidate shredder $N(D_g) = \{u_1, u_2, \ldots, u_k\}$ by a $k$-tuple $\langle num(u_1), num(u_2), \ldots, num(u_k)\rangle$, where we assume that $u_i \in P_i$, $1 \leq i \leq k$.

(5) Repeatedly discard incomparable pairs of $k$-tuples, until no incomparable pair remains.

($k$-tuples $\langle num(u_1), num(u_2), \ldots, num(u_k)\rangle$ and $\langle num(w_1), num(w_2), \ldots, num(w_k)\rangle$ are incomparable if there exist $i$ and $j$, $1 \leq i, j \leq k$, such that $num(u_i) < num(w_i)$ and $num(u_j) > num(w_j)$.)

Lexicographically order the remaining $k$-tuples. Let the list in ascending order be $S_1, S_2, \ldots, S_f$.

(6) Examine all the bridges of $P_1 \cup P_2 \cup \ldots \cup P_k$, and discard every candidate shredder $S_g$, $1 \leq g \leq f$, such that $S_g$ is "straddled" by some bridge.

(A candidate shredder $S_g$ with $k$-tuple $\langle num(u_1), num(u_2), \ldots, num(u_k)\rangle$ is straddled by a bridge $B$ if there exist $i$ and $j$, $1 \leq i, j \leq k$, such that $B$ has attachments $w \in P_i$ and $x \in P_j$ such that $num(w) < num(u_i)$ and $num(x) > num(u_j)$.)

The remaining candidate shredders are all the $k$-shredders separating $r$ and $v$.

**End**

with a totally ordered list of candidate shredders $S_1, S_2, \ldots, S_f$ ($S_s$ occurs before $S_t$ iff the $k$-tuple for $S_s$ precedes the $k$-tuple for $S_t$).

Suppose that one of the remaining candidate shredders $S_g$, $1 \le g \le f$, with $k$-tuple $\langle u_1, u_2, \ldots, u_k \rangle$, is *not* a $k$-shredder separating $r$ and $v$. (Recall that a *bridge* of a subgraph $H$ means either an edge of $G$ that is not in $H$ but has both end nodes in $H$, or a component of $G \backslash V(H)$ together with all edges incident to the component. An *attachment* of a bridge $B$ is a node of $H$ that is incident to an edge in $B$.) Then there exists a bridge $B$ of $P_1 \cup P_2 \cup \ldots \cup P_k$ that "straddles" $S_g$, i.e., there exist $i$ and $j$, $1 \le i, j \le k$, such that $B$ has an attachment in the $r \to v$ path $P_i$ that strictly precedes $u_i$, and $B$ has an attachment in the $r \to v$ path $P_j$ that strictly follows $u_j$. The last step of *Shredders(r,v)* searches for all candidate shredders that are "straddled" by some bridge of $P_1 \cup P_2 \cup \ldots \cup P_k$, and discards all such candidate shredders. The remaining candidate shredders form the set of all $k$-shredders separating $r$ and $v$.

Observe that algorithm *All-k-shredders* finds a $k$-shredder $S$ that maximizes the number of components of $G \backslash S$, since it finds all the $k$-shredders of $G$.

**Theorem 3.3** *The algorithm correctly finds all k-shredders of G. Shredders(r,v) runs in* $O(\min(k, \sqrt{n})m)$ *time, and algorithm All-k-shredders runs in* $O((k^2 + n) \cdot \min(k, \sqrt{n})m) = O(knm + k^2\sqrt{n}m)$ *time.*

**Proof**: First consider the correctness of subroutine *Shredders(r,v)*. Clearly, the set of candidate shredders contains the set of $k$-shredders separating $r$ and $v$. If a candidate shredder $S$ is a $k$-shredder separating $r$ and $v$, then no bridge of $P_1 \cup P_2 \cup \ldots \cup P_k$ "straddles" $S$. Therefore, $S$ will not be discarded by the last two steps of *Shredders(r,v)*. On the other hand, if candidate shredder $S$ is *not* a $k$-shredder separating $r$ and $v$, then there must be a bridge $B$ of $P_1 \cup P_2 \cup \ldots \cup P_k$ that "straddles" $S$. This will be detected by either the last step or the second last step of *Shredders(r,v)*, and so $S$ will be discarded.

Next, consider the correctness of Algorithm *All-k-shredders*. Focus on an arbitrary $k$-shredder $S$ of the input graph $G$. Either $S$ separates some pair of nodes $y_i, y_j$, $1 \le i < j \le k$, or not. In the former case, $S$ will be found by Step (3) of Algorithm *All-k-shredders*. Otherwise, either there is one component of $G \backslash S$ that contains all nodes of $\{y_1, \ldots, y_k\} \backslash S$, i.e., $S$ separates $\{y_1, \ldots, y_k\} \backslash S$ from some node $v \in V \backslash (S \cup \{y_1, \ldots, y_k\})$, or, $S = \{y_1, \ldots, y_k\}$. In this case $S$ will be found by Step (5) of Algorithm *All-k-shredders*.

Algorithm *All-k-shredders* invokes *Shredders(r,v)* $O(k^2+n)$ times. We will show that *Shredders(r,v)* runs in $O(\min(k, \sqrt{n})m)$ time. The running time claimed in the theorem for Algorithm *All-k-shredders* will follow immediately. The rest of the proof shows how to implement *Shredders(r,v)* to run in $O(\min(k, \sqrt{n})m)$ time. Step 1 can be implemented in time $O(\min(k, \sqrt{n})m)$ [G 80], and Steps 2, 3, and 4 take linear time. Step 5 can be implemented by applying a radix sort to order the $k$-tuples (of the candidate shredders) according to the total order described above. Whenever the radix sort encounters a pair of incomparable $k$-tuples, it discards both. Since the number of candidate shredders is $\le n$, the running time for the radix sort is $O(kn)$.

Finally, consider Step 6 of *Shredders(r,v)*. Since the candidate shredders remaining at the start of Step 6 are totally ordered, we may view the collection of candidate shredders as a grid with $f$ rows (recall that $f$ is the number of remaining candidate shredders) and $k$ columns.

In this setting, Step 6 checks whether for every row $S$ and for every bridge $B$ of $P_1 \cup P_2 \cup \ldots \cup P_k$, all the attachments of $B$ are either "above" or "below" $S$.

Here is a more formal description of Step 6. Consider a bridge $B$ of $P_1 \cup P_2 \cup \ldots \cup P_k$. We say that a candidate shredder $S$ with $k$-tuple $\langle u_1, u_2, \ldots, u_k \rangle$ is *above* (respectively, *below*) $B$ if for *every* attachment $w$ of $B$, say, $w \in P_i$, $1 \le i \le k$, $u_i$ follows $w$ on the $r \to v$ path $P_i$ (respectively, $u_i$ precedes $w$ on the $r \to v$ path $P_i$). For each bridge $B$ of $P_1 \cup P_2 \cup \ldots \cup P_k$, we compute an open interval $(\ell_B, h_B)$, $0 \le \ell_B \le h_B \le f+1$, by examining the attachments of $B$: Take $\ell_B$ (respectively, $h_B$) to be the highest (respectively, lowest) index of a candidate shredder (from among $S_1, \ldots, S_f$) that is below (respectively, above) $B$, and if there is no candidate shredder below (respectively, above) $B$, then take $\ell_B = 0$ (respectively, $h_B = f+1$). The open intervals $(\ell_B, h_B)$ for all the bridges $B$ of $P_1 \cup P_2 \cup \ldots \cup P_k$ can be found in linear time. The computation of the $\ell_B$ values is as follows (the computation of the $h_B$ values is similar). Sequentially, for each $i = 1, \ldots, k$, we scan the nodes of the $r \rightarrow v$ path $P_i$, keeping track of the highest index of a candidate shredder seen so far, and whenever an attachment of a bridge $B$ is encountered, then we update $\ell_B$ (initially, $\ell_B = f$ for every bridge $B$). Once we have the open intervals $(\ell_B, h_B)$ for all the bridges $B$, we can delete all candidate shredders $S_g$, $1 \le g \le f$, such that there is a bridge $B$ with $\ell_B < g < h_B$. As the intervals may overlap, this process can be made more efficient by first computing the union of all the open intervals, and then deleting candidate shredders whose indices lie in the union. The union of a set of open intervals $\{(\ell_B, h_B)\}$ can be computed in linear time, by first sorting the tuples $(\ell_B, h_B)$ in lexicographic order, because there are at most $(n-1)$ tuples $(\ell_B, h_B)$ and the $\ell_B, h_B$ values are integers in the interval $[0, n]$. Thus, Step 6 can be implemented in linear time. $\square$

The time bound in the above theorem can be improved by precomputing a sparse certificate for $(k+1)$-connectivity, $G' = (V, E')$, $E' \subseteq E$, see [NI 92, CKT 93, FIN 93]. $G'$ has $|E'| \le (k+1)(n-1) = O(kn)$, and if $G$ is $(k+1)$-connected, then $G'$ is $(k+1)$-connected. Moreover, $G'$ can be computed in linear time, [NI 92]. In detail, we construct a legal ordering $v_1 \prec v_2 \prec \ldots \prec v_n$ of $V$, and retain an edge $v_i v_j$, $i < j$, in $E'$ iff $|\{v_\ell : v_\ell v_j \in E,\ \ell \le i\}| \ \le\ k+1$. Also, we need an extension of [CKT 93, Corollary 2.17] and [FIN 93, Corollary 2.3].

**Proposition 3.4** *(1) $S \subset V$ with $|S| \le k$ is a shredder (or separator) of $G$ iff $S$ is a shredder (or separator) of $G'$.*
*(2) If $G$ is $k$-connected, then $Q \subset V$ is a tight set of $G$ iff $Q$ is a tight set of $G'$.*

**Proof**: We prove part (1) for shredders. Suppose that $S$ is a shredder of $G'$ but not of $G$. Then there is an edge $vw$ in $E \backslash E'$ such that $v$ and $w$ are in different components of $G' \backslash S$. In the legal ordering for finding $G'$, let $v = v_i$ and $w = v_j$, $i < j$, and note that $|\{v_\ell : v_\ell v_j \in E,\ \ell \le i\}| \ >\ k+1$. But then the main lemma in [FIN 93] gives the desired contradiction, $|S \cap \{v_1, \ldots, v_{i-1}\}| \ \ge\ |\{v_\ell v_j \in E' : \ell = 1, \ldots, i-1\}| \ \ge\ k+1$. $\square$

This gives an improvement on the previous theorem: By precomputing a sparse certificate $G' = (V, E')$ for $(k+1)$-connectivity, and running the algorithm for finding $k$-shredders on $G'$, all the $k$-shredders of $G$ can be found in time $O((k^2 + n) \cdot (k^2 n))$.

