Using Expander Graphs to Find Vertex Connectivity

Harold N. Gabow *

September 4, 2000

Abstract

The (vertex) connectivity κ of a graph is the smallest number of vertices whose deletion separates the graph or makes it trivial. We present the fastest known algorithm for finding κ. For a digraph with n vertices, m edges and connectivity κ the time bound is $O((n+\min\{\kappa^{5/2}, \kappa n^{3/4}\})m)$. This improves the previous best bound of $O((n+\min\{\kappa^3, \kappa n\})m)$. For an undirected graph both of these bounds hold with m replaced by κn. Our approach uses expander graphs to exploit nesting properties of certain separation triples.

1 Introduction

The *(vertex) connectivity* κ of a graph is the smallest number of vertices whose deletion separates or trivializes the graph. (Other basic terminology is defined at the end of this section.) This is a central concept of graph theory [11]. Computing the connectivity is posed as Research Problem 5.30 in [1] where in fact a linear-time algorithm is conjectured. Yet relatively little progress has been made on computing connectivity.[1] If the conjectured linear-time algorithm exists it involves techniques that are radically different from the known ones.

Here we present the most efficient known algorithm for computing graph connectivity. We first summarize the results most relevant to ours. Other work on vertex connectivity, including randomized algorithms, is surveyed in [10]. To *compute the connectivity* means to find κ and a corresponding separator. Throughout this paper n and m denote the number of vertices and edges of the given graph, respectively.

Henzinger, Rao and Gabow give an algorithm to compute the connectivity of a digraph in time $O(\min\{\kappa^3+n, \kappa n\}m)$ [10]. This algorithm applies two different high-level approaches to finding the connectivity, due to Even and Tarjan [7] and Even [6]. For undirected graphs the bound improves using the maximal forest decomposition of Nagamochi and Ibaraki [15]. This allows the graph to be pruned to $O(\kappa n)$ edges. So the time bound of [10] for undirected graphs is the above bound with m replaced by κn.

Our algorithm runs in time $O((n+\min\{\kappa^{5/2}, \kappa n^{3/4}\})m)$ for digraphs. For undirected graphs m can be replaced by κn in this bound. To compare our digraph bound with [10] observe that both bounds are $O(nm)$ when $\kappa \le n^{1/3}$. For larger values of κ the new algorithm is faster: For $n^{1/3} \le \kappa \le \sqrt{n}$, [10] is $O(\kappa^3 m)$; in this range the new bound is $O(nm)$ for $\kappa \le n^{2/5}$ and $O(\kappa^{5/2}m)$ for $\kappa \ge n^{2/5}$. For $\kappa \ge \sqrt{n}$, [10] is $O(\kappa nm)$ and the new bound is $O(\kappa n^{3/4}m)$.

To *check k-connectedness* for a given k means to verify that $\kappa \ge k$ or else find κ and a corresponding separator. Let δ denote the minimum degree of the given graph. Henzinger [9] uses

*Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309. e-mail: hal@cs.colorado.edu

[1]This contrasts with a fairly large number of diverse and efficient algorithms for computing edge connectivity.

maximal forest decompositions to check $\delta/2$-connectedness of an undirected graph. The time is $O(\min\{\kappa, \sqrt{n}\}n^2)$. Henzinger's algorithm is also based on the relation $\delta - \kappa \geq \kappa$ that holds when $\kappa \leq \delta/2$. Our work (for both finding connectivity and checking k-connectedness) investigates general properties of this "gap" $\delta - \kappa$. We show how to check $a\delta$-connectedness, where a is an arbitrary fixed constant < 1, in time $O((\kappa + \sqrt{n})\sqrt{n}m)$ for digraphs and the same bound with m replaced by κn for undirected graphs. This is faster than our general connectivity finding algorithm.

We present a nesting property of certain separation triples in undirected graphs, and a generalization to digraphs. This eventually leads to a two step algorithm to compute vertex connectivity. The first step enlarges the gap $\delta - \kappa$ to some guaranteed size. (Initially the gap can be 0.) The second step is an algorithm to find the connectivity of a graph with a large gap.

Expander graphs play a crucial role in the second step. There are many diverse important applications of expander graphs (see e.g. the lists in [3, 17], and [4], [16]) but we are unaware of similar applications to graph algorithms.

Here is a simple example illustrating how expanders can help find vertex connectivity: Suppose in an undirected graph $G = (V, E)$ we want to check that any set of exactly $n/3$ vertices has $\geq n/3$ neighbors. (This is clearly necessary to have $\kappa \geq n/3$.) Let X be a graph on vertex set V, such that any set of $n/3$ vertices has at least $n/3$ neighbors in X. Such an X with $O(n)$ edges can be constructed using an expander graph. If A is a set of $n/3$ vertices with $< n/3$ neighbors, some edge e of X joins A to a nonneighbor. So the ends of e can be separated by $< n/3$ vertices. We can test for the existence of such an e by computing a maximum flow for every edge of X.

Although our asymptotic time bounds for undirected graphs are implied by those for digraphs, we present a separate algorithm for undirected graphs. Undirected graphs have a stronger nesting property, which results in a simpler algorithm. Section 2 presents the connectivity algorithm for undirected graphs with large gaps. Section 3 gives the undirected gap enlargement algorithm. Sections 4–5 give the analogous results for digraphs, Section 4 first reviewing digraph fundamentals and then presenting the gap enlargement algorthm, Section 5 presenting the more involved gap algorithm. Section 5 also presents our algorithm to check $a\delta$-connectedness. Section 6 combines the gap enlargement and large gap algorithms to get our connectivity algorithm. Readers interested only in undirected graphs can skip Sections 4–5. The rest of this section gives notation, definitions and some background.

We usually denote singleton sets $\{v\}$ by v, as in $S - v$. Consider a graph $G = (V, E)$. Assume G is undirected; the minor changes needed when G is directed are discussed in Section 4. If H is not the given graph we usually refer to its vertex and edge set as $V(H)$ and $E(H)$ respectively. δ denotes the minimum degree of a vertex of G. A *neighbor* of a set of vertices X is a vertex $y \notin X$ on an edge (x, y) for some $x \in X$. A *nonneighbor* of X is a vertex $y \notin X$ that is not a neighbor. If a function refers to a graph we include the graph as an extra argument if it is not clear, e.g., $\delta(G)$, $n(G)$.

A *separation triple* is an ordered triplet (S, X, Y) of sets forming a partition of V, with X and Y nonempty and no edge going from X to Y. S is the *separator* and X and Y are the *shores* of the triple. The *vertex connectivity* κ is the smallest cardinality of a separator, or $n-1$ if the graph is complete. A *κ-separation triple* has $|S| = \kappa$. In that case S is a *κ-separator* and X and Y are *κ-shores.*

For $x, y \in V$, $\kappa(x, y) = \min\{|S|, n-1 : (S, X, Y)$ is a separation triple with $x \in X$, $y \in Y\}$. Note that $\kappa(x, y) = n - 1$ exactly when $x = y$ or $(x, y) \in E$. The *rooted connectivity* at x is $\kappa(x) = \min\{\kappa(x, y) : y \in V\}$.

We also use a notion introduced by Even [6]: For a set of vertices S, $\kappa_W(S) = \min\{|S|, |C| : (C, A, B)$ is a separation triple of G with $S \subseteq A \cup C\}$. (The "W" stands for weak separation.) To compute $\kappa_W(S)$ first form the graph $G_W(S)$ by starting with G and adding a new vertex s with edges (s,v), $v \in S$. Then $\kappa_W(S) = \kappa(s, G_W(S))$.

A quantity $\kappa(x,y)$ can be computed using one max flow computation on a network with unit vertex capacities [2]. The time is $O(\sqrt{n}m)$. Alternatively for any value k the time is $O(km)$ if we must either find $\kappa(x,y)$ or verify that $\kappa(x,y) \geq k$. A quantity $\kappa(x)$ can be computed in $O(nm)$ time [10]. Alternatively the time is $O(n(n-\delta)^2)$. This follows from [10, p. 233]. There the bound $O(n(n-\kappa)^2)$ is shown, but the same argument proves the stronger bound. (We prove a similar result for max flows in Lemma 2.5(iii).)

2 Gap Algorithm

A set $R \subseteq V$ is *ρ-rich* if it contains at least ρ vertices from each shore of some κ-separation triple. Hence $|R| \geq 2\rho$. R is *ρ-superrich* if it contains at least ρ vertices from each shore of every κ-separation triple.

Lemma 2.1 (i) *The neighbors of a vertex v with $\kappa(v) > \kappa$ form a ρ-superrich set for $\rho = \min\{\kappa(v), \delta\} - \kappa + 1$.*

(ii) *Any set $S \subseteq V$ with $\kappa_W(S) > \kappa$ is ρ-superrich for $\rho = \min\{\kappa_W(S), \delta + 1\} - \kappa$.*

Proof: (i) Let (C, A, B) be any κ-separation triple. Clearly C contains v. Let S be the set of neighbors of v and assume $|A \cap S| < \rho$. Since every vertex has degree at least δ, $|A| + |C| > \delta$. Thus $|A| \geq \delta - \kappa + 1 \geq \rho$. We conclude that $A - S \neq \emptyset$. Now it is easy to see that $(C \cup (A \cap S) - v, A - S, B \cup v)$ is a separation triple. This implies $\kappa(v) \leq |C \cup (A \cap S) - v| < \kappa + \rho - 1 \leq \kappa(v)$, a contradiction.