**Theorem 3.5** *All the $k$-shredders of a $k$-connected graph can be found in $O(k^2 n^2 + k^3 n^{1.5})$ time. The same time bound suffices to find a $k$-separator $S$ that maximizes $\#c(G \backslash S)$.*

## 3.1 A dynamic algorithm for maintaining the set of all $k$-shredders over edge insertions/deletions

Given a $k$-connected graph $G$, $b(G)$ denotes the maximum number of components obtained by deleting a $k$-separator from $G$, where we take $b(G) = 2$ if $G$ has no $k$-separators, i.e., $b(G) = \max\{2,\ \{\#\mathrm{c}(G \backslash S) : S \subset V, |S| = k\}\}$. In this subsection, we sketch an algorithm for maintaining $b(G)$ over a sequence of edge insertions/deletions, assuming that $G$ stays $k$-connected throughout. At the start, we run our algorithm for finding all $k$-shredders of $G$ (if there are no $k$-shredders, then $b(G) = 2$). Next, using the lexicographically sorted list $\mathcal{L}$ of all $k$-shredders, we insert each $S \in \mathcal{L}$ into a (max) heap, see [CLR 90], using the value $\#\mathrm{c}(G \backslash S)$ as the key. (Our heap is organized by maximum key values, and each insertion or deletion takes $O(\log|\mathcal{L}|) = O(\log n)$ time.) Whenever an edge $xy$ is added to (or deleted from) $G$, we update the list $\mathcal{L}$ and the heap as follows. First, we run our algorithm *Shredders*($x$,$y$) on the graph $G\backslash xy$ (here, $G$ is the graph after the edge update), to find all the $k$-shredders separating $x$ and $y$. This takes time $O(|E| + \min(k, \sqrt{n})kn)$, and returns a set of at most $n$ $k$-shredders $\mathcal{L}_{xy}$. For each shredder $S$ in $\mathcal{L}_{xy}$, we search for $S$ in our list $\mathcal{L}$, and if successful, we also obtain a pointer to $S$ in the heap. If $S \in \mathcal{L}$, then we decrement (or increment) the key of $S$ by one, since inserting (or deleting) edge $xy$ decreases (or increases) the number of components of $G\backslash S$ by one. If $\#\mathrm{c}(G\backslash S)$ becomes two (after an edge insertion), then we delete $S$ from $\mathcal{L}$ as well as from the heap. If we do not find $S$ in $\mathcal{L}$ (after an edge deletion), then we insert $S$ in $\mathcal{L}$ as well as in the heap. Thus the overall time per edge insertion or edge deletion is $O(|E| + (\min(k, \sqrt{n}) + \log n)kn)$, and the time per query of $b(G)$ is $O(1)$.

**Theorem 3.6** *Given a $k$-connected graph $G$, $b(G)$ and the set of all the $k$-shredders can be maintained over a sequence of edge insertions/deletions such that the time per edge update is $O(|E| + (\min(k, \sqrt{n}) + \log n)kn)$, the time per query of $b(G)$ is $O(1)$, and the preprocessing time is $O((k + \log n)kn^2 + k^3 n^{1.5})$.*

## 4 Counting the number of $k$-separators and $k$-shredders

Our first result in this section settles the open question of counting the number of $k$-separators in a $k$-connected graph: this problem is #P-complete. Our remaining results focus on $k$-shredders in a $k$-connected graph. The algorithm in Section 3 and Proposition 3.1 straightaway give a bound of $O((k^2 + n)n)$ on the number of $k$-shredders in a $k$-connected graph. We derive tighter bounds for some special cases. Lemma 4.3 provides the key tool for handling meshing $k$-shredders and $k$-separators. Recall that a separator $T$ is said to mesh with a separator $S$ if $T$ has nodes from at least two components of $G\backslash S$.

**Theorem 4.1** *The problem of counting the number of $k$-separators in a $k$-connected graph is #P-complete.*

**Proof**: Clearly, the problem is in #P since minimum-cardinality separators can be recognized in polynomial time. We give a reduction to our problem from the problem of counting the number of minimum node covers in a bipartite graph $H$ such that $H$ has a perfect matching. The latter problem is well known to be #P-complete, see [PB 83, Problem 4, page 783] (note that the bipartite graph there has a perfect matching). Let the bipartition of $V(H)$ be given by $P, Q$ (so $V(H) = P \cup Q$), and let $k = |P| = |Q|$. Since $H$ has a perfect matching (of cardinality $k$), it is clear that the minimum cardinality of a node cover is $k$. We construct a $k$-connected graph $G$ by adding all possible edges between nodes of $P$, and similarly adding all possible edges between nodes of $Q$, i.e., we set up a $k$-clique on each of $P$ and $Q$. The proof is completed using two claims.
Claim 1: $G$ is $k$-connected.
Let $S \subset V(G)$ have cardinality $< k$. Consider $G\backslash S$. The nodes in $P\backslash S$ induce a connected subgraph (by the $k$-clique on $P$), and similarly the nodes in $Q\backslash S$ induce a connected subgraph. $G$ must have at least one edge between $P\backslash S$ and $Q\backslash S$, otherwise every edge of $H$ is covered by $S$, and this is not possible since every node cover of $H$ has cardinality $\geq k$. Then $G\backslash S$ is connected. Since $G$ has no separator of cardinality $< k$, it is $k$-connected.
Claim 2: $S \subseteq P \cup Q$ with $S \neq P$, $S \neq Q$ is a $k$-separator of $G$ iff $S$ is a minimum node cover of $H$. This follows directly from the proof of Claim 1. □

**Remarks**: (1) The above reduction is not parsimonious because $P$ and $Q$ are minimum node covers of $H$ but are not minimum separators of $G$. A parsimonious reduction is obtained by modifying the construction of $G$: we add two new nodes, one adjacent to all nodes in $P$ and the other adjacent to all nodes in $Q$.

(2) The number of $k$-separators in a $k$-connected graph may be as high as $2^k \binom{\lfloor n/k \rfloor}{2}$,

Before presenting Lemma 4.3, we give a few examples to convince the reader that simpler versions of the lemma are not valid. The next result focuses on 2-connected/3-connected graphs.

**Proposition 4.2** *(1) The 2-shredders of a 2-connected graph form a nonmeshing family. In fact, no 2-separator meshes with a 2-shredder.*
*(2) Except for the complete bipartite graph $K_{3,3}$, in every 3-connected graph, the 3-shredders form a nonmeshing family. In a 3-connected graph $G = (V, E)$, $G \neq K_{3,3}$, there may be 3-separators meshing with a 3-shredder, but the removal of each such 3-separator results in a single node and another component.*

**Proof**: Part (1) follows by Lemma 2.1, since every 2-separator meshing with a 2-shredder has cardinality $\geq 3$. To see part (2), let $S$ be a 3-shredder and let $T$ be a 3-separator meshing with $S$. By Lemma 2.1, $G \backslash S$ has exactly three components $D_1, D_2, D_3$, and $T$ has exactly one node in each of these components. For $|V(G)| \leq 6$, it is clear that $K_{3,3}$ is the unique graph having a 3-shredder and a 3-separator meshing with the 3-shredder. If $|V(G)| > 6$, then $V \backslash (S \cup T) \neq \emptyset$. For each node $v \in V \backslash (S \cup T)$, say, $v$ is in $D_1$, the induced subgraph $G[V(D_1) \cup S]$ has three openly disjoint paths from $v$ to $S$ (these paths have only node $v$ in common), so at least two of these paths survive in $G \backslash T$. Hence, all nodes of $V \backslash (S \cup T)$ are in one component of $G \backslash T$, and also this component has at least two nodes of $S$. Part (2) follows. $\square$

For higher $k$ and $n < 2k$, there may be $\Theta(k)$ $k$-shredders such that every pair is meshing. Let $k = 3k'$, where $k'$ is a positive integer. Take $G$ to be the graph obtained from the clique $K_{k+3}$ by removing the $3(k'+1)$ edges of $(k'+1)$ node-disjoint triangles ($K_3$'s) $T_1, \ldots, T_{k'+1}$. It is easily checked that $G$ is $k$-connected, each of the $k$-sets $V \backslash T_i$, $1 \leq i \leq k'+1$, is a $k$-shredder, and for $1 \leq i < j \leq k'+1$, $V \backslash T_i$ and $V \backslash T_j$ mesh. Finally, consider some meshing $k$-shredders on graphs obtained from the complete bipartite graph $K_{k,k}$, $k \geq 5$, as follows. Let the node sets of the bipartition be $S = \{s_1, \ldots, s_k\}$ and $T = \{t_1, \ldots, t_k\}$. Take two new nodes $v$ and $w$, and join $v$ to $K_{k,k}$ by the edges $vs_1, vs_2, vt_3, \ldots, vt_k$, and similarly join $w$ to $K_{k,k}$ by the edges $ws_3, ws_4, wt_3, \ldots, wt_k$. The resulting graph $G$ is easily seen to be $k$-connected. Now, $S$ is a $k$-shredder since $G \backslash S$ has components $\{t_1\}, \{t_2\}, \{t_3, \ldots, t_k, v, w\}$, and $T$ is a $k$-shredder meshing with $S$, where the components of $G \backslash T$ are $\{s_1, s_2, v\}, \{s_3, s_4, w\}, \{s_5\}, \ldots, \{s_k\}$. In the above example, $|V(G)| = 2k + 2$, but this construction easily extends to any number of nodes $\geq 2k + 2$.

**Lemma 4.3** *Let $G$ be a $k$-connected graph, $k \geq 1$, and let $S$ be a $k$-shredder of $G$. If there is a $k$-separator $T$ that meshes with $S$, then there is a component $Q$ of either $G \backslash T$ or of $G \backslash S$ such that $Q$ contains every node of $V \backslash (S \cup T)$.*

**Proof**: First, note that the lemma holds trivially if $V \backslash (S \cup T) = \emptyset$. Now assume that $V \backslash (S \cup T) \neq \emptyset$. Let $D_1, D_2, \ldots, D_h$ denote the components of $G \backslash S$, where $h \geq 3$. W.l.o.g. suppose that $D_h$ is a component of $G \backslash S$ having the maximum number of nodes from $V \backslash T$, and let $z$ be any node in $V(D_h) \backslash T$. W.l.o.g. suppose that the components $D_1$ and $D_2$ of $G \backslash S$ each have one or more nodes of $T$.

**Claim**: Every node $v \in V \backslash (S \cup T)$ in one of the components $D_1, \ldots, D_{h-1}$ of $G \backslash S$ has a path to $z$ in $G \backslash T$.
To prove the claim, consider $v \in V(D_i) \backslash T$, $1 \le i \le h-1$. There are $k$ openly disjoint $v \leftrightarrow z$ paths in $G$, since $G$ is $k$-connected. It can be seen that each of these paths is contained in the subgraph of $G$ induced by $V(D_i) \cup S \cup V(D_h)$, i.e., no path uses a node of $(V(D_1) \cup \ldots \cup V(D_{h-1})) \backslash V(D_i)$. Since $T$ has at least one node in each of $V(D_1)$ and $V(D_2)$, it has $< k$ nodes in $V(D_i) \cup S \cup V(D_h)$. Hence, at least one of the $k$ openly disjoint $v \leftrightarrow z$ paths survives in $G \backslash T$. This proves the claim.

If $G \backslash T$ has a component that contains $V(D_h) \backslash T$, then the lemma follows since every node in $V \backslash (S \cup T)$ has a path to $V(D_h) \backslash T$ in $G \backslash T$ (by the claim). Otherwise (i.e., if $V(D_h) \backslash T$ is disconnected in $G \backslash T$), then $T$ contains all nodes of the other components $D_1, \ldots, D_{h-1}$ of $G \backslash S$. To see this, suppose that there is a node $v \in (V(D_1) \cup \ldots \cup V(D_{h-1})) \backslash T$. By the claim, every node $z \in V(D_h) \backslash T$ has a path to $v$ in $G \backslash T$, hence $V(D_h) \backslash T$ is contained in a component of $G \backslash T$. The lemma follows by taking $Q = D_h$, since $T \supset V(D_1) \cup \ldots \cup V(D_{h-1})$. □

We can obtain another proof of Proposition 3.1, namely, the family of $k$-shredders separating a given pair of nodes $v, z$ in a $k$-connected graph is nonmeshing; hence, the family has cardinality $O(n)$.

**Proposition 4.4** *Let $G$ be a $k$-connected graph, and let $v, z$ be nodes of $G$. Let $S$ and $T$ be two $k$-shredders that separate $v$ and $z$. Then $S$ and $T$ are nonmeshing.*

**Proof**: Clearly, both $v$ and $z$ are in $V \backslash (S \cup T)$. By way of contradiction, suppose that $S$ and $T$ mesh. Then by Lemma 4.3, there is a component either of $G \backslash S$ or of $G \backslash T$ that contains both $v$ and $z$. Contradiction. □

# 5 Augmenting node connectivity by one

Our algorithm for augmenting the node connectivity of a graph by one is a variant of Jordán's algorithm [J 95] but is significantly faster. First, we describe a lower bound on the number of new edges required to increase the node connectivity from $k$ to $(k+1)$. Several recent algorithms for edge/node connectivity augmentation problems are based on splitting-off theorems, see the survey paper [F 94]. In particular, Jordán's algorithm is based on a key theorem for splitting off edges while preserving node connectivity. We state and prove a simpler version of this theorem in Section 5.2 (Theorem 5.1). In Section 5.3, we present the augmentation algorithm, prove it correct, and analyze its running time.

Readers interested in algorithmic aspects may prefer to skip Section 5.2 after reading the overview of the augmentation algorithm given there, and to refer back when required to Theorem 5.1 and Lemmas 5.6–5.8.

See Figure 2 for an illustration of the algorithm.

## 5.1 A lower bound on the number of augmented edges

Let $G$ be a $k$-connected graph. Recall from Section 2 that a *tight set* is a node set $Q$ such that $|N(Q)| = k$ and $|V \backslash Q| \ge (k+1)$. The maximum number of pairwise disjoint tight sets in $G$ is denoted by $t(G)$, i.e., $t(G)$ is the maximum integer $\ell \ge 0$ such that $D_1, \ldots, D_\ell$ are tight sets and $D_i \cap D_j = \emptyset, 1 \le i < j \le \ell$. Recall from Section 3.1 that $b(G)$ denotes the maximum number of components obtained by deleting a $k$-separator (assuming there is one) from $G$.

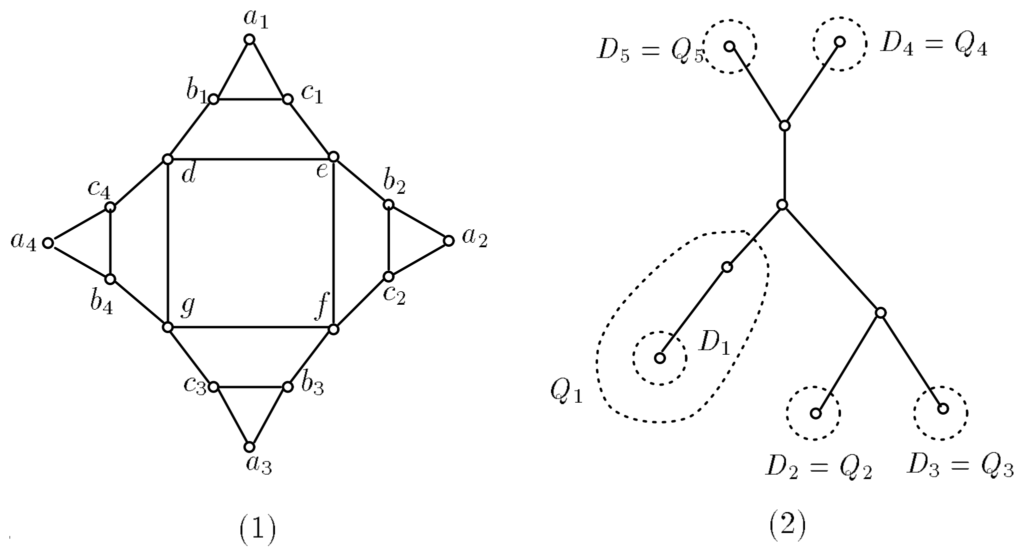


**Figure 2:** Illustrating algorithm augment node connectivity.
(1) $G$ is 2-connected, $b(G) = 2$, and $t(G) = 4$. The leafs are $D_i = \{a_i\}$, $1 \le i \le 4$, and the superleafs are $Q_i = \{a_i, b_i, c_i\}$, $1 \le i \le 4$. Suppose the algorithm (Lemma 5.6) chooses $Q_i = Q_1$ (so $N(Q_i) = \{d, e\}$ is not a shredder) and takes $Q_j = Q_2$, $Q_p = Q_4$. Adding edge $xy = a_1a_2$ fails ($G + a_1a_2$ has a new leaf $Q_i \cup Q_j \cup \{e\}$), similarly, adding edge $xz = a_1a_4$ fails. Adding edge $yz = a_2a_4$ is guaranteed to succeed.
(2) $G$ is a tree, $b(G) = 3$ and $t(G) = 5$. Suppose the algorithm (Lemma 5.7) chooses $Q_i = Q_1$ (so $N(Q_i)$ is a shredder) and takes $Q_j = Q_2$. Adding the edge between the degree-one nodes in $Q_i$ and $Q_j$ succeeds.

EXAMPLES: Suppose that $G$ is a tree. Then $t(G)$ is the number of degree-one nodes, and $b(G)$ is the maximum degree of a node. If $G$ is the complete bipartite graph $K_{k,k}$, then $t(G) = |V(G)| = 2k$ and $b(G) = k$. Lastly, for the graph $G$ in Figure 2(1), $t(G) = 4$ and $b(G) = 2$.

Consider our problem of adding some edges to augment the connectivity of $G$ from $k$ to $k+1$. Let $G'$ be the augmented graph. An obvious lower bound on the minimum number of edges required is $\max(b(G) - 1, \lceil t(G)/2 \rceil)$. To see this, first consider a $k$-separator $S$ such that $G \backslash S$ has $b(G)$ components, and note that we must add $\ge b(G) - 1$ edges to ensure that $G' \backslash S$ is connected. Secondly, for every tight set $D$, $G'$ must have an edge with one end in $D$ and the other in $V \backslash (D \cup N(D))$. Since $G$ has $t(G)$ pairwise disjoint tight sets, we must add $\ge \lceil t(G)/2 \rceil$ edges. Unfortunately, the lower bound is not tight and there may be a slack of $(k-2)$, as shown by the following example due to Jordán [J 95]: consider the complete bipartite graph $K_{k,k}$, and note that the minimum number of new edges required is $2k-2$, but our lower bound is $k$, since $b(K_{k,k}) = k$ and $t(K_{k,k}) = 2k$. Hence, an algorithm based on the above lower bound, such as the algorithm in this section, will not find the optimal augmentation on *all* graphs.

## 5.2 A splitting-off theorem for node connectivity

Let $s$ be a distinguished node of a graph. *Splitting off* a pair of edges $vs$ and $sw$ incident to $s$ means removing edges $vs$ and $sw$, and adding the edge $vw$ (if $vw$ is already present, then no edge is added). The algorithm for augmenting node connectivity is based on a subroutine for finding and splitting off a pair of edges incident to $s$ such that the node connectivity of the resulting graph does not decrease. Here is an overview of the augmentation algorithm that skips some important points:

> Let $G$ be a $k$-connected graph that is not $(k+1)$-connected. We first construct a $(k+1)$-connected graph $\widetilde{G}$ by adding a new node $s$ and new edges between $s$ and each node $v \in V(G)$. ($\widetilde{G}$ is $(k+1)$-connected because every separator of $\widetilde{G}$ contains the node $s$ as well as a separator of $G$.) Then for each node $v \in V(G)$, in an arbitrary order, we remove the edge $sv$ from $\widetilde{G}$ if doing so preserves the $(k+1)$-connectivity of the resulting graph (also denoted $\widetilde{G}$). For each tight set $D$ of $G$, note that $\widetilde{G}$ has an edge between some node of $D$ and $s$ (otherwise, $N_G(D)$ is a $k$-separator of $\widetilde{G}$). We attempt to pair up the edges incident to $s$ and split off all these edge pairs, while preserving $(k+1)$-connectivity. If we succeed, then the resulting graph $G'$ (without node $s$) will be a $(k+1)$-connected augmentation of $G$.

The earliest splitting-off theorem is due to Lovász [Lo 74] and concerns the edge connectivity of multigraphs: If $s$ is a node of even degree in a multigraph $\widetilde{G}$, and there are at least $k \geq 2$ edge-disjoint paths between any two nodes of $V(\widetilde{G})\backslash s$, then all edges incident to $s$ can be paired up and split off such that the resulting multigraph (without node $s$) has at least $k$ edge-disjoint paths between any two nodes. Mader [Ma 78] gave a deep generalization. Mader [Ma 82] also gave a splitting-off theorem for the edge connectivity of directed multigraphs. Other expositions of these three results may be found in [F 92a], [F 92b] and [F 93, FJ 95a], respectively. To the best of our knowledge, the earliest splitting-off theorem for node connectivity is due to Bienstock, Brickell and Monma [BBM 90, Theorem 3]. A different proof of a variant of this theorem is given by Jordán [J 95, Theorem 3.1]. Jordán gave a splitting-off theorem for the node connectivity of directed graphs [J 93, Theorem 2]. Another proof appears in [FJ 95b].

Splitting-off theorems for node connectivity hold only under appropriate conditions. Here are three examples (violating the appropriate conditions) such that splitting off any (or all) edge pair(s) incident to $s$ decreases the connectivity. These examples are due to Bienstock et al [BBM 90, p. 324], and to Hsu [H 92].

Example (1): Start with the complete bipartite graph $G = K_{3,3}$, and obtain the 4-connected graph $\widetilde{G}$ by adding a new node $s$ and all the edges $\{sv : v \in V(G)\}$. Splitting off any edge pair incident to $s$ results in a 3-connected graph. This example generalizes to all $K_{k,k}, k \geq 3$.

Example (2): For another example, start with the complete bipartite graph $G = K_{1,p}$, $p \geq 4$, and obtain the 2-connected graph $\widetilde{G}$ by adding a new node $s$ and $p$ new edges $sv$ where $v \in V(G)$ is in the larger part of the bipartition of $K_{1,p}$. Splitting off any edge pair incident to $s$ results in a 1-connected graph. This example generalizes to all $K_{k,p}, k \geq 1, p \geq k+3$. Moreover, we can replace one or more nodes $v$ in the larger part of the bipartition of $K_{k,p}$ by $(k+1)$-connected graphs $H_v$ (or $(k+1)$-cliques $H_v$) and replace the $k$ edges incident to $v$ by $k$ edges incident to distinct nodes of $H_v$.