(ii) As in part (i) for any κ-separation triple (C, A, B) with $|A \cap S| < \rho$, $(C \cup (A \cap S), A - S, B)$ is a separation triple. This implies $\kappa_W(S) \leq |C \cup (A \cap S)| < \kappa + \rho \leq \kappa_W(S)$, a contradiction. $\square$

A shore of a separation triple (C, A, B) is *extreme* if each of its nonempty subsets has at least $|C|$ neighbors. As examples, a separation triple (C, A, B) has B extreme if it is a κ-separation triple, or it is a separation triple corresponding to $\kappa(x)$ (i.e, $x \in A$ and $|C| = \kappa(x)$) or it is a separation triple corresponding to $\kappa_W(S)$ (i.e, $S \subseteq A \cup C$ and $|C| = \kappa_W(S)$). The next lemma is illustrated in Fig. 1.

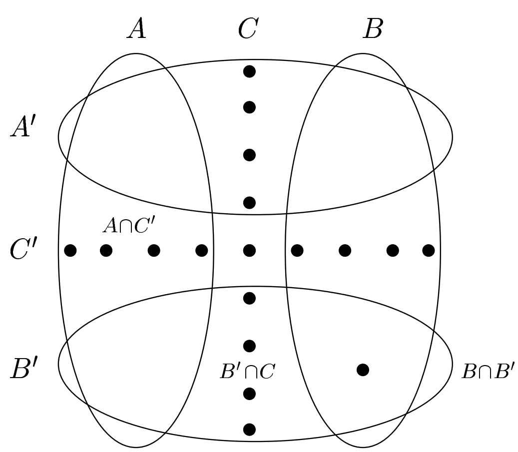

Figure 1: The two separation triples in Lemma 2.2.

Lemma 2.2 *Let (C, A, B) and (C', A', B') be separation triples with B' extreme and $B \cap B' \neq \emptyset$.*

(i) $|B' \cap C| \geq |A \cap C'|$.

(ii) *$A \cap A'$ has at most $|C|$ neighbors.*

Proof: We first show that for two arbitrary separation triples (C, A, B) and (C', A', B'), the neighbors of $B \cap B'$ are contained in $(B' \cap C) \cup (C' - A)$. In proof since no edge joins B' and A', any neighbor of $B \cap B'$ belongs to B' or C'. Since no edge joins B and A, any neighbor of $B \cap B'$ actually belongs to $B' \cap C$ or $C' - A$.

(i) Since B' is extreme, $B \cap B'$ has at least $|C'|$ neighbors, i.e., $|B' \cap C| + |C' - A| \geq |C'|$. This implies the desired inequality.

(ii) The preliminary observation shows the neighbors of $A \cap A'$ are contained in $(A \cap C') \cup (C - B')$. By (i) the size of this set is $|A \cap C'| + |C - B'| \leq |B' \cap C| + |C - B'| = |C|$. □

We apply the lemma to prove the following nesting property, an important ingredient in the gap algorithm.

Lemma 2.3 *Let (C, A, B) be a separation triple with B extreme. Either*

(i) *A or C contains a κ-shore, or*

(ii) *A or B is contained in a κ-separator, and C is $\min\{|A|, |B|\}$-rich.*

Proof: Let (C', A', B') be a κ-separation triple. We can assume

$$A' \not\subseteq C, \; B' \not\subseteq C \tag{1}$$

since otherwise (i) clearly holds. Next we show that we can assume the first part of (ii), more precisely either $A \subseteq C'$ or $B \subseteq C'$. Suppose not, i.e.,

$$A \not\subseteq C', \; B \not\subseteq C'. \tag{2}$$

After possibly interchanging sets A' and B' we have $A \cap A'$, $B \cap B' \neq \emptyset$. (To see this choose A' so $A \cap A' \neq \emptyset$ by (2). If this makes $B \cap B' = \emptyset$ then $B \cap A' \neq \emptyset$ by (2) and $A \cap B' \neq \emptyset$ by (1).) Now apply Lemma 2.2(ii) (with the primed and nonprimed triplets interchanged) to show that $A \cap A'$ has at most $|C'| = \kappa$ neighbors. Since $A \cap A'$ is nonempty and has nonneighbors, it is a κ-shore and (i) holds.

It remains to show the second part of (ii). This argument will not use the extremeness of B, i.e., it is symmetric in A and B. So without loss of generality assume $B \subseteq C'$. Then (1) implies $A' \cap A$, $B' \cap A \neq \emptyset$. Lemma 2.2(i) with A' extreme and $A' \cap A \neq \emptyset$ shows $|A' \cap C| \geq |B \cap C'| = |B|$. Similarly Lemma 2.2(i) with B' extreme and $B' \cap A \neq \emptyset$ shows $|B' \cap C| \geq |B \cap C'| = |B|$. This gives the second part of (ii). □

Recall that an *(n, d, c)-expander* is a d-regular bipartite multigraph (V, W, F) where $|V| = |W| = n/2$ such that any subset $S \subseteq V$ has at least $(1 + c(1 - 2|S|/n))|S|$ neighbors [14]. Our techniques can be applied using linear-sized (n, d, c) expanders, such as the graphs of Gabber and Galil [14]. This leads to a vertex connectivity algorithm with the same bounds as this paper for graphs with $\kappa \leq n^{3/4}$. To achieve the full range of our results (allowing $\kappa \leq n$) we rely on graphs with a stronger expansion property.

We use the family of graphs $X_{n,d}$ constructed by Lubotzky, Phillips and Sarnak [12] and Margulis [13] with the following property. There is a constant C_0 such that for any integer d where $d-1$ is a prime power congruent to 1 modulo 4, and any positive integer r, there is an integer n, $r \leq n \leq C_0 r$

and a regular graph $X_{n,d}$ that has n vertices, degree d, and every eigenvalue of its adjacency matrix except for the trivial eigenvalue d has magnitude at most $2\sqrt{d-1}$. Furthermore the graphs $X_{n,d}$ can be constructed in time linear in their size, $O(nd)$. The graphs can be constructed edge-by-edge using no auxiliary storage. The eigenvalue property implies the following expansion property: Any two sets $A, B \subseteq V(X_{n,d})$ are joined by an edge if

$$|A|\,|B|\,d > 4n^2. \tag{3}$$

This property follows trivially from [5, p. 122, Corollary 2.5] (also see [14, p. 160, Problem 6.27]).

Consider our given graph $G = (V, E)$. For any set $R \subseteq V$ and any positive value d, define a graph $R_d = (R, F)$ as follows. Take d' to be one more than a prime power congruent to 1 modulo 4 with $d \leq d' \leq 4d$. Choose n so that $|R| \leq n \leq C_0|R|$ and the above expander graph $X_{n,d'}$ exists. Identify each vertex of R with a unique vertex of $X_{n,d'}$. Let F consist of the edges induced by $X_{n,d'}$ on R.

Lemma 2.4 *Take any $k \geq \kappa$ and let R be a ρ-rich set of r vertices. Set $\rho' = \max\{\rho,\ (r-k)/2\}$ and $d = 1 + 4(C_0 r)^2/(\rho\rho')$. Then*

$$\kappa = \min\{\kappa(x,y) : (x,y) \text{ an edge of } R_d\}.$$

Proof: The assumption on R shows there is a κ-separation triple (C, A, B) such that, writing $\alpha = |A \cap R|$ and $\beta = |B \cap R|$, we have $\alpha \geq \beta \geq \rho$. Also $\alpha \geq (r-\kappa)/2 \geq (r-k)/2$. Hence $\alpha\beta d' \geq \rho'\rho d > 4(C_0 r)^2$. Thus (3) implies R_d has an edge (x,y) joining vertices of A and B. So $\kappa(x,y) = \kappa$. $\square$

Now we present our first algorithm. All the algorithms in this paper calculate a number of separation triples. They maintain (C_0, A_0, B_0) as a triple having the smallest separator C_0 seen so far. We say the algorithm *has found* κ when $|C_0| = \kappa$. So $|C_0| > \kappa$ as long as the algorithm has not found κ.

The Gap Algorithm checks k-connectedness, i.e., given k it finds a κ-separator if $\kappa < k$ or it determines that $\kappa \geq k$. Define the "gap" γ by

$$\gamma = \delta - k,\ \ \tau = \gamma/2.$$

Assume $\gamma \geq 4$. (This assumption can be weakened but it suffices for our purposes.) As we present the algorithm we also verify its correctness. The comments verifying correctness assume that $\kappa < k$.

Gap Algorithm

Step 1. If $n < 2\delta$ then set R to V and go to Step 4. Note that V is τ-rich since a κ-shore has $\geq \delta + 1 - \kappa > \delta - k = 2\tau$ vertices.

Step 2. Let a be a vertex of degree δ. Find $\kappa(a)$ and a corresponding separation triple (C, A, B) with $a \in A$. Initialize (C_0, A_0, B_0) to this triple.