Example (3): For the last example, take three copies of the complete graph $K_4$ on the node sets $\{a_i, b_i, c_i, d_i\}, 1 \leq i \leq 3$. Identify the nodes $b_1$ and $a_2$, i.e., replace $b_1$ and $a_2$ by a new node that is incident to all edges incident to $b_1$ or $a_2$. Similarly, identify the nodes $b_2$ and $a_3$, and the nodes $b_3$ and $a_1$. Also, add a new node $f$ and the edges $fc_i, 1 \leq i \leq 3$. The resulting graph $G$ is 3-connected. Obtain the 4-connected graph $\widetilde{G}$ from $G$ by adding a new node $s$ and the edges $sf$ and $sd_i, 1 \leq i \leq 3$. For every pairing of the edges incident to $s$, splitting off all the edge pairs (and ignoring the node $s$) results in a 3-connected graph. This example generalizes to all odd $k \geq 3$: take three copies of the complete graph $K_{k+1}$, "join" them as above, then add a copy of $K_{k-1}$, and for each copy of $K_{k+1}$ add the edge set of a matching between the degree-$k$ nodes and the copy of $K_{k-1}$. Take $s$ to be one of the nodes of the copy of $K_{k-1}$.

Our version of the splitting-off theorem is weaker than the splitting-off theorem in [J 95, Theorem 3.1]: we add the condition $\deg_{\widetilde{G}}(s) \geq 2k$. This allows us to simplify the proof. For the main problem of augmenting the connectivity from $k$ to $(k+1)$, even our weaker theorem implies the same slack of $(k-2)$ between the number of new edges and the lower bound. Also, our theorem omits the condition $|V(\widetilde{G})| \geq 2(k+1)$, consequently it has to allow the possibility that $\widetilde{G}\backslash s = G$ is the complete bipartite graph $K_{k,k}$. (The only use of this condition in [J 95, Theorem 3.1] is to show that $\widetilde{G}\backslash s \neq K_{k,k}$.) The difference between our version of the splitting-off theorem and Bienstock et al's splitting-off theorem [BBM 90, Theorem 3] is that a new condition (see (3) in Theorem 5.1) has been added. This guarantees that the connectivity can be preserved by a single splitting-off operation, whereas in [BBM 90, Theorem 3] one or two splitting-off operations are required to preserve the connectivity. Our proof hinges on the notions of superleafs and the maximal tight sets $W_{ij}$ (defined below), and follows immediately from Lemmas 5.6–5.8. Recall that an edge $vw$ of a graph $H$ is called critical if the node connectivity of $H\backslash vw$ is less than that of $H$.

**Theorem 5.1** *Let $\widetilde{G}$ be a $(k+1)$-node connected graph ($k \geq 1$), and let $s$ be a node of $\widetilde{G}$. Suppose that $s$ is incident to $t \geq \max(2k, k+3)$ edges each of which is critical. Then either*

*(1) there is a pair of edges incident to $s$ such that splitting off this pair results in a $(k+1)$-node connected graph, or*

*(2) $\widetilde{G}\backslash s = K_{k,k}$, or*

*(3) there is a $(k+1)$-separator $X$ of $\widetilde{G}$ such that $s \in X$ and $\widetilde{G}\backslash X$ has $\deg_{\widetilde{G}}(s)$ components.*

The necessity of three of the conditions in the theorem, namely, $t \geq k+3$, (2) and (3) can be seen from Examples (3), (1) and (2), respectively.

Suppose that splitting off an edge pair $v_i s, sv_j$ in a $(k+1)$-connected graph $\widetilde{G}$ results in a graph $\widetilde{G}_{ij}$ that is *not* $(k+1)$-connected. Then $\widetilde{G}_{ij}$ has a $k$-separator $X$.

**Fact 5.2** *Let $X$ be a $k$-separator of $\widetilde{G}_{ij}$. Then (1) $s \notin X$, (2) either $v_i \notin X$ or $v_j \notin X$, and (3) in $\widetilde{G}_{ij}\backslash X$ the component containing $s$ contains neither $v_i$ nor $v_j$.*

**Proof**: First, note that $\widetilde{G}_{ij}\backslash s = (\widetilde{G}\backslash s) + (v_i v_j)$, since the edges $v_i s$ and $sv_j$ "vanish" when $s$ is removed. Hence, a $k$-separator $X$ of $\widetilde{G}_{ij}$ with $s \in X$ is also a $k$-separator of $\widetilde{G}$. Since $\widetilde{G}$ has no $k$-separator, part (1) follows. Similarly, for part (2), $\widetilde{G}_{ij}\backslash\{v_i, v_j\} = \widetilde{G}\backslash\{v_i, v_j\}$, so a $k$-separator $X$ of $\widetilde{G}_{ij}$ with $v_i \in X, v_j \in X$ is also a $k$-separator of $\widetilde{G}$. See Figure 3 for an illustration.

To see that $s$ and $\{v_i, v_j\}\backslash X$ are contained in different components of $\widetilde{G}_{ij}\backslash X$, first suppose that neither $v_i$ nor $v_j$ is in $X$. Then $v_i$ and $v_j$ are in the same component of $\widetilde{G}_{ij}\backslash X$, since there is a new edge $v_i v_j$. If $s$ is also in this component, then $X$ is a $k$-separator of $\widetilde{G}$, contradicting the $(k+1)$-connectivity of $\widetilde{G}$. Similarly, if $v_i \in X$ ($v_j \in X$), then $s$ and $v_j$ ($v_i$) must be in different components of $\widetilde{G}_{ij}\backslash X$. □

$\widetilde{G}_{ij}$ has a tight set (w.r.t. $k$-connectivity) that contains $v_i$ or $v_j$ (or both) but not $s$, by the previous fact. Let $W_{ij}$ denote such a tight set that is (inclusionwise) maximal, i.e., no proper superset of $W_{ij}$ is tight. (The maximality of $W_{ij}$ will be exploited in Fact 5.5.) Clearly, $W_{ij}$ contains no neighbor of $s$ in $\widetilde{G}$ other than $v_i$ and $v_j$ (i.e., $W_{ij}$ is disjoint from $N_{\widetilde{G}}(s)\backslash\{v_i, v_j\}$), otherwise the $k$-separator $N_{\widetilde{G}_{ij}}(W_{ij})$ of $\widetilde{G}_{ij}$ contains $s$, contradicting the previous fact. There are three cases:

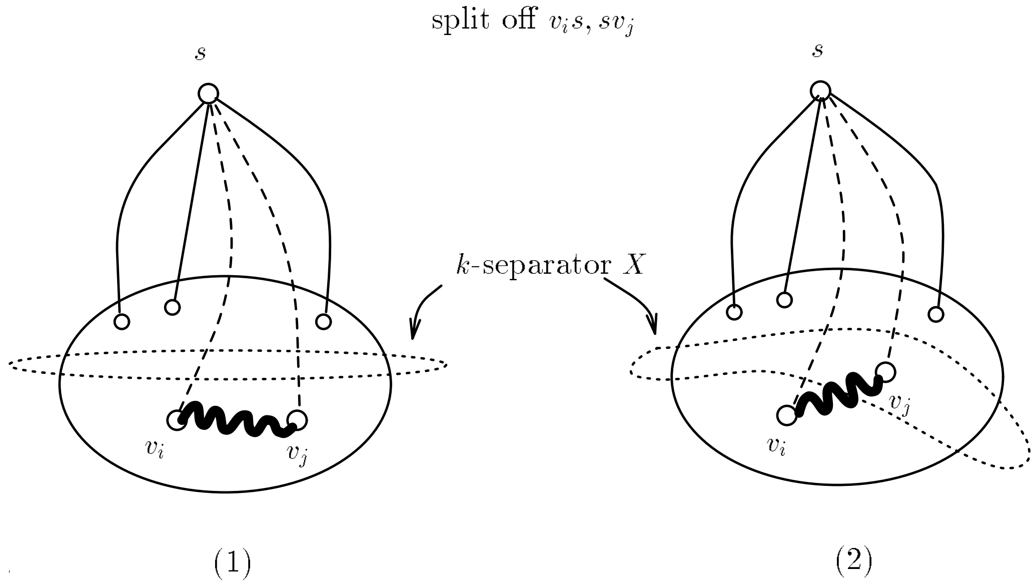


**Figure 3:** Illustrating the definition of $W_{ij}$. (1) Case (i). (2) Case (ii).

(i) $W_{ij}$ contains both $v_i$ and $v_j$;

(ii) $W_{ij}$ contains $v_i$ but not $v_j$, and so $v_j \in N_{\widetilde{G}_{ij}}(W_{ij})$;

(iii) $W_{ij}$ contains $v_j$ but not $v_i$, and so $v_i \in N_{\widetilde{G}_{ij}}(W_{ij})$.

(Possibly, there are two different maximal tight sets, one satisfying (ii) and the other satisfying (iii), but then we take $W_{ij}$ to be either of these two sets.) Case (i) is crucial for our proof of the splitting-off theorem; we will avoid cases (ii) and (iii) altogether. (These three cases correspond to cases ($\alpha$), ($\beta$) and ($\gamma$) in [J 95, p. 13].)

Let $G$ be a $k$-connected graph. We call an (inclusionwise) minimal tight set of $G$ a *leaf*, and denote the leafs by $D_i$, $i = 1, 2, \ldots$. For example: (1) if $G$ is a tree, then every degree-one node is a leaf, (2) if $G = K_{k,k}$, then every node is a leaf (in both graphs, there are no other leafs), and (3) the graph in Figure 2(1) has four leafs, $\{a_i\}$, $1 \leq i \leq 4$. In general, leafs need not be disjoint. (Example: Take a complete graph $K_5$ having nodes $a, b, c, d, e$, and add two more nodes $f$ and $g$, where $f$ has edges to $a, b, g$ and $g$ has edges to $c, d, f$. The resulting graph is 3-connected. Consider the 3-separators that isolate $f$ and $g$, $\{a, b, g\}$ and $\{c, d, f\}$, and note that the leafs $\{a, b, e\}$ and $\{c, d, e\}$ intersect.) The next result is from [J 95]. Recall from Section 5.1 that $t(G)$ denotes the maximum number of pairwise disjoint tight sets in $G$.

**Fact 5.3 (Lemma 2.1 [J 95])** *If a $k$-connected graph $H$ has $t(H) \geq k+1$, then all the leafs are pairwise disjoint, and the number of leafs is $t(H)$.*

**Fact 5.4** *Let $\widetilde{G}$ be a $(k+1)$-connected graph, and let $s$ be a node of $\widetilde{G}$. Every tight set (w.r.t. $k$-connectivity) of $G = \widetilde{G} \backslash s$ contains a neighbor of $s$ in $\widetilde{G}$. If $\widetilde{G}$ has $\ell \geq (k+2)$ critical edges incident to $s$, then $G$ has $\geq \ell$ leafs, all the leafs are pairwise disjoint, and $t(G) \geq \ell$.*

An (inclusionwise) maximal tight set that contains exactly one leaf is called a *superleaf*, and is denoted by $Q_i$, $i = 1, 2, \ldots$. (This definition allows a superleaf to have a nonempty intersection with several leafs. A superleaf may be a leaf.) For example, if $G$ is a tree, a superleaf is a maximal path starting at a degree-one node such that all other nodes are degree-two nodes. For another example, the graph $G$ in Figure 2(1) has four superleafs $\{a_i, b_i, c_i\}$, $1 \le i \le 4$. The notion of superleafs is used in the proofs of all the splitting-off theorems for node connectivity cited above. The next result is essentially from [J 95] (see Claim I in Theorem 3.1) and summarizes some useful properties of superleafs.

**Fact 5.5** *Let $G$ be a $k$-connected graph with $t = t(G) \ge k+3$. Let $D_1, \ldots, D_t$ be the (pairwise disjoint) leafs of $G$. Then:*

*(0) For every leaf, as well as every superleaf, the induced subgraph is connected.*

*(1) For every leaf $D_i$, $1 \le i \le t$, there is a unique superleaf $Q_i$ containing it.*

*(2) All the superleafs are pairwise disjoint. Hence, except for the leaf contained in it, a superleaf is disjoint from all other leafs.*

*Let the $(k+1)$-connected graph $\widetilde{G}$ be obtained from $G$ by adding a new node $s$, and a new edge between $s$ and one node $v_i$ in $D_i$, for each $i = 1, \ldots, t$. Suppose that (in $\widetilde{G}$) splitting off the edge pair $v_i s, s v_j$, $1 \le i < j \le t$, decreases the connectivity. Let $W_{ij}$ be the node set defined above. Then:*

*(3) $W_{ij}$ is disjoint from all superleafs $Q_\ell$, $1 \le \ell \le t$, $i \ne \ell \ne j$.*

*(4) Either $W_{ij}$ contains both the superleafs $Q_i$ and $Q_j$ (case (i)), or $W_{ij} = Q_i$ and $D_j \cap N(Q_i) \ne \emptyset$ (case (ii)), or $W_{ij} = Q_j$ and $D_i \cap N(Q_j) \ne \emptyset$ (case (iii)).*

*(5) If $Q_j$ is disjoint from $N(Q_i)$ (this implies that $Q_i$ is disjoint from $N(Q_j)$), then $W_{ij}$ contains both the superleafs $Q_i$ and $Q_j$ (case (i) for $W_{ij}$).*

Let $G$ be a $k$-connected graph with $t = t(G) \ge k+3$, and let $D_1, \ldots, D_t$ be the leafs of $G$. A node pair $\{v_i, v_j\}$ of $G$ is called a *saturating* pair if adding the edge $v_i v_j$ decreases the number of leafs by two, i.e., if $t(G + v_i v_j) = t(G) - 2$. Alternatively, $\{v_i, v_j\}$ is a saturating pair if there are leafs, say, $D_i$ and $D_j$, $1 \le i \ne j \le t$, with $v_i \in D_i$ and $v_j \in D_j$ such that splitting off the edge pair $v_i s, s v_j$ in the $(k+1)$-connected graph $\widetilde{G}$ preserves the connectivity, where $\widetilde{G}$ is obtained from $G$ by choosing an arbitrary node $v_\ell \in D_\ell$, for each $\ell$, $1 \le i \ne \ell \ne j \le t$, adding a new node $s$, and adding the new edges $s v_\ell$, $1 \le \ell \le t$. If $\{v_i, v_j\}$ is *not* saturating, then $G$ has a tight set $W_{ij}$ containing $v_i$ or $v_j$ (or both) and satisfying case (i), (ii) or (iii) above.

The proof of Lemma 5.6 follows the proof of Step 3.4, Theorem 3 of [BBM 90] and the proof of Claim II(a)$\rightarrow$(b), Theorem 3.1 of [J 95]. For the sake of completeness, a proof is included in the appendix.

**Lemma 5.6** *Let $G = (V, E)$ be a $k$-connected graph ($k \ge 1$) with $t(G) \ge k+3$. Let $Q_i$, $Q_j$ and $Q_p$ be three distinct superleafs such that $N(Q_i)$ is disjoint from each of $Q_j$ and $Q_p$. Let $D_i$, $D_j$ and $D_p$ be the leafs contained in $Q_i$, $Q_j$ and $Q_p$ respectively. Then for every three nodes $x \in D_i$, $y \in D_j$ and $z \in D_p$, either one of the node pairs $\{x, y\}$, $\{x, z\}$ or $\{y, z\}$ is saturating, or $N(Q_i) = N(Q_j) = N(Q_p)$, i.e., $N(Q_i)$ is a $k$-shredder.*

**Lemma 5.7** *Let $G = (V, E)$ be a $k$-connected graph ($k \ge 1$) with $t(G) \ge \max(2k, k+3)$. Let $Q_i \subset V$ be an arbitrary superleaf such that $G \backslash N(Q_i)$ has at least three components (so $N(Q_i)$ is a $k$-shredder).*

*(1) If one of the components of $G \backslash N(Q_i)$ contains two or more leafs, then that component contains*

*a superleaf $Q_j$, $i \neq j$.*
*(2) If a component of $G \backslash N(Q_i)$ contains a superleaf $Q_j$ as well as another (disjoint) leaf $D_p$, then for every node $x \in D_i$, and for every node $y \in D_j$, the node pair $\{x, y\}$ is saturating, where $D_i$ is the leaf contained in $Q_i$ and $D_j$ is the leaf contained in $Q_j$.*

**Proof:** Since $t(G) \geq k+3$, $G$ has $t(G)$ (pairwise disjoint) leafs and $t(G)$ (pairwise disjoint) superleafs (Facts 5.3–5.5). Let the component $C$ of $G \backslash N(Q_i)$ contain leafs $D_h$ and $D_g$, $h \neq g$. Consider the superleaf $Q_h$, $Q_h \supseteq D_h$, and let $X = N(Q_h)$. If $Q_h \cap N(Q_i) = \emptyset$, then the proof of part (1) is done since $Q_h \subseteq C$. Otherwise, if $Q_h \cap N(Q_i)$ is nonempty (i.e., there is an edge with one end in $Q_i$ and the other end in $Q_h$), then $X$ meshes with $N(Q_i)$ since $X$ has nodes in two components of $G \backslash N(Q_i)$, namely, $Q_i$ and $C$. (To see that $X$ has a node in $C$, note that $C$ contains a path from $D_h$ to $D_g$ and $X$ separates $Q_h$ from $D_g$.) By Lemma 4.3, there are two possibilities: (I) except for one component of $G \backslash N(Q_i)$, every component of $G \backslash N(Q_i)$ is contained in $X$. Clearly, the exceptional component is $C$. (II) There is a component $C'$ of $G \backslash X$ that contains $V \backslash (X \cup N(Q_i))$. In Case (I), $|V \backslash (C \cup N(Q_i))| \leq k-1$ because $X \cap C \neq \emptyset$. Hence, from among the $\geq 2k$ superleafs of $G$ at least $2k-(k-1) = k+1$ superleafs are contained in $C \cup N(Q_i)$, and one of these (say, $Q_j$) is disjoint from $N(Q_i)$. In Case (II), since $C' = Q_h$, the remaining superleafs are contained in $X \cup N(Q_i)$. Since $X \cup N(Q_i)$ contains $\geq 2k-1$ superleafs and $|(X \cup N(Q_i)) \backslash Q_h| \leq 2k-1$ ($Q_h$ has at least one node of $N(Q_i)$), we see that every superleaf other than $Q_h$ is a single node, so the superleaf $Q_g$ of $D_g$ is a single node and is disjoint from $N(Q_i)$. This completes the proof of part (1), and shows that if a component of $G \backslash N(Q_i)$ contains at least two leafs, then the component contains a superleaf as well as another (disjoint) leaf.

Now consider part (2). Clearly, $Q_j$ is disjoint from $N(Q_i)$, since $Q_j$ is contained in a component of $G \backslash N(Q_i)$. Suppose that $\{x, y\}$ is not saturating. Then $G$ has a maximal tight set $W_{ij}$ such that $W_{ij} \supseteq Q_i \cup Q_j$. (Cases (ii) and (iii) for $W_{ij}$ cannot occur by Fact 5.5 since $N(Q_i) \cap Q_j = \emptyset$.) Focus on the $k$-separator $N(W_{ij}) = X$. Since $W_{ij}$ contains $Q_i$, $X$ has no nodes from $Q_i$. Then by Lemma 2.1, $X$ cannot mesh with $N(Q_i)$, i.e., all nodes of $X \backslash N(Q_i)$ are contained in one component of $G \backslash N(Q_i)$. Take $H$ to be a component of $G \backslash N(Q_i)$ that contains neither $Q_i$ nor $Q_j$. We now have the desired contradiction. Either $X \backslash N(Q_i)$ is contained in $H$, or $X \backslash N(Q_i)$ is contained in the component of $G \backslash N(Q_i)$ which contains $D_j$ and $D_p$. In the first case, $W_{ij}$ must contain three leafs $D_i$, $D_j$ and $D_p$. In the second case, $W_{ij}$ must contain $H$ (and at least one leaf contained in $H$), because $W_{ij}$ contains all nodes in $N(Q_i) \backslash X$, and each such node has a neighbour in $H$. But, by Fact 5.5, $W_{ij}$ contains no leafs besides $D_i$ and $D_j$. $\square$

By a *J-graph* we mean a $k$-connected graph $G$ such that there is a $k$-shredder $S$ such that every node in $S$ has degree $k$, no two nodes in $S$ are adjacent, $G \backslash S$ has exactly $k$ components, and each of these components contains exactly one leaf. Clearly, $k$ is $\geq 3$. The next lemma and Proposition 5.9 show that a $J$-graph is either the complete bipartite graph $K_{k,k}$, or is obtained from $K_{k,k}$ by fixing one of the two parts of the bipartition, replacing one or more nodes $v$ in this part by appropriate subgraphs $H_v$ on $\geq (k+1)$ nodes, and replacing the $k$ edges incident to $v$ by $k$ edges incident to distinct nodes of $H_v$.

**Lemma 5.8** *Let $G = (V, E)$ be a $k$-connected graph ($k \geq 1$) with $t(G) \geq \max(2k, k+3)$. Let $Q_i \subset V$ be an arbitrary superleaf such that $G \backslash N(Q_i)$ has $b \geq 3$ components (so $N(Q_i)$ is a $k$-shredder). Suppose that each of the $b$ components of $G \backslash N(Q_i)$ contains exactly one leaf. Then:*

*(1) Either $b \geq k+1$ and $b = t(G)$, or $b = k$ and $G$ is a $J$-graph.*

*(2) For a $J$-graph $G$, the minimum number of edges required to augment the connectivity to $(k+1)$ is $(2k-2)$ if $G = K_{k,k}$, and $k + \lceil k/2 \rceil - 1$ otherwise.*

**Proof**: Let $S$ denote the shredder $N(Q_i)$. Let $C_1, C_2, \ldots, C_b$ be the components of $G \backslash N(Q_i)$. Since $t(G) \geq k+3$, $t(G)$ equals the number of leafs, and the leafs are pairwise disjoint (Fact 5.3). We will call a leaf *bad* if it contains one or more nodes of $S$. Similarly, we call a superleaf *bad* if it contains one or more nodes of $S$. The proof of part (1) follows from three (easy) claims.
**Claim 1**: If $b \geq k+1$, then there are no bad leafs, and no bad superleafs.
This claim follows from Lemma 2.1, since no $k$-separator of $G$ meshes with $S$ when $b \geq k+1$, so for every tight set $Q$, either $Q$ is contained in a component of $G \backslash S$ or $Q$ contains two or more components of $G \backslash S$.