Step 3. If κ has not been found, this step either finds κ or makes R a τ-rich set of $\leq \delta$ vertices.

If $\kappa(a) \geq k + \tau$ then let R be the set of neighbors of a and go to Step 4. Since $\kappa(a) \leq \delta$, Lemma 2.1(i) implies that R is the desired τ-rich set.

Consider the opposite case $\kappa(a) < k + \tau$. We will apply Lemma 2.3 to the separation triple (C, A, B). Let D be any set of k vertices in $A \cup B$. (D exists since $n \geq 2\delta > 2k + \tau > k + |C|$.)

Compute $\kappa_W(C)$ and $\kappa_W(D)$ and corresponding separation triples and update (C_0, A_0, B_0). Set R to C and go to Step 4.

Observe that if A contains a κ-shore then $\kappa = \kappa_W(C)$ (since we can assume $|C| > \kappa$). Similarly if C contains a κ-shore then $\kappa = \kappa_W(D)$. The remaining case is Lemma 2.3(ii). Since every vertex has degree at least $\delta = k + 2\tau$ and $|C| < k + \tau$ we have $|A|, |B| \geq \delta + 1 - |C| > \tau$. So either we have found κ or C is the desired τ-rich set.

Step 4. If κ has not been found and $\kappa < k$ then R is now τ-rich and either $R = V$ with $n < 2\delta$, or $|R| \leq \delta \leq n/2$. If $R \neq V$ add vertices to make $|R| = 2\delta$. Apply Lemma 2.4, constructing the expander R_d and computing the values $\kappa(x, y)$ specified in the lemma. Update (C_0, A_0, B_0) for these values. At this point if $\kappa < k$ the algorithm has found κ. $\square$

The next lemma gives three time bounds for this algorithm. (i) is the main bound. (ii) and (iii) are used when κ and δ are very close to n (Lemma 6.2). These two bounds assume max flow and rooted connectivity computations are performed as described in the proof.

Lemma 2.5 *If $\gamma \geq 4$ then the Gap Algorithm either verifies that $\kappa \geq k$ or finds a κ-separator.*

(i) *If $n \geq 2\delta$ the time is* $O\left(\left(n + \frac{\delta^2}{\gamma}\min\{\delta, \sqrt{n}\}\right) m\right)$.

(ii) *Alternatively if $n \geq 2\delta$ the time is* $O\left(\left(n + \frac{\delta^2 \sqrt{n}}{\gamma \log_{4n/\delta} n}\right) m\right)$.

(iii) *In general the time is* $O\left(\frac{n^{9/2}}{\gamma \log (n-\delta)}\right)$.

(iv) *If $\gamma \geq \epsilon\delta$ for some fixed $\epsilon > 0$ the time is* $O((\sqrt{n} + \delta)\sqrt{n}m)$.

Proof: (i) We will show the time for the Gap Algorithm is dominated by the time for $O(1)$ rooted connectivity computations and $O(\delta^2/\gamma)$ max flow computations. As mentioned in Section 1 we compute a rooted connectivity in $O(nm)$ time and a max flow in $O(\min\{k, \sqrt{n}\}m)$ time. These bounds imply (i).

Steps 2 and 3 compute 3 rooted connectivities. Step 4 does a max flow computation for each edge of R_d. Thus the time to construct the expander R_d is dominated by the max flow time, and there are $\leq (4d)r$ max flow computations. Since $r = 2\delta \geq \delta + k$ Lemma 2.4 gives $d = O(\delta^2/(\tau\delta)) = O(\delta/\gamma)$. This implies the desired bound.

(ii) To achieve this bound Step 4 computes max flows using the algorithm of [8]. This algorithm finds a maximum flow on a graph with unit vertex capacities in time $O(\sqrt{n}m/\log_b n)$ for $b = n^2/m$. If F is the flow graph in our computations then $n(F) = 2n$, $m(F) = 2m + n \geq \delta n$ [2]. Thus $b(F) \leq 4n/\delta$. The desired time bound now follows as in part (i).

(iii) As mentioned in Section 1 a rooted connectivity computation uses $O(n(n-\delta)^2)$ time. This is dominated by the desired time bound since we can assume $n^{3/2} \geq \gamma \log n$. Hence we need only analyze the time to compute max flows in Step 4.

When $n < 2\delta$ we have $R = V$ and Lemma 2.4 gives $d = O(n^2/(\tau(n-k))) = O(n^2/(\gamma(n-\delta)))$. This bound also holds when $n \geq 2\delta$ since part (i) shows $d = O(\delta/\gamma)$. Thus $O(n^3/(\gamma(n-\delta)))$ max flows are computed. It suffices to compute each flow in time $O(n + (n-\delta)^{5/2}/\log(n-\delta))$.

Recall the flow graph for computing $\kappa(x, y)$ has two vertices v_T, v_S and a unit capacity edge (v_T, v_S) for each $v \in V$, and infinite capacity edges (v_S, w_T) and (w_S, v_T) for each $(v, w) \in E$. The source is x_S and the sink is y_T. Each edge (x_S, v_T) can be contracted into the source, and each edge (v_S, y_T) can be contracted into the sink. This leaves $\le n-\delta$ vertices of each type v_S and v_T. Hence the contracted graph has $O(n-\delta)$ vertices. We construct the contracted graph in time $O(n+(n-\delta)^2)$ (using an adjacency matrix for the given graph) and find a maximum flow on this graph in time $O((n-\delta)^{5/2}/\log{(n-\delta)})$ (since a maximum flow in a graph with unit vertex capacities can be found in time $O(n^{5/2}/\log n)$ [8]). This implies the desired bound for computing a max flow in Step 4. □

3 Gap Enlargement

This section gives an algorithm that modifies the given graph to enlarge the gap $\delta - \kappa$.

Say that a digraph $G' = (V', E')$ is *sub-conformal* to $G = (V, E)$ if $V' \subseteq V$ and the κ-separators of G are precisely the minimum separators of G' with vertices $V - V'$ added in. When $V' = V$ say that G' is *conformal* to G.

Lemma 3.1 (i) *Let $R \subseteq V$ be 1-superrich. Any vertex x with $\kappa(x) = \kappa$ has $\kappa = \min\{\kappa(x, y) : y \in R\}$.*

(ii) *For any vertex x with $\kappa(x) > \kappa$, $G - x$ is sub-conformal to G.*

(iii) *For any vertices x, y with $\kappa(x, y) > \kappa$, $G + (x, y)$ is conformal to G.*

Proof: (i) Some κ-separation triple (C, A, B) has $x \in A$. R contains a vertex $y \in B$. Thus $\kappa(x, y) = \kappa$.

(ii) x belongs to every κ-separator of G. Hence $\kappa(G - x) = \kappa - 1$ and (ii) follows.

(iii) No κ-separation triple has x and y on opposite shores. Hence (iii) follows. □

Now we present the gap enlargement algorithm. It is called with a value Δ. It either finds a κ-separator or increases the gap $\delta - \kappa$ by at least Δ.

The algorithm transforms the given graph G into a new graph H. During the algorithm H always denotes the current graph. Initially H is G.

We shall see that H is constructed by deleting vertices of G and adding edges. This implies that a separation triple (C, A, B) of H gives a separation triple $(C \cup (V - V(H)), A, B)$ of G. Call $(C \cup (V - V(H)), A, B)$ the *expansion* of (C, A, B). As before the algorithm maintains (C_0, A_0, B_0) as a separation triple of G with the smallest separator C_0 seen so far. So each time the algorithm finds a separation triple of H, we use its expansion to update (C_0, A_0, B_0).

The algorithm maintains this invariant:

If κ has not been found then H is sub-conformal to G.

As before we present the algorithm along with a verification of its correctness.

Gap Enlarger Algorithm

Let a be a vertex of degree δ. Initialize (C_0, A_0, B_0) to the separation triple that has C_0 consisting of the neighbors of a. Repeat the following two steps Δ times:

Step 1. Choose a vertex $x \neq a$. Find $\kappa(x, H)$ and a corresponding separation triple. Use the triple's expansion to update (C_0, A_0, B_0). Delete x from H. If we have not yet found κ then $\kappa(x, H) > \kappa(H)$ and the minimum separators change as specified in Lemma 3.1(ii). Thus the invariant is preserved.

Step 2. Repeat the following until $\delta(H) = \delta$: Let S be the set of vertices of degree $< \delta$. (Each vertex of S has degree $\delta - 1$.) Choose a vertex $y \neq a$ that is adjacent to the fewest number of vertices of S. Find $\kappa(y,H)$ and a corresponding separation triple. Use the triple's expansion to update (C_0, A_0, B_0). Add an edge from y to each vertex $z \in S$ that is not currently adjacent to y. If we have not yet found κ then $\kappa(y,H) > \kappa(H)$ and Lemma 3.1(iii) shows the new edges do not change the minimum separators of H. Thus the invariant is preserved. $\square$

The analysis uses the value

$$\beta = \frac{n-\Delta}{\delta}.$$

Lemma 3.2 *Assume the Gap Enlarger is called with $n > \delta + \Delta$.*

(i) *The final graph H has $n(H) = n - \Delta$, $m(H) \leq m$, $\delta(H) = \delta$. Furthermore the expansion of a separation triple of H is a separation triple of G.*

(ii) *Either the Gap Enlarger finds κ, or H has gap $\geq \Delta$ and is sub-conformal to G.*

(iii) *The time is $O(\Delta n \min\{m, (n-\delta)^2\} \log_\beta n)$.*

Proof: (i) –(ii) Assume for the moment that the algorithm halts. Most details of (i) – (ii) are clear so we just check three points. $m(H) \leq m$ holds since Step 2 adds an edge incident to a vertex z only if Step 1 has deleted such an edge. $\delta(H) = \delta$ holds because vertex a belongs to H and has degree δ. If the algorithm completes without finding κ the final graph H has gap $\delta(H) - \kappa(H) = \delta - (\kappa - \Delta) \geq \Delta$.

Now we show the algorithm halts. In fact we prove that each execution of Step 2 iterates at most $\log_\beta n$ times. This is a finite number since the lemma's hypothesis makes $\beta > 1$.

If $\delta = 1$ obviously there is only 1 iteration of Step 2. Hence assume $\delta > 1$. The current graph H has at least $n - \Delta$ vertices. So the pigeonhole principle guarantees that y is adjacent to at most $|S|(\delta-1)/(n-\Delta-1)$ vertices of S. Hence every iteration of Step 2 reduces $|S|$ by a factor that is at least $(n-\Delta-1)/(\delta-1) > \beta$. This implies the desired number of iterations.

(iii) The time for Step 1 and for one iteration of Step 2 is dominated by the time for a rooted connectivity computation. As mentioned in Section 1 this is $O(n \min\{m, (n-\delta)^2\})$, implying (iii). $\square$

When $\delta \leq \sqrt{n}$ an alternate implementation of the Gap Enlarger is faster. We require a *conditional computation* of $\kappa(x)$ to return the correct value along with a corresponding separator if κ has not been found and $\kappa(x) = \kappa$. Otherwise (if κ has been found, or if $\kappa(x) > \kappa$) the computation can return any valid separator and corresponding value. Clearly the Gap Enlarger remains correct if it uses conditional rooted connectivity computations.

The alternate implementation of the Gap Enlarger works as follows. Before Step 1 find $\kappa(a)$ and a corresponding separation triple. Initialize (C_0, A_0, B_0) to this triple. Initialize R to the set of neighbors of a. If κ has not been found then Lemma 2.1(i) shows R is 1-superrich.

Now in Step 1 compute $\kappa(x,H)$ conditionally by finding the values $\kappa(x,y,H)$ specified in Lemma 3.1(i). Similarly for $\kappa(y,H)$ in Step 2. Whenever Step 1 deletes a vertex x, delete it from R also. Note that R remains 1-superrich if we have not found κ. Hence each conditional rooted connectivity computation works correctly.

Lemma 3.3 *Assume the alternate implementation of the Gap Enlarger is called with $n \geq 2\Delta$ and $\sqrt{n} \geq \delta > 2$.*

(i) *Lemma 3.2*(i)–(ii) *continue to hold.*

(ii) *The time is $O((n + \delta^2\Delta)m)$.*

Proof: (i) The assumption $n > \delta + \Delta$ of Lemma 3.2 holds since $\delta + \Delta \leq \sqrt{n} + n/2 < n$.

(ii) Before Step 1 we find $\kappa(a)$ in time $O(nm)$ [10]. In Steps 1 and 2 every max flow computation takes time $O(\delta m)$ since we can stop when a flow of value δ has been found. It remains only to show there are $O(\Delta\delta)$ max flow computations.

As shown for Lemma 3.2 each execution of Step 2 iterates at most $\log_\beta n$ times. This quantity is $O(1)$ since $\beta = (n-\Delta)/\delta \geq (n/2)/\sqrt{n} = \sqrt{n}/2$. Hence there are $O(\Delta)$ conditional rooted connectivity computations. Each one consists of $|R| \leq \delta$ max flow computations, as desired. $\square$

4 Digraphs and Gap Enlargement

This section reviews our terminology for digraphs and presents the results corresponding to Section 3 for gap enlargement on digraphs.

For a digraph we use superscripts $^-$ and $^+$ to refer to incoming and outgoing edges respectively. For instance δ^+ is the smallest out-degree of a vertex, $\kappa^-(x) = \min\{\kappa(y,x) : y \in V\}$, etc. Define $\delta = \min\{\delta^+, \delta^-\}$, $\kappa(x) = \min\{\kappa^+(x), \kappa^-(x)\}$. (In contrast $\kappa(x,y)$ retains its usual meaning, distinct from $\kappa(y,x)$.) An *out-neighbor* (*in-neighbor*) of a set of vertices X is a vertex $y \notin X$ on an edge (x,y) ((y,x)) for some $x \in X$, respectively. All other terminology defined in Section 1 remains unchanged, and the time bounds for computing $\kappa(x,y)$ and $\kappa(x)$ are still valid. In addition if (S,X,Y) is a κ-separation triple, X is a κ^+*-shore* and Y is a κ^-*-shore*. We adopt the convention that a κ-separation triple is always denoted by (C^*, A^*, B^*).

We now present the results in Sections 2–3 that have close directed analogs. We begin with Lemma 2.1. The notions of rich and superrich are defined as before. An analog of part (i) holds but is not useful so we only generalize (ii).

Lemma 4.1 *Any set $S \subseteq V$ with $\kappa_W(S) > \kappa$ is ρ-superrich for $\rho = \min\{\kappa_W(S), \delta+1\} - \kappa$.*

Proof: Let (C^*, A^*, B^*) be any κ-separation triple and assume $|A^* \cap S| < \rho$. Since every vertex has out-degree at least δ, $|A^*| + |C^*| > \delta$. Thus $|A^*| \geq \delta - \kappa + 1 \geq \rho$ and $A^* - S \neq \emptyset$. This makes $(C^* \cup (A^* \cap S), A^* - S, B^*)$ a separation triple. Hence $\kappa_W^-(S) \leq |C^* \cup (A^* \cap S)| < \kappa + \rho \leq \kappa_W(S)$, a contradiction. $\square$

Lemma 2.4 applies to digraphs without change. We turn to the directed version of Lemma 3.1. This lemma will be applied in both the Gap Enlarger of this section and the Gap Algorithm of the next section. The new lemma also contains some related facts that are used in the Gap Algorithm.

We first extend the κ_W notation as follows. For $S \subseteq V$ and $x \in V$, $\kappa_W(S,x) = \min\{|S|, |C| : (C,A,B)$ is a separation triple of G with $S \subseteq A \cup C,\ x \in B\}$. (If $x \in S$ then $\kappa_W(S,x) = |S|$.) To compute $\kappa_W(S,x)$ when $x \notin S$ first form the graph $G_W(S)$ as described in Section 1 (add a vertex s to G along with directed edges (s,v), $v \in S$). Then $\kappa_W(S,x) = \kappa(s,x,G_W(S))$. $\kappa_W(x,S)$ is defined and computed analogously.

Lemma 4.2 (i) *Let $R \subseteq V$ be 1-superrich. Any vertex x with $\kappa^+(x) = \kappa$ has $\kappa = \min\{\kappa(x,y) : y \in R\}$. Similarly any set of vertices S with $\kappa_W^+(S) = \kappa$ has $\kappa = \min\{\kappa_W(S,y) : y \in R\}$.*

(ii) *For any vertex x with $\kappa(x) > \kappa$, $G - x$ is sub-conformal to G.*

(iii) *For any vertices x,y with $\kappa(x,y) > \kappa$, $G + (x,y)$ is conformal to G.*

(iv) *For any vertex x and any set of vertices S with $\kappa^+(x) < \kappa_W^+(S)$, $\kappa^+(x) = \min\{\kappa(x,y) : y \in S\}$.*

(v) *For any set of vertices S, any separation triple (C,A,B) and any vertex $b \in B - S$, $\kappa_W(S,b) \leq |C| + |S \cap B|$.*

Proof: (iv) Let (C, A, B) be a separation triple corresponding to $\kappa^+(x)$. If B and S are disjoint then $S \subseteq A \cup C$ and so $\kappa^+_W(S) \le |C| = \kappa^+(x)$, a contradiction. Thus $B \cap S \ne \emptyset$, which gives (iv).

(v) The hypothesis makes $(C \cup (S \cap B), A, B - S)$ a separation triple. This implies (v). □

Now we show that the algorithms of Section 3 for gap enlargement extend naturally to digraphs, with the same time bound.

The Gap Enlarger Algorithm works as before with minor changes to Step 2: S is defined to be the multiset of vertices whose in-degree or out-degree is $< \delta$; a vertex is included twice if both degrees are $< \delta$. The remaining changes are to use the natural interpretation of how edges are directed. Lemma 3.2 and its proof apply unmodified.