From Claim 1, it is clear that if $b \geq k+1$, then $b = t(G)$.
**Claim 2**: If $b \leq k$, then there are $k$ bad leafs, $b = k \geq 3$, and $t(G) = 2k$.
There are at most $k$ bad leafs (since each contains a distinct node of $S$), and exactly $b \leq k$ nonbad leafs (since each component of $G \backslash S$ contains exactly one leaf). So the number of leafs, $t(G)$, is $\leq b + k \leq 2k$. Since $t(G) \geq 2k$, the claim follows.
**Claim 3**: If $b \leq k$, then every bad leaf consists of exactly one node.
Clearly, every bad leaf has exactly one node of $S$, since there are $k$ bad leafs. Let $D$ be a tight set such that (I) exactly one node of $S$ is in $D$, (II) some component, say, $C_k$ of $G \backslash S$, has a node in $D$, and (III) for $j = 1, \ldots, k$, at least one node $x_j$ of $C_j$ is *not* in $D$. Clearly, $|V \backslash (C_k \cup D)| \geq 2k-2 \geq k+1$, because $|S \backslash (C_k \cup D)| = |S \backslash D| = k-1$, and there are $(k-1)$ other nodes $x_1, \ldots, x_{(k-1)}$ not in $C_k \cup D$ (the last inequality holds since $k \geq 3$). Applying Lemma 2.2 to the tight sets $C_k$ and $D$, we see that $C_k \cap D$ is a tight set. Hence, $D$ is not a leaf. This proves the claim.

Suppose that $b \leq k$. Claim 3 implies that every node in $S$ has degree $k$. No two nodes in $S$ are adjacent, since each node $z \in S$ must have a neighbor in each of the $k$ components of $G \backslash S$. Part (1) of the lemma is done: If $b \leq k$, then $b = k$ and $G$ is a $J$-graph.
**Part 2**: Now consider a minimum-cardinality set of new edges whose addition to the $J$-graph $G$ augments the connectivity to $k+1$. If $G = K_{k,k}$, then it is clear that $(2k-2)$ edges are necessary and sufficient.
**Claim 4**: If $G$ is a $J$-graph and $G \neq K_{k,k}$, then $k + \lceil k/2 \rceil - 1$ edges are necessary and sufficient to augment the connectivity to $(k+1)$.
To see the lower bound, note that $G \backslash S$ has $k$ components, so we need to add $\geq (k-1)$ new edges incident to nodes of $G \backslash S$. Moreover, $S$ contains $k$ pairwise disjoint tight sets, so we need to add $\geq \lceil k/2 \rceil$ new edges incident to nodes of $S$. The lower bound follows since the two augmenting edge sets are disjoint. To construct the optimal augmentation, first choose one node in each leaf of each component of $G \backslash S$, and add the edge set of an arbitrary tree that spans these nodes. Then add $\lceil k/2 \rceil$ new edges incident to $S$ such that every node of $S$ is incident to a new edge (i.e., add a maximum matching on $S$, and if $|S|$ is odd, then add one more new edge). Let $G'$ denote the augmented graph. The proof of this claim and part (2) of the lemma follows from the next claim.
**Claim 5**: $G'$ is $(k+1)$-connected.
The proof is by contradiction. If $G'$ is not $(k+1)$-connected, then $G'$ has a $k$-separator $X$. We examine three mutually exclusive cases.
Case (I): $X = S$. By the augmented tree on the leafs in $G \backslash S$, $G' \backslash X$ is connected.
Case (II): $X \neq S$ and $X$ is nonmeshing w.r.t. $S$. Again, by the augmented tree on the leafs in $G \backslash S$, $G' \backslash X$ is connected.
Case (III): $X$ meshes with $S$. By Lemma 2.1, $X$ has a node in each of the $k$ components of $G \backslash S$. Clearly, $X$ and $S$ are disjoint, and every component of $G \backslash S$ has exactly one node of $X$. Let $C_j$ be an arbitrary component of $G \backslash S$ with $|C_j| > 1$ ($C_j$ exists since $G \neq K_{k,k}$). For each $v \in C_j \backslash X$, $G[C_j \cup S]$ has $k$ openly disjoint paths from $v$ to $S$, so at least $(k-1) \geq 2$ of these paths survive in $G' \backslash X$. Hence, all nodes of $(C_j \cup S) \backslash X$, except possibly one node, say, $z \in S$, are in the same component of $G' \backslash X$. Because of the $\lceil k/2 \rceil$ augmented edges incident to $S$, there must

be an augmented edge from $z$ to some node of $S \backslash z$, and so all nodes of $S$ are connected in $G' \backslash X$. Then $G' \backslash X$ is connected. The lemma is proved. □

**Proof**: (**Theorem 5.1**) The splitting-off theorem follows straightaway from Lemmas 5.6–5.8. Let $\widetilde{G}$ be the graph in the theorem, and let $G = \widetilde{G} \backslash s$. Since $t \geq (k+3)$, $G$ has $t$ (pairwise disjoint) leafs, and $t$ (pairwise disjoint) superleafs, by Facts 5.3–5.5. Take an arbitrary superleaf $Q_i$ and focus on the $k$-separator $S = N(Q_i)$. At most $k$ superleafs can intersect $N(Q_i)$, so there must be two superleafs (besides $Q_i$) that are disjoint from $N(Q_i)$. Take these superleafs to be $Q_j$ and $Q_p$. If $S$ is not a shredder, then Lemma 5.6 guarantees a saturating node pair $\{v, w\}$, i.e., in the graph $\widetilde{G}$, the connectivity is preserved on splitting off the edge pair $vs, sw$. If $S$ is a shredder, then depending on whether there is a component of $G \backslash S$ that contains two leafs, either Lemma 5.7 guarantees a saturating node pair $\{v_i, v_j\}$, or Lemma 5.8 guarantees that $G \backslash S$ has $t(G) = \deg_{\widetilde{G}}(s)$ components, or Lemma 5.8 guarantees that $G$ is a $J$-graph. In the first and second cases, we are done (by the first and third items in the consequent of the theorem). If $G$ is a $J$-graph, then either $G = K_{k,k}$ or not. In the first case, we are done, since the theorem allows $G = K_{k,k}$. In the second case, let $S$ be a $k$-shredder of $G$ as in the definition of $J$-graph. For each node $z \in S$, $z$ is a leaf of $G$, and so $\widetilde{G}$ has the edge $zs$. By Lemma 5.8, part (2), splitting off an arbitrary edge pair of $\widetilde{G}$ of the form $z_i s, s z_j$, $i \neq j$, $z_i \in S$, $z_j \in S$ results in a graph $\widetilde{G}_{ij}$ that is $(k+1)$-connected. □

**Remark**: Note that in the last case of the above proof, the graph $\widetilde{G}_{ij}$ resulting from the splitting-off operation will not satisfy the conditions of the theorem, since $t(\widetilde{G}_{ij}) = 2k - 2$.

The next result helps to characterize $J$-graphs.

**Proposition 5.9** *If $G$ is a $J$-graph, $G \neq K_{k,k}$, and $S$ is a $k$-shredder of $G$ as in the definition of a $J$-graph, then the number of nodes in a component of $G \backslash S$ is either one or $\geq k+1$.*

**Proof**: Let $C$ be an arbitrary component of $G \backslash S$, and let $c$ denote the number of nodes in $C$. We get lower and upper bounds on the sum of the degrees of the nodes in $C$ since (I) every node in $C$ has degree $\geq k$ and at most one node in $C$ has degree $k$, (II) there are $\leq \binom{c}{2}$ edges with both ends in $C$, and (III) there are exactly $k$ edges with one end but not the other in $C$:

$$k + (k+1)(c-1) \leq \sum_{v \in C} \deg(v) \leq c(c-1) + k.$$

Then $(c-1)(c-(k+1)) \geq 0$, implying that $c = 1$ or $c \geq k+1$. This proves the claim. □

## 5.3 The augmentation algorithm

We first sketch the augmentation algorithm, and then give the running time analysis. Given a $k$-connected graph $G = (V, E)$, an *augmenting set* means a set $F$ of node pairs (i.e., edges of the complete graph on $V$) such that the augmented graph $(V, E \cup F)$ is $(k+1)$-connected and $E \cap F = \emptyset$. The *slack* of an augmenting set $F$ is the difference between the cardinality, $|F|$, and the lower bound on the number of new edges required for augmenting the connectivity of $G$ by one, namely, $\max(b(G) - 1, \lceil t(G)/2 \rceil)$. Throughout this subsection, we use $N'(.)$ for $N_{G'}(.)$, and $\widetilde{N}(.)$ for $N_{\widetilde{G}}(.)$.

**Theorem 5.10** *Given a $k$-node connected graph with $n \geq (k+2)$, the augmentation algorithm correctly increases the connectivity to $k+1$, and the number of new edges added is at most $k-2$ plus the lower bound of* $\max(b(G) - 1, \lceil t(G)/2 \rceil)$.
*The running time is $O(\min(k, \sqrt{n}) k^2 n^2 + (\log n) k n^2)$.*

**Algorithm 3** *Augment node connectivity by one*

**Input**: Graph $G = (V, E)$, integer $k \geq 1$. $G$ is $k$-connected and $|V| \geq k + 2$.
**Output**: $(k + 1)$-connected graph $G'$ and augmenting set $E(G')\backslash E$ with slack $\leq (k - 2)$.

Let $E' = E$, and $G' = (V, E')$ (initially, $G' = G$).

If $G'$ is $(k + 1)$-connected, then stop else use Algorithm 1 (Section 3, page 6) to compute $b = b(G') = \max_{S \subset V, |S|=k} \#c(G'\backslash S)$.

Obtain a $(k + 1)$-connected graph $\widetilde{G} = (V + s, \widetilde{E})$ from $G'$ by adding a new node $s$ and an (inclusionwise) minimal subset of the edge set $\{sv : v \in V\}$.
Throughout the algorithm $G'$ denotes $\widetilde{G}\backslash s$. Let $t = \deg_{\widetilde{G}}(s) = |\widetilde{N}(s)|$.

While $t \geq 2k$ do (*main loop*)

If $b > \lceil t/2 \rceil$ then

use Jordán's Theorem 2.4 [J 95] to augment the connectivity of $G'$ to $(k + 1)$ by adding a minimum-cardinality edge set, and stop.

End (If).

Let $Q_i$ be an arbitrary superleaf of $G'$.

If either $N'(Q_i)$ is not a shredder of $G'$ (Lemma 5.6) or $N'(Q_i)$ is a shredder of $G'$ and one component of $G'\backslash N'(Q_i)$ contains two leafs (Lemma 5.7) then

find and split off an edge pair incident to $s$ such that $\widetilde{G}$ stays $(k + 1)$-connected;

else

$G'$ is a $J$-graph, so use Lemma 5.8 to (suboptimally) augment the connectivity of $G'$ to $(k + 1)$, and stop.