The alternate implementation of the Gap Enlarger for $\delta \le \sqrt{n}$ is changed to use the following procedure before Step 1:

Let a be a vertex with in-degree or out-degree δ. Let R be a set of $\delta + 1$ vertices, specifically vertex a and its δ in-neighbors or out-neighbors. Find $\kappa_W(R)$ and a corresponding separation triple. (Obviously $\kappa_W(R) \le \delta$.) Initialize (C_0, A_0, B_0) to this triple. Observe that if a κ-separator has not been found then Lemma 4.1 shows that R is 1-superrich.

The rest of the algorithm is unmodified. Lemma 3.3 holds as before.

5 Digraph Gap Algorithm

Our nesting property for directed separation triples is weaker than its undirected counterpart Lemma 2.3. This leads to a more involved directed Gap Algorithm. In addition the directed Gap Algorithm needs to be more general (i.e., handle low connectivities) since Henzinger's algorithm [9] applies only to undirected graphs. Nonetheless the same asymptotic running time can be achieved for the directed Gap Algorithm. This section begins with the directed nesting property. Then two versions of the Gap Algorithm are presented, the first more efficient for large connectivities and the second more efficient for small.

In a separation triple (C, A, B), B is *in-extreme* if each of its nonempty subsets has at least $|C|$ in-neighbors; similarly for A *out-extreme.* As examples, a separation triple (C, A, B) has B in-extreme if it is a κ-separation triple, or it is a separation triple corresponding to $\kappa^+(x)$ (i.e, $x \in A$ and $|C| = \kappa^+(x)$) or it is a separation triple corresponding to $\kappa^+_W(S)$ (i.e, $S \subseteq A \cup C$ and $|C| = \kappa^+_W(S)$).

Lemma 5.1 *Let (C, A, B) be a separation triple with B in-extreme. Either*

(i) *A contains a κ^+-shore, or*

(ii) *for every κ-separation triple (C^*, A^*, B^*) either $A \cap A^*$ or $B \cap B^*$ is empty. Furthermore $A \cap A^* \ne \emptyset$ implies $|(B \cup C) \cap A^*| \ge |B|$, and symmetrically $B \cap B^* \ne \emptyset$ implies $|(A \cup C) \cap B^*| \ge |A|$.*

Proof: We begin by considering two arbitrary separation triples (C, A, B) and (C', A', B') (see Fig.1). First observe that the out-neighbors of $A \cap A'$ are contained in $(A' \cap C) \cup (C' - B)$. In proof since no edge goes from A' to B', any out-neighbor of $A \cap A'$ belongs to A' or C'. Since no edge goes from A to B, any out-neighbor actually belongs to $A' \cap C$ or $C' - B$, as desired. Next observe that if $A \cap A'$ is nonempty its out-neighbors form a separator. This follows since no vertex of B is an out-neighbor.

Now let (C^*, A^*, B^*) be a κ-separation triple with $A \cap A^* \ne \emptyset$ and $B \cap B^* = \emptyset$. The previous remarks show that $|A^* \cap C| + |C^* - B| \ge |C^*|$, or equivalently $|A^* \cap C| \ge |C^* \cap B|$. Thus $|(B \cup C) \cap A^*| = |B \cap A^*| + |C \cap A^*| \ge |B \cap A^*| + |C^* \cap B| = |B|$. Hence $A \cap A^* \ne \emptyset$ and $B \cap B^* = \emptyset$ implies $|(B \cup C) \cap A^*| \ge |B|$ as in the second part of (ii). Applying this assertion to the reverse graph we get that $B \cap B^* \neq \emptyset$ and $A \cap A^* = \emptyset$ implies $|(A \cup C) \cap B^*| \geq |A|$, as in the second part of (ii).

Finally assume the hypothesis of the lemma, i.e., (C, A, B) is a separation triple with B in-extreme. For any κ-separation triple (C^*, A^*, B^*), if $B \cap B^*$ is nonempty it has at least $|C|$ in-neighbors, by extremeness. This implies $|B^* \cap C| + |C^* - A| \geq |C|$, or equivalently $|C^* \cap B| \geq |C \cap A^*|$. Thus the number of out-neighbors of $A \cap A^*$ is at most $|A^* \cap C| + |C^* - B| \leq |C^* \cap B| + |C^* - B| = |C^*|$. We conclude that $A \cap A^*, B \cap B^* \neq \emptyset$ implies $A \cap A^*$ is a κ^+-shore. So if (i) of the lemma does not hold, the first part of (ii) holds. The second part of (ii) now follows from the previous paragraph. □

We turn to the directed Gap Algorithm. As before the algorithm checks k-connectedness for a given k. Define

$$D = \min\{2k, \delta\},\ \gamma = D - k,\ \tau = \gamma/2.$$

Assume $\gamma \geq 4$. The comments verifying correctness assume that $\kappa < k$.

As in the Gap Enlarger we modify the given graph to a new graph H. Initially H is G, and H always denotes the current graph. H is constructed from G by adding edges, so a separation triple of H is a separation triple of G. The algorithm maintains these invariants:

I1. If κ has not been found then H is conformal to G and for every κ-separation triple (C^*, A^*, B^*), $X \cap B^* = Y \cap A^* = \emptyset$.

I2. After each iteration of Step 3, every vertex of X (Y) has in-degree (out-degree) $n - 1$, respectively.

Directed Gap Algorithm

Step 1. If $n < 2D$ then set R to V and go to Step 4. Note that V is τ-rich since a κ-shore has $\geq \delta + 1 - \kappa > \delta - k \geq 2\tau$ vertices.

Step 2. Let S be a set of $2D$ vertices. Initialize (C_0, A_0, B_0) to a separation triple with $|C_0| = \delta$ (use a vertex of in- or out-degree δ). Initialize sets X and Y to $\emptyset$.

Step 3. Repeat Steps 3.1–3.2 until Step 3.2 either halts or goes on to Step 4:

Step 3.1. Find $\kappa_W(S, H)$ and a corresponding separation triple (C, A, B) if one exists. In the latter case update (C_0, A_0, B_0).

Step 3.2. If $\kappa_W(S, H) \geq k + \tau$ then let R be the set S and go to Step 4. Note that R is a τ-rich set of $2D$ vertices, by Lemma 4.1 with $\rho = \min\{\kappa_W(S, H), \delta(H) + 1\} - \kappa \geq (k + \tau) - k = \tau$. (Since the algorithm never deletes edges, $\delta(H) \geq \delta$.)

The other possibility is $\kappa_W(S, H) < k + \tau$. We first discuss the case $\kappa_W(S, H) = \kappa_W^+(S, H)$ and then sketch the symmetric case $\kappa_W(S, H) = \kappa_W^-(S, H)$.

If $\kappa_W(S, H) = \kappa_W^+(S, H)$ then $S \subseteq A \cup C$ and B is in-extreme. We will apply Lemma 5.1 to (C, A, B). Compute $\kappa_W(A, H)$ and $\kappa_W(C, H)$ with corresponding separation triples and update (C_0, A_0, B_0). Add B to X. If this makes $|X| > k$ then compute $\kappa_W(X, H)$ and a corresponding separation triple, update (C_0, A_0, B_0) and halt. Otherwise add edge (x, y) to H for every $x \in V$, $y \in B$ where this edge does not exist.

Observe that if A contains a κ^+-shore then $\kappa = \kappa_W(C, H)$ (note that $|C| > \kappa$ if κ has not been found). Hence we can assume Lemma 5.1(ii) applies. If a κ-separation triple (C^*, A^*, B^*) has $A \cap A^* = \emptyset$ then $\kappa_W^-(A, H) = \kappa$, since $|A| \geq |S| - |C| > 2D - k - \tau > D > \kappa$. Hence assume that $A \cap A^* \neq \emptyset$ and, by Lemma 5.1(ii), $B \cap B^* = \emptyset$, for every κ-separation triple (C^*, A^*, B^*). Now adding B to X preserves invariant I1. If $|X| > k$ invariant I1 shows we find κ as $\kappa^+_W(X, H)$. If $|X| \le k$ Lemma 4.2(iii) shows that the new edges (x, y) preserve invariant I1, as well as I2.

If $\kappa_W(S, H) = \kappa^-_W(S, H)$ proceed symmetrically, adding A to Y and adding edges directed from the vertices of Y. In both cases after adding the edges, go back to Step 3.1 to start the next repetition.

Step 4. If κ has not been found and $\kappa < k$ then R is now τ-rich and either $R = V$ with $n < 2D$, or $|R| = 2D$. Apply Lemma 2.4, constructing the expander R_d and computing the values $\kappa(x, y)$ specified in the lemma. Update (C_0, A_0, B_0) accordingly. At this point if $\kappa < k$ the algorithm has found κ. □

For the efficiency analysis we show that Steps 3.1–3.2 are executed $O(k/\tau)$ times. It suffices to prove that each execution with $\kappa_W(S, H) < k + \tau$ increases $|X| + |Y|$ by at least τ, since this implies there are at most $2k/\tau$ such executions.

To prove this assume $\kappa_W(S, H) = \kappa^+_W(S, H)$, the case $\kappa_W(S, H) = \kappa^-_W(S, H)$ is symmetric. Invariant I2 shows that right after Step 3.1, $X \cap B = \emptyset$. (In fact this holds for any separation triple (C, A, B).) Since every degree is $\ge \delta$, $|B| \ge \delta + 1 - |C| > (k + \gamma) - (k + \tau) = \tau$. Thus adding B to X increases $|X|$ by at least τ, as desired.

Next observe that $m(H) = O(m)$. This follows since Step 3.2 only adds edges when $|X| \le k$ (or $|Y| \le k$). Hence $O(kn)$ edges are added to H. This quantity is $O(m)$ since $m \ge \delta n$ and $\delta > k$.

Now we show the total time for Step 3 is $O(kn + kn \min\{m, (n - \delta)^2\}/\tau)$. An execution of Steps 3.1–3.2 performs $O(1)$ rooted connectivity computations. As mentioned in Section 1 each such computation uses time $O(n \min\{m, (n - \delta)^2\})$, so this is within the bound. The remaining time in Step 3 is for adding edges, and it is easy to see this can be done in total time $O(kn)$.

The next lemma gives four time bounds for the algorithm, similar to Lemma 2.5.

Lemma 5.2 *If $\gamma \ge 4$ then the Digraph Gap Algorithm either verifies that $\kappa \ge k$ or finds a κ-separator.*

(i) *If $k \ge \sqrt{n}/2$ and $n \ge 2D$ the time is $O\left(\frac{D^2}{\gamma}\sqrt{n}m\right)$.*

(ii) *If $k \ge \sqrt{n}\log n$ and $n \ge 2D$ the time is $O\left(\frac{D^2\sqrt{n}m}{\gamma \log_{4n/\delta} n}\right)$.*

(iii) *In general the time is $O\left(\frac{n^{9/2}}{\gamma \log(n-\delta)}\right)$.*

(iv) *If $\gamma \ge \epsilon D$ for some fixed $\epsilon > 0$ the time is $O((\sqrt{n} + D)\sqrt{n}m)$.*

Proof: For each part we first verify that the time for Step 3 is within the desired bound. Then we use an argument similar to Lemma 2.5 to estimate the time for Step 4.

(i) The assumption $k \ge \sqrt{n}/2$, with $D \ge k$, implies $kn = O(Dn) = O(D^2\sqrt{n})$. Hence the time for Step 3 is $O(knm/\gamma) = O(D^2\sqrt{n}m/\gamma)$. For Step 4 it suffices to show there are $O(D^2/\gamma)$ max flow computations. Since $r = 2D \ge D + k$ Lemma 2.4 gives $d = O(D^2/(\tau D)) = O(D/\gamma)$. Just as in Lemma 2.5 Step 4 performs $O(dr)$ max flow computations, as desired.

(ii) The assumption on k implies $D \ge \sqrt{n}\log n$. Similar to part (i) we get $O(knm/\gamma) = O(D^2\sqrt{n}m/(\gamma \log n))$, which is within the desired time bound. For Step 4 we combine the estimate of $O(D^2/\gamma)$ max flow computations of part (i) with the time bound of Lemma 2.5(ii) for one max flow. (This time bound is valid for digraphs.)

(iii) Step 3 uses time $O(kn + kn(n-\delta)^2/\tau) = O(n^4/\gamma)$. This is within the desired bound since $\sqrt{n} \geq \log(n-\delta)$. The time for Step 4 is estimated exactly as in Lemma 2.5(iii). (The estimate is valid for digraphs.)

(iv) By assumption $\gamma \geq \epsilon D \geq \epsilon k$, so Step 3 uses time $O(knm/\gamma) = O(nm)$. Step 4 does $O(D)$ max flow computations since in Lemma 2.4 we have $r \leq 2D$ and $d = O(1)$. The latter holds because $\tau \geq \epsilon D/2$ implies $d = O(r^2/\rho^2) = O(D^2/\tau^2) = O(1)$. □

Now we present a version of the Directed Gap Algorithm that is more efficient when $k \leq \sqrt{n}$. For definiteness we refer to the original Directed Gap Algorithm as the basic algorithm. The idea is to use Lemma 4.2(iv) to replace the rooted connectivity computations in Step 3 of the basic algorithm by max flow computations. But in order to satisfy the hypothesis of Lemma 4.2(iv) we need to separate out a subcase of Step 3. (This is done in Step 2 below.)

As in the alternate Gap Enlarger we use conditional computations of $\kappa(x)$. We also use conditional computations of $\kappa_W(S)$, defined similarly. (If κ has been found or $\kappa_W(S) > \kappa$ the computation can return any valid separator and corresponding value.) We use Lemma 4.2(i) to do conditional computations. Clearly any algorithm that does some of its computations conditionally still maintains (C_0, A_0, B_0) as a valid separator and finds κ, if at some point it conditionally computes a value $\kappa(x)$ or $\kappa_W(S)$ that actually equals κ.

All assumptions of the basic Directed Gap Algorithm remain unchanged. Invariants I1–I2 still hold. In addition we assume $n \geq 4D$ for convenience.

Directed Gap Algorithm, Flow Version

Step 1. Let S be an arbitrary set of $2D$ vertices. Initialize (C_0, A_0, B_0) to a separation triple with $|C_0| = \delta$ (use a vertex of in- or out-degree δ). Initialize sets X and Y to $\emptyset$.

Find $\kappa_W(S)$. If a corresponding separation triple exists, update (C_0, A_0, B_0). From now on we can assume that S is 1-superrich (else κ has been found). We will use S in all conditional connectivity computations (i.e., in applying Lemma 4.2(i) take R to be S).

If $\kappa^+_W(S)$ and $\kappa^-_W(S)$ are both $< k + \tau$ go to Step 2, else go to Step 3.

Step 2. Now $\kappa^+_W(S)$, $\kappa^-_W(S) < k + \tau$. Let $\kappa^+_W(S)$ ($\kappa^-_W(S)$) correspond to the separation triple (C^+, A^+, B^+) ((C^-, A^-, B^-)) respectively. Conditionally compute $\kappa_W(T)$ for $T = A^+$, B^+, C^+, A^-, B^-, C^- and update (C_0, A_0, B_0) for the separation triples that are found. Then set $R = (B^+ \cup C^+) \cup (A^- \cup C^-)$ and go to Step 4.

Observe that $|A^+| \geq |S| - |C^+| > 2D - (k + \tau) > D > \kappa$. Hence if $A^+ \cap A^* = \emptyset$ for some κ-separation triple (C^*, A^*, B^*) then $\kappa^-_W(A^+) = \kappa$. Thus we can assume $A^+ \cap A^* \neq \emptyset$ for every κ-separation triple (C^*, A^*, B^*). Apply Lemma 5.1 to the separation triple (C^+, A^+, B^+) with B^+ in-extreme. If part (i) of Lemma 5.1 holds then $\kappa^-_W(C^+) = \kappa$. So assume part (ii) holds. We claim

$$|(B^+ \cup C^+) \cap A^*| \geq \tau.$$

In proof part (ii) shows the left-hand side is $\geq |B^+|$, so it suffices to show $|B^+| \geq \tau$. A vertex of B^+ has $\geq \delta \geq k + 2\tau$ in-neighbors, all belonging to $B^+ \cup C^+$. Now $|C^+| = \kappa^+_W(S) < k + \tau$ implies the desired inequality.

Part (ii) also implies $B^+ \cap B^* = \emptyset$ for every κ-separation triple (C^*, A^*, B^*). Hence if $|B^+| \geq k$ then $\kappa^+_W(B^+) = \kappa$. So assume

$$|B^+| < k.$$

A symmetric argument shows we can assume the analogs of the displayed inequalities for separation triple (C^-, A^-, B^-), specifically $|(A^- \cup C^-) \cap B^*| \geq \tau$ and $|A^-| < k$. Together these 4 inequalities show that the set R is a τ-rich set of $\leq 2(2k+\tau) \leq 4D$ vertices.

Step 3. At this point one or both $\kappa_W^+(S)$ and $\kappa_W^-(S)$ is $\geq k+\tau$. Assume $\kappa_W^-(S) \geq k+\tau$. If only $\kappa_W^+(S) \geq k+\tau$ then proceed in a symmetric manner, applying what follows to the reverse graph. (If both $\kappa_W^+(S), \kappa_W^-(S) \geq k+\tau$ we could go directly to Step 4 as in the basic Directed Gap Algorithm.) Recall that Step 3 will preserve invariants I1–I2.

Initialize the set T to S, sets X and Y to $\emptyset$, and graph H to G. Repeat Steps 3.1–3.2 until they halt or go on to Step 4:

Step 3.1. If $T = \emptyset$ then set R to S and go to Step 4. Otherwise let b be a vertex of T. If $\kappa_W(S-b, b, H) \geq k+\tau$ then delete b from T and repeat Step 3.1. Otherwise go to Step 3.2.

Let us show that if T becomes empty then we go to Step 4 with $R = S$ a τ-rich set. We first show

$$|S \cap A^*| \geq \tau$$

for every κ-separation triple (C^*, A^*, B^*). If not then $\kappa_W^-(S) \leq \kappa + |S \cap A^*| < k+\tau$ (as in Lemma 4.1) contradicting the initial assumption of Step 3. Next we show that if T becomes empty then

$$|S \cap B^*| \geq \tau.$$

Suppose the opposite inequality holds for some κ-separation triple (C^*, A^*, B^*). Initially T contains a vertex b of B^*, since S is 1-superrich. If b is ever chosen in Step 3.1 then Lemma 4.2(v) shows $\kappa_W(S-b, b, H) \leq \kappa + |(S-b) \cap B^*| < k+\tau$. Hence T never becomes empty.