End (If).

Decrease $t$ by 2 (since we want $t = \deg_{\widetilde{G}}(s)$), and use the dynamic algorithm (Section 3.1, page 8) to update $b = b(G')$.

End (While).

Augment $G'$ (suboptimally) using Phase 5 of Jordán's algorithm [J 95] and stop.

**End**

The proof is given in two parts. The first part proves the correctness and the performance guarantee, and the second part analyzes the running time. The first part follows from similar results for Jordán's algorithm [J 95], but for the sake of completeness, we include the proof in the appendix.

**Proof**: (**Running time analysis**) Our improvement of Jordán's $O(n^5)$ running time mainly comes from (1) replacing the input graph $G$ by a sparse certificate, and (2) using our fast dynamic algorithm for maintaining $b(G')$. At the start of the algorithm, we replace the $k$-connected input graph $G = (V, E)$ by $(V, \hat{E})$, where $\hat{E} \subseteq E$ is a sparse certificate for the $(k+1)$-connectivity of $G$, see [NI 92, CKT 93, FIN 93]. The cardinality of $\hat{E}$ is $< (k+1)n = O(kn)$, and $\hat{E}$ can be computed in linear time by finding a so-called legal ordering of the nodes. The key point is that for every node set $Q \subset V$, $Q$ is a tight set (or a $k$-separator, or a $k$-shredder) of $(V, E)$ iff $Q$ is a tight set (or a $k$-separator, or a $k$-shredder, respectively) of $(V, \hat{E})$, see Proposition 3.4. For the rest of the analysis, assume that the input graph $G$ has $|E(G)| = O(kn)$. Let $\widetilde{G}$ and $G' = \widetilde{G} \backslash s$ be as in the algorithm.

There are four basic steps in the algorithm: (I) determine whether an edge $vs$ of $\widetilde{G}$ is critical, (II) given $v \in \widetilde{N}(s)$, find the leaf and (III) the superleaf of $G' = \widetilde{G} \backslash s$ containing $v$, and (IV) determine whether splitting off the edge pair $vs, sw$ in $\widetilde{G}$ preserves the $(k+1)$-connectivity. The basic steps can be implemented to run in time $O(\min(k, \sqrt{n})|E(\widetilde{G})|) = O(\min(k, \sqrt{n})kn)$ using standard network flow techniques, see [E 79].

Focus on the overall algorithm. The initial computation of $b(G')$ takes time $O(k^2n^2 + k^3n^{1.5})$, by Theorem 3.5. While constructing $\widetilde{G}$, for each node $v_i$ adjacent to $s$ in $\widetilde{G}$, we also find a leaf $D_i$ containing $v_i$. This takes time $O(\min(k, \sqrt{n})kn^2)$, since we need $O(n)$ maximum flow computations. Consider an iteration of the while loop. If $b(G') > \lceil t/2 \rceil \geq k$, then we use the construction in Theorem 2.4 of [J 95]. This takes linear time. Otherwise, we take an arbitrary neighbor $v_i$ of $s$ in $\widetilde{G}$, and compute the superleaf $Q_i$ containing $v_i$. If $N'(Q_i)$ is not a shredder, then we apply Lemma 5.6. We step through the other neighbors of $s$ in $\widetilde{G}$ and compute the corresponding superleafs till we find two superleafs $Q_j$ and $Q_p$ that are each disjoint from $N'(Q_i)$. Let $v_j$ ($v_p$) be the node of $\widetilde{N}(s)$ in $Q_j$ ($Q_p$). We update $\widetilde{G}$ by splitting off either one of the two edge pairs $v_is, sv_j$ or $v_is, sv_p$ (if one of these two preserves the connectivity), or the edge pair $v_js, sv_p$ (otherwise). Applying Lemma 5.6 takes time $O(\min(k, \sqrt{n})k^2n)$, since there are $O(k)$ maximum flow computations. If $N'(Q_i)$ is a shredder, then we first determine whether $G' \backslash N'(Q_i)$ has a component containing two leafs of $G'$. This takes time $O(n)$, since all the leafs of the current $G'$ are available (we computed all the leafs of the initial $G'$, and the surviving leafs stay the same throughout the execution). If there is a component, say, $C$ of $G' \backslash N'(Q_i)$ that contains two leafs, then for each leaf contained in $C$ we construct the superleaf, till we find a superleaf $Q_j$ disjoint from $N'(Q_i)$. Then we split off the edge pair $v_is, sv_j$ in $\widetilde{G}$ (here, $v_j$ is the node in $Q_j \cap \widetilde{N}(s)$). As before, there are $O(k)$ maximum flow computations, and it takes time $O(\min(k, \sqrt{n})k^2n)$ to apply Lemma 5.7. If no component of $G' \backslash N'(Q_i)$ contains two leafs of $G'$, then Lemma 5.8 applies, and gives the optimal augmenting set for the current graph $G'$. Updating $b(G')$ takes time $O((\min(k, \sqrt{n}) + \log n)kn)$, by Theorem 3.6. Summarizing, the $O(n)$ iterations of the while loop take time $O(\min(k, \sqrt{n})k^2n^2 + (\log n)kn^2)$ altogether. Phase 5 of Jordán's algorithm [J 95] takes time $O(\min(k, \sqrt{n})k^3n)$, since it essentially consists of $\binom{t}{2} = O(k^2)$ maximum flow computations. Totaling up, the running time of the algorithm is $O(\min(k, \sqrt{n})k^2n^2 + (\log n)kn^2)$. □

## 6 Conclusions

Very recently, Jordán showed that in a $k$-connected graph $G = (V, E)$ with $|V| \geq 3k - 1$, there are at most $|V|$ $k$-shredders, [J 96b]. Except for the case $|V| < 3k - 1$, this solves one of the open questions in a preliminary version of this paper.

Let $T(n, k)$ denote the running time for testing an $n$-node graph for $k$-connectivity. Is it possible to find all the $k$-shredders of a $k$-connected graph in $o(T(n, k))$ running time?

The algorithm *All-k-shredders* in Section 3 has been implemented by T. Yip at the University of Waterloo (as part of an undergraduate research project). As expected, the most time consuming part of the program is to find the $k$ openly disjoint $v \leftrightarrow r$ paths in Step (1) of *Shredders(r,v)*.

## 7 Appendix

The proofs of some of the known results are given here, for the sake of completeness.

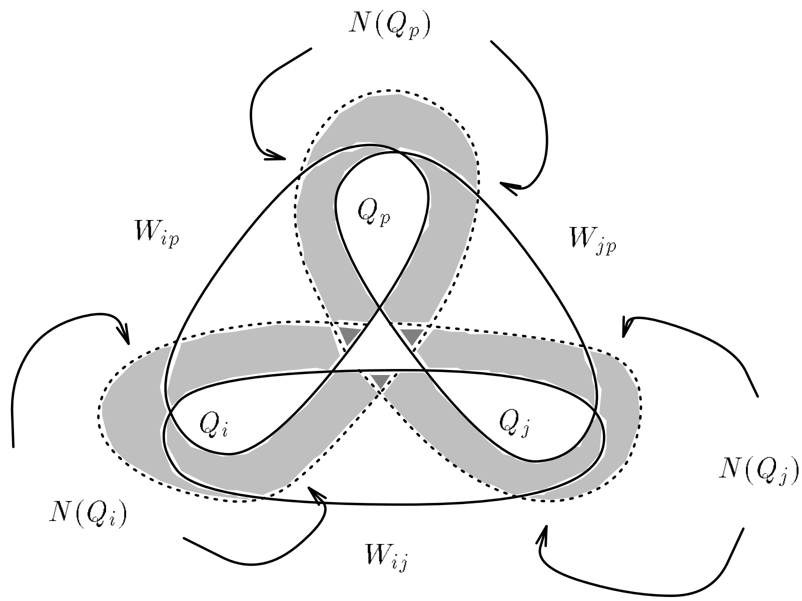


**Figure 4:** An illustration of the proof of Lemma 5.6.

See Figures 2(1) and 4 for illustrations of the next lemma.

**Lemma** 5.6 *Let $G = (V, E)$ be a $k$-connected graph ($k \geq 1$) with $t(G) \geq k + 3$. Let $Q_i$, $Q_j$ and $Q_p$ be three distinct superleafs such that $N(Q_i)$ is disjoint from each of $Q_j$ and $Q_p$. Let $D_i$, $D_j$ and $D_p$ be the leafs contained in $Q_i$, $Q_j$ and $Q_p$ respectively. Then for every three nodes $x \in D_i$, $y \in D_j$ and $z \in D_p$, either one of the node pairs $\{x, y\}$, $\{x, z\}$ or $\{y, z\}$ is saturating, or $N(Q_i) = N(Q_j) = N(Q_p)$, i.e., $N(Q_i)$ is a $k$-shredder.*

**Proof**: Since $t(G) \geq k + 3$, $G$ has $t(G)$ leafs, these are pairwise disjoint, and also, the superleafs are pairwise disjoint (Facts 5.3–5.5). Suppose that $\{x, y\}$ and $\{x, z\}$ are not saturating. Then $G$ has maximal tight sets $W_{ij}$ and $W_{ip}$ such that $W_{ij} \supseteq Q_i \cup Q_j$ and $W_{ip} \supseteq Q_i \cup Q_p$. (Cases (ii) and (iii) for $W_{ij}$ and $W_{ip}$ cannot occur by Fact 5.5 since $Q_j$ and $Q_p$ are disjoint from $N(Q_i)$.) Since $W_{ij} \supseteq Q_i$, $W_{ip} \supseteq Q_i$, both $W_{ij}$ and $W_{ip}$ are tight sets, and there are at least $k$ nodes not in $W_{ij} \cup W_{ip}$ (from the other $t(G) - 3 \geq k$ leafs), Lemma 2.2 shows that $W_{ij} \cap W_{ip}$ is a tight set and

$N(W_{ij} \backslash W_{ip})$ is disjoint from $W_{ip} \backslash (W_{ij} \cup N(W_{ij} \cap W_ip))$. Since $W_{ij} \cap W_{ip}$ satisfies all the conditions for the superleaf containing $D_i$, the maximality of $Q_i$ implies that $Q_i = W_{ij} \cap W_{ip}$. Then $Q_p$ is disjoint from $N(W_{ij} \cap W_ip) = N(Q_i)$, and hence $Q_p$ is disjoint from $N(Q_j)$, since $Q_j \subseteq W_{ij} \backslash W_{ip}$ and $Q_p \subseteq W_{ip} \backslash (W_{ij} \cup N(Q_i))$. If $\{y, z\}$ is saturating, then the proof is done. Otherwise, $G$ has another maximal tight set $W_{jp}$ such that $W_{jp} \supseteq Q_j \cup Q_p$ (Cases (ii) and (iii) for $W_{jp}$ cannot occur by Fact 5.5 since $Q_p$ is disjoint from $N(Q_j)$). As above, we can show that $W_{ij} \cap W_{jp} = Q_j$ and $W_{ip} \cap W_{jp} = Q_p$. Finally, we examine the two sets $(W_{ip} \cup W_{jp})$ and $W_{ij}$. Since $|N(W_{ip} \cup W_{jp})| = k$ (this holds since either $W_{ip} \cup W_{jp}$ is a tight set or there are exactly $k$ nodes not in this set), $W_{ij}$ is a tight set, and $|N(W_{ip} \cup W_{jp} \cup W_{ij})| \geq k$, the submodularity of $|N(Q)|$ implies that the intersection $(W_{ip} \cup W_{jp}) \cap W_{ij} = (W_{ip} \cap W_{ij}) \cup (W_{jp} \cap W_{ij}) = Q_i \cup Q_j$ must have $|N(Q_i \cup Q_j)| = k$. The proof is done since $|N(Q_i \cup Q_j)| = k$ implies that $N(Q_i) = N(Q_j)$. Similarly, it follows that $N(Q_i) = N(Q_p)$. Clearly, $N(Q_i)$ is a $k$-shredder. □

**Theorem** 5.10 (Part 1) *The augmentation algorithm correctly increases the connectivity of a graph $G$ from $k$ to $k+1$, and the number of edges added is at most $k-2$ plus the lower bound of* $\max(b(G) - 1, \lceil t(G)/2 \rceil)$.

**Proof**: (**Correctness and performance guarantee**) Let $G'$ denote $\widetilde{G} \backslash s$ throughout the proof. The initial graph $\widetilde{G}$ has at least $(k+1)$ edges incident to $s$ since $\widetilde{G}$ is $(k+1)$-connected. If $\widetilde{G}$ has $t \geq (k+2)$ edges incident to $s$ each of which is critical, then $t(G') = t$, and $G'$ has $t$ (pairwise disjoint) leafs and $t$ (pairwise disjoint) superleafs, by Facts 5.3–5.5. (The case $k = 1$ and $t = \deg_{\widetilde{G}}(s) = k + 1 = 2$ is an exception, because for $k = 1$ the leafs of $G'$ are always pairwise disjoint, so $G'$ satisfies the previous assertion in this case too.) For $k > 1$, if the initial graph $\widetilde{G}$ has exactly $(k+1)$ edges incident to $s$, then the leafs of $G'$ may not be pairwise disjoint, and possibly $t(G')$ is less than $\deg_{\widetilde{G}}(s)$. Nevertheless, every leaf of $G'$ must contain at least one of the neighbors of $s$ in $\widetilde{G}$, since $\widetilde{G}$ is $(k+1)$-connected.

First, consider a nonterminal iteration of the while loop. Then we have $t = \deg_{\widetilde{G}}(s) \geq 2k$, and $b(G') \leq \lceil t/2 \rceil$. If $t \geq (k+3)$, then by Theorem 5.1 and Lemmas 5.6–5.7, we add a new edge $v_i v_j$ to $G'$ such that $t(G')$ decreases by two. In terms of $\widetilde{G}$, we split off an edge pair $v_i s, s v_j$ that preserves the $(k+1)$-connectivity. (For $k = 1, 2,\ \ t \geq 2k$ does not imply $t \geq (k+3)$. But then we have one of the special cases $k = 1, t = 2,\ \ k = 1, t = 3$, or $k = 2, t = 4$, and it can be proved that for each of these cases there exists a new edge whose addition to $G'$ decreases $t(G')$ by two.) Thus every nonterminal iteration of the while loop satisfies a key property:

> the cardinality of the augmenting set increases by one, and the lower bound decreases by one.

At the terminating steps of the algorithm, if we can prove that the slack for the current graph $G'$ is at most $(k-2)$, then this key property guarantees that the slack for the original graph $G$ is at most $(k-2)$.

To complete the proof, we examine each of the cases in which the algorithm terminates, and show that the slack for the current graph $G'$ is at most $(k-2)$. If the current graph $G'$ in an execution of the while loop has $b(G') > \lceil t(G')/2 \rceil \geq k$, then a minimum-cardinality augmenting set for $G'$ is easily found by Theorem 2.4 of [J 95]:

> Suppose that a $k$-connected graph $G$ has $b(G) \geq k+1$ and $b(G) > \lceil t(G)/2 \rceil$. Then there is an augmenting set of cardinality $b(G) - 1$.

In this case, the overall augmenting set for the original graph $G$ is optimal. If the current graph $G'$ in an execution of the while loop is a $J$-graph, then a minimum-cardinality augmenting set $F'$ for

$G'$ is easily found by Lemma 5.8. In this case, the overall augmenting set $F$ for the original graph $G$ may not be optimal, because $|F'|$ exceeds the lower bound for the current graph $G'$. However, the slack of $F$ for the original graph is at most $k-2$, because the slack of $F'$ for the current graph is at most $k-2$.

If $t = \deg_{\widetilde{G}}(s) < 2k$, either initially or after several iterations of the while loop, then the algorithm executes the last step (Phase 5 of Jordán's algorithm). This step increases the connectivity of the current graph $G'$ to $(k+1)$ by adding an (inclusionwise) minimal subset $F'$ of the edges $\{v_i v_j : 1 \leq i < j \leq t\}$, where $v_1, \ldots, v_t$ are the neighbors of $s$ in $\widetilde{G}$. As shown in [J 95], a well-known result of Mader implies that $F'$ contains no cycles.

> Mader's result is: In a $(k+1)$-connected graph, a cycle consisting of critical edges must be incident to at least one node of degree $k+1$.

Hence, $|F'| \leq (t-1)$. If $t \geq (k+2)$, then since $t = t(G')$, the lower bound is $\geq \lceil t(G')/2 \rceil = \lceil t/2 \rceil$, and so the slack is $\leq (t-1) - \lceil t/2 \rceil \leq (k-2)$, since $t \leq (2k-1)$. Otherwise, if $t \leq (k+1)$, then possibly $t(G') < t$, but we may assume $t(G') \geq 3$, so the slack is $\leq (k) - \lceil 3/2 \rceil \leq (k-2)$. (The algorithm can easily recognize the special case $t(G') = 2$, and find a one-edge augmenting set by [J 95, Lemma 3.2].) $\square$

# References


[BBM 90] D. Bienstock, E. F. Brickell and C. L. Monma, "On the structure of minimum-weight $k$-connected spanning networks," *SIAM J. Discrete Math.* **3** (1990), 320–329.

[CKT 93] J. Cheriyan, M. Y. Kao and R. Thurimella, "Scan-first search and sparse certificates: An improved parallel algorithm for $k$-vertex connectivity," *SIAM J. Computing* **22** (1993), 157–174.

[CLR 90] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1990.

[E 79] S. Even, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.

[ET 76] K. P. Eswaran and R. E. Tarjan, "Augmentation problems," *SIAM J. Computing* **5** (1976), 653–665.

[FIN 93] A. Frank, T. Ibaraki and H. Nagamochi, "On sparse subgraphs preserving connectivity properties," *J. Graph Theory* **17** (1993), 275–281.

[F 92a] A. Frank, "Augmenting graphs to meet edge-connectivity requirements," *SIAM J. Disc. Math.* **5** (1992), 25–53.

[F 92b] A. Frank, "On a theorem of Mader," *Annals of Discrete Math.* **101** (1992), 49–57.

[F 93] A. Frank, "Submodular functions in graph theory," *Discrete Math.* **111** (1993), 231–243.

[F 94] A. Frank, "Connectivity augmentation problems in network design," in *Mathematical Programming: State of the Art 1994*, (Eds. J. R. Birge and K. G. Murty), The University of Michigan, Ann Arbor, MI, 1994, 34–63.

[FJ 95a] A. Frank and T. Jordán, "Minimal edge-coverings of pairs of sets," *J. Combinatorial Theory Series B* **65** (1995), 73–110.

[FJ 95b] A. Frank and T. Jordán, "How to make a strongly connected digraph two-connected," *Proc. 4th I.P.C.O.*, Egon Balas and Jens Clausen (Eds.), LNCS 920, Springer-Verlag, Berlin, (1995), 414–425, [G 80] Z. Galil, "Finding the vertex connectivity of graphs," *SIAM J. Computing* **9** (1980), 197–199.

[GW 95] M. X. Goemans and D. P. Williamson, "The primal-dual method for approximation algorithms and its application to network design problems," In *Approximation Algorithms for NP-hard Problems*, (Ed. D. S. Hochbaum), PWS Publishing Co., Boston, MA, 1995.

[HRG 96] M. R. Henzinger, S. Rao and H. N. Gabow, "Computing vertex connectivity: new bounds from old techniques," *Proc. 37th IEEE F.O.C.S.* (1996), 462–471.

[HR 91] T. Hsu and V. Ramachandran, "A linear time algorithm for triconnectivity augmentation," *Proc. 32nd IEEE F.O.C.S.* (1991), 548–559.

[H 92] T. Hsu, "On four-connecting a triconnected graph," *Proc. 33rd IEEE F.O.C.S.* (1992), 70–79.

[HR 93] T. Hsu and V. Ramachandran, "Finding a smallest augmentation to biconnect a graph," *SIAM J. Computing* **22** (1993), 889–912.

[H 95] T. Hsu, "Undirected vertex-connectivity structure and smallest four-vertex-connectivity augmentation," *Proc. 6th ISAAC* (1995).

[J 93] T. Jordán, "Increasing the vertex-connectivity in directed graphs," *Proc. Algorithms — ESA'93, 1st Annual European Symposium*, LNCS 726, Springer, New York (1993), 236–247.

[J 95] T. Jordán, "On the optimal vertex-connectivity augmentation," *J. Combinatorial Theory, Series B* **63** (1995), 8–20. Preliminary version in *Proc. 3rd I.P.C.O.* (1993), 75-88.

[J] T. Jordán, Ph. D. thesis.

[J 96] T. Jordán, Personal communication, February 1996.

[J 96b] T. Jordán, "On the number of shredders," manuscript, November 1996.

[K 95] D. Karger, "A randomized fully polynomial time approximation scheme for the all terminal network reliability problem," *Proc. 27th ACM S.T.O.C.* (1995), 11–17.

[Lo 74] L. Lovász, Lecture at Conference in Graph Theory, Prague, 1974.

[Ma 78] W. Mader, "A reduction method for edge-connectivity in graphs," *Annals of Discrete Math.* **3** (1978), 145–164.

[Ma 82] W. Mader, "Konstruktion aller $n$-fach kantenzusammenhängenden Digraphen," *European J. Combinatorics* **3** (1982), 63–67.

[NI 92] H. Nagamochi and T. Ibaraki, "A linear-time algorithm for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph," *Algorithmica* **7** (1992), 583–596.

[PB 83] J. S. Provan and M. O. Ball, "The complexity of counting cuts and of computing the probability that a graph is connected," *SIAM J. Computing* **12** (1983), 777–788.

[RW 95] R. Ravi and D. P. Williamson, "An approximation algorithm for minimum-cost vertex-connectivity problems," Preliminary version in *Proc. 6th ACM-SIAM S.O.D.A.* (1995), 332-341. To appear in *Algorithmica.*

[WN 90] T. Watanabe and A. Nakamura, "A smallest augmentation to 3-connect a graph," *Discrete Appl. Math.* **28** (1990), 183–186.