Step 3.2. We have $\kappa_W(S-b, b, H) < k+\tau$. Apply Lemma 4.2(iv) to calculate $\kappa^-(b, H)$ as $\min\{\kappa(y, b, H) : y \in S\}$. The hypothesis of Lemma 4.2(iv) holds since $\kappa^-(b, H) \leq \kappa_W(S-b, b, H) < k+\tau \leq \kappa_W^-(S) \leq \kappa_W^-(S, H)$. (The last inequality holds since H is the result of adding edges to G.) Let (C, A, B) be a separation triple corresponding to $\kappa^-(b, H)$.

Apply Lemma 5.1 to (C, A, B) with A out-extreme as follows. Conditionally compute $\kappa_W(C, H)$ using Lemma 4.2(i) and update (C_0, A_0, B_0). We can now assume part (i) of Lemma 5.1 does not hold, so part (ii) does. At least one of the sets A, B has size $\geq k$, since $n \geq 4D \geq 3D + k + \tau > 2k + |C|$.

First suppose $|A| \geq k$. Use Lemma 4.2(i) to conditionally compute $\kappa_W(A, H)$ and update (C_0, A_0, B_0). So we can now assume $\kappa_W(A, H) > \kappa$. Thus every κ-separation triple (C^*, A^*, B^*) has $A \cap A^* \neq \emptyset$. Lemma 5.1(ii) implies $B \cap B^* = \emptyset$. Add B to X. If this makes $|X| > k$ then conditionally compute $\kappa_W(X, H)$ and a corresponding separation triple, update (C_0, A_0, B_0) and halt. Otherwise add edge (x, y) to H for every $x \in V$, $y \in B$ where this edge does not exist.

Note the additions to X and H preserve I1–I2. Also when $|X| > k$ the algorithm halts having found κ, by I1.

Next suppose $|B| \geq k$. Follow a symmetric procedure (adding A to Y and adding edge (x, y) for every $x \in A$, $y \in V$). In both cases after adding the edges, go back to Step 3.1 to start another iteration.

Step 4. If κ has not been found and $\kappa < k$ then R is now τ-rich with at most $4D$ vertices. Add vertices to make $|R| = 4D$. Apply Lemma 2.4, constructing the expander R_d and computing the values $\kappa(x, y)$ specified in the lemma. Update (C_0, A_0, B_0) accordingly. At this point if $\kappa < k$ the algorithm has found κ. □

Lemma 5.3 *If $\gamma \geq 4$ and $n \geq 4D$ then the Flow Version of the Directed Gap Algorithm either verifies that $\kappa \geq k$ or finds a κ-separator. The time is $O((n + D^2k/\gamma)m)$.*

Proof: We have verified the Flow Version is correct so we need only estimate the time. We use rooted connectivity computations to find $\kappa^+_W(S)$ and $\kappa^-_W(S)$ in time $O(nm)$. All other connectivity quantities are found using max flow computations (because of conditional computations). We show that the algorithm performs $O(D^2/\gamma)$ max flow computations on graphs with $O(m)$ edges. This implies the desired bound. Step 2 performs $O(|S|) = O(D)$ max flow computations, which is within the desired bound.

Consider Step 3. At most $|S|$ executions of Step 3.1 delete a vertex from T, and every other execution is followed by an execution of Step 3.2. There are $O(k/\tau) = O(D/\gamma)$ executions of Step 3.2 by the analysis of the basic Directed Gap Algorithm. That analysis also shows $m(H) = O(m)$. Each execution of Step 3.1 computes one max flow and each execution of Step 3.2 computes $O(D)$ max flows. This gives $O(D + D^2/\gamma)$ max flow computations total for Step 3.

Finally consider Step 4. Since $r = 4D$ the value d of Lemma 2.4 is $O(D^2/(\tau D)) = O(D/\gamma)$. Hence it performs $O(D^2/\gamma)$ max flow computations. $\Box$

We complete this section by showing how to check $a\delta$-connectedness, for any fixed constant $a < 1$. We first present an algorithm that checks $\delta/2$-connectedness: If $\delta < 8$ or $n < 16$ use the algorithm of [10] to find κ in time $O(nm)$. Otherwise check k-connectedness for k starting at $\min\{\delta/2, \sqrt{n}\}$ and doubling, until either κ is found or we verify that $\kappa \geq \delta/2$. Perform these connectivity checks with the basic Directed Gap Algorithm.

To analyze the time, we can assume $k \leq \delta/2$ always holds. This implies $D = 2k$ and $\gamma = k$. The hypothesis $\gamma \geq 4$ of the Gap Algorithm holds, by the first step. Hence Lemma 5.2(iv) shows the Gap Algorithm uses time $O((\sqrt{n} + k)\sqrt{n}m)$. If k initially equals $\delta/2$ then the Gap Algorithm is executed once and the total time is $O(nm)$. If k initially equals $\sqrt{n}$ then, regardless of whether we find κ or not, the largest value of k is $O(\kappa)$. So summing over all k gives total time $O(\kappa\sqrt{n}m)$.

Now suppose we wish to check $a\delta$-connectedness for some fixed a, $1/2 < a < 1$. We can assume $\kappa \geq \delta/2$ else the previous algorithm finds κ. We can also assume the gap $\gamma = (1 - a)\delta \geq 4$. If not then $\delta = O(1)$ and the algorithm of [10] computes κ in time $O(nm)$. Now apply the basic Directed Gap Algorithm. Since $\gamma = (1 - a)\delta$ the time is given by Lemma 5.2(iv).

For undirected graphs a similar algorithm applies. First a preprocessing step reduces the number of edges to $O(\delta n)$ by finding a maximal forest decomposition and replacing the given graph by the first $\delta + 1$ forests. The new graph has the same minimum separators and has $O(\delta n)$ edges. Using the algorithm of Nagamochi and Ibaraki the time is $O(m)$ [15]. We check $\delta/2$-connectedness using Henzinger's algorithm, which runs in time $O(\min\{\kappa, \sqrt{n}\}n^2)$ [9]. When $a > 1/2$ we proceed as above. If desired the undirected Gap Algorithm can be used here, since it also satisfies the time bound of Lemma 5.2(iv).

Lemma 5.4 *For any fixed $a < 1$ we can check $a\delta$-connectedness in time $O((\kappa + \sqrt{n})\sqrt{n}m)$. In an undirected graph the same bound holds with m replaced by κn. The space is $O(m)$.* $\Box$

When $\kappa > n^{2/5}$ this bound is superior to our general bound to find κ. (In the opposite case the two bounds are the same.)

For completeness we include an alternate algorithm for checking $\delta/2$-connectedness in undirected graphs, in time bound $O(\kappa n^2)$. For any $k \leq \delta/2$ we show how to check k-connectedness in time $O(kn^2)$. Using this procedure with k doubling gives our algorithm for checking $\delta/2$-connectedness.

Apply the algorithm of [15] to get a maximal forest decomposition of the given graph. Let $S \subseteq V$ consist of the last $\delta - k$ vertices to be labelled. Let H be the graph consisting of the first k forests. Compute $\kappa_W(S, H)$. If this quantity is strictly less than k then it equals κ. Otherwise $\kappa \geq k$.

This algorithm is correct because of two properties. First, $\kappa < k$ implies $\kappa_W(S, G) = \kappa$. This follows since any two vertices $x, y \in S$ have $\kappa(x, y) \geq k$ [9] and $|S| = \delta - k \geq k$. Second G and H have the same separators of $< k$ vertices [15] so we can compute $\kappa_W(S, H)$ rather than $\kappa_W(S, G)$. The time is $O(kn^2)$ using [10] and the fact that H has $O(kn)$ edges.

6 Connectivity Algorithm

This section completes the algorithm to find the connectivity of a graph. The time for digraphs is $O((n + \min\{\kappa^{5/2}, \kappa n^{3/4}\})m)$. It may be helpful to note the equivalent bound, $O((n + \kappa \min\{\kappa^{3/2}, n^{3/4}\})m)$. For undirected graphs we can replace m by κn in both bounds.

The connectivity algorithm begins by treating some simple cases. First use the algorithm of Lemma 5.4 (or [9]) to either find κ or verify that $\kappa \geq \delta/2$. The time is within the desired bound. Assume that $\delta \geq 16$, since in the opposite case the algorithm of [10] computes κ in time $O(nm)$. If $\delta \geq n - n^{7/8}$ find κ using the algorithm of [10]. As mentioned in Section 1 it runs in time $O(\kappa n(n-\delta)^2) = O(n^{15/4})$.

Define a parameter Δ for gap enlargement by first setting

$$\Delta_0 = \begin{cases} \sqrt{\delta} & \delta \leq \sqrt{n} \\ \delta/n^{1/4} & \sqrt{n} < \delta \leq n^{2/3} \\ \frac{\delta}{n^{1/4} \log_{n/\delta} n} & n^{2/3} < \delta \leq n/3 \\ \frac{n^{3/4}}{\log n} & n/3 < \delta < n - n^{7/8} \end{cases}$$

and defining $\Delta = \max\{4, \lfloor \Delta_0 \rfloor\}$. Execute the Gap Enlarger with this value of Δ to either find κ or get the graph H with gap $\geq \Delta$. (If $\delta \leq \sqrt{n}$ use the alternate implementation of the Gap Enlarger.) Execute the Gap Algorithm on H with $k = \delta - \Delta$. (Recall $\delta(H) = \delta$. If the graph is directed use the Flow Version of the Directed Gap Algorithm for $\delta \leq \sqrt{n}$ and use the basic version otherwise.) Compare the two separation triples of G, (C_0, A_0, B_0) found by the Gap Enlarger and the expansion of the separation triple for $\kappa(H)$ found by the Gap Algorithm. (The latter need not exist if $\kappa(H) \geq k$.) The smaller separator is the desired κ-separator for the given graph G.

To show a κ-separator is found, suppose the Gap Enlarger does not find κ. Since the Gap Enlarger initializes (C_0, A_0, B_0) to a separator of δ vertices, $\kappa < \delta$. Thus $\kappa(H) < k$, the Gap Algorithm finds $\kappa(H)$, $\kappa = \kappa(H) + \Delta$ and the expanded separator is the desired κ-separator.

Remark: In the definition of Δ_0 the exponent 2/3 can be replaced by any exponent strictly between 1/2 and 1. For undirected graphs the second case ($\Delta_0 = \delta/n^{1/4}$) can be eliminated and incorporated into the third case.

We analyze the efficiency of this algorithm in the next two lemmas. First note that in the Gap Algorithm the hypothesis $\gamma \geq 4$ (of Lemmas 2.5, 5.2 and 5.3) always holds since $\gamma = \Delta$.

Lemma 6.1 *When $\delta \leq \sqrt{n}$ the algorithm finds κ and a κ-separator in time $O((n + \kappa^{5/2})m)$.*

Proof: Assume $\delta \geq 16$ by the preliminary processing. As a result $\Delta = \lfloor \sqrt{\delta} \rfloor$.

Consider the Gap Enlarger. To show the requirements of Lemma 3.3 hold, we have $n \geq 2\Delta$ since $n \geq \delta^2 \geq 16\delta \geq 16\Delta$. The second requirement $\sqrt{n} \geq \delta > 2$ is obvious. Hence the Gap

Enlarger works correctly and its time is given by Lemma 3.3(ii), i.e.,

$$O((n + \delta^2 \Delta)m).$$

Now consider the Gap Algorithm executed on H. We will apply Lemma 5.3 (2.5(i)). We need the hypothesis $n(H) \geq 4\delta(H)$. This holds since $n - \Delta \geq \delta^2 - \lfloor\sqrt{\delta}\rfloor \geq 15\delta$. Applying the lemma with $\gamma = \Delta$, $n(H) \leq n$ and $m(H) \leq m$ gives time

$$O((n + \delta^3/\Delta)m).$$

Finally note that the two displayed quantities are within the time bound of the lemma because $\kappa \geq \delta/2$ and $\Delta = \lfloor\sqrt{\delta}\rfloor \geq \sqrt{\delta}/2$. □

Lemma 6.2 *When $\delta \geq \sqrt{n}$ we find κ and a κ-separator in time $O(\kappa n^{3/4} m)$.*

Remark: The analysis is complicated by two phenomena which emerge as δ becomes large. First, the factor $\log_\beta n$ in the time for gap enlargement (Lemma 3.2(iii)) is $O(1)$ for small δ, e.g., $n \geq 3\delta^e$ for any fixed $e > 1$. But this factor grows for larger δ, e.g., it is about $n^{1/4} \log n$ when $\delta = n - n^{7/8}$. Second, the degree of the expander graph is $O(\delta/\gamma)$ when $n \geq 2\delta$ (Lemma 5.2(i)). When $n = 2\delta$ this is $O(n/\gamma)$. For larger values of δ the degree bound is a factor $n/(n-\delta)$ larger (Lemma 5.2(iii)). So when $\delta = n - n^{7/8}$ this is a factor $n^{1/8}$ larger.

Proof: We consider the same three cases as in the definition of Δ. Observe that we always have

$$\Delta \leq n^{3/4}.$$

Case 1. $\delta \leq n^{2/3}$.

Consider the Gap Enlarger. The requirement $n > \delta + \Delta$ of Lemma 3.2 is satisfied since $n \geq \delta^{3/2} \geq 4\delta \geq \delta + \Delta$. Hence the Gap Enlarger works correctly. To compute the time note that $\beta = (n - \Delta)/\delta \geq (n - n^{3/4})/\delta \geq (n/2)/n^{2/3} \geq n^{1/3}/2$. Hence Lemma 3.2(iii) shows the time is

$$O(\Delta n m \log_\beta n) = O\left(\frac{\delta}{n^{1/4}} nm\right) = O(\kappa n^{3/4} m).$$

To estimate the time for the Gap Algorithm observe that $n(H) = n - \Delta \geq n - n^{3/4} \geq n/2 \geq \delta^{3/2}/2 \geq 2\delta$. Also $k = \delta - \Delta \geq \delta/2 \geq \sqrt{n}/2 \geq \sqrt{n(H)}/2$. Hence part (i) of Lemma 5.2 (2.5) applies. Using $n(H) \leq n$, $m(H) \leq m$ we get time

$$O\left(\frac{\delta^2}{\Delta}\sqrt{n} m\right) = O\left(\frac{\delta^2}{\delta/n^{1/4}}\sqrt{n} m\right) = O(\kappa n^{3/4} m).$$

Case 2. $n^{2/3} < \delta \leq n/3$.

We will use several times the fact that for any constant $c > 1/3$, $\log(cn/\delta) = \Theta(\log(n/\delta))$.

Consider the Gap Enlarger. To show the requirement of Lemma 3.2, $n > \delta + \Delta$, observe that $n \geq 3\delta > 2\delta + \Delta$. Hence the Gap Enlarger works correctly. To compute the time note that $\beta = (n - \Delta)/\delta \geq n/(2\delta)$ since $\Delta \leq \delta \leq n/2$. So Lemma 3.2(iii) shows the time is

$$O(\Delta n m \log_\beta n) = O\left(\frac{\delta}{n^{1/4} \log_{n/\delta} n} nm \log_{n/2\delta} n\right) = O(\kappa n^{3/4} m).$$

To estimate the time for the Gap Algorithm observe that $n(H) = n - \Delta \geq 3\delta - \delta = 2\delta$. Also $k = \delta - \Delta \geq \delta/2 \geq n^{2/3}/2 \geq \sqrt{n(H)}\log n$. Hence part (ii) of Lemma 5.2 (2.5) applies. Now using $n(H) \leq n$, $m(H) \leq m$ gives time

$$O\left(\frac{\delta^2\sqrt{n}m}{\gamma \log_{4n/\delta} n}\right) = O\left(\frac{\kappa^2\sqrt{n}m}{\frac{\kappa}{n^{1/4}\log_{n/\delta} n}\log_{4n/\delta} n}\right) = O(\kappa n^{3/4} m).$$

Case 3. $n/3 < \delta < n - n^{7/8}$.

In this case the desired time bound is $O(n^{15/4})$, since $\kappa \geq \delta/2 \geq n/6$ and $m \geq n\delta/2 \geq n^2/6$.

Consider the Gap Enlarger. The requirement $n > \delta + \Delta$ is satisfied since $n > \delta + n^{7/8} \geq \delta + n^{3/4} \geq \delta + \Delta$. Define ϵ so that

$$\delta = (1-\epsilon)n.$$

The assumption of Case 3 implies $1/n^{1/8} \leq \epsilon < 1$. Lemma 3.2(iii) shows the time for the Gap Enlarger is

$$O(\Delta n(n-\delta)^2 \log_\beta n) = O\left(\frac{n^{7/4}(n-\delta)^2}{\log\beta}\right) = O\left(\frac{n^{15/4}\epsilon^2}{\log\beta}\right).$$

Hence it suffices to show $\ln\beta \geq \epsilon/2$. First observe that $\beta = (n-\Delta)/\delta \geq (1 - 1/n^{1/4})/(1-\epsilon)$. Hence $\ln\beta \geq -2/n^{1/4} + \epsilon$. For sufficiently large n, $\epsilon \geq 1/n^{1/8} \geq 4/n^{1/4}$. This implies $\ln\beta \geq \epsilon/2$.

We estimate the time for the Gap Algorithm using part (iii) of Lemma 5.2 (2.5), which with $n(H) \leq n$ gives time

$$O\left(\frac{n^{9/2}}{\gamma\log(n(H)-\delta)}\right) = O\left(\frac{n^{15/4}\log n}{\log(n(H)-\delta)}\right).$$

Hence it suffices to show $n(H) - \delta \geq n^{3/4}$ for sufficiently large n. This follows since $n(H) - \delta = n - \Delta - \delta \geq n^{7/8} - n^{3/4}$. □

This completes the analysis of the connectivity algorithm for digraphs. For undirected graphs we use the same algorithm with this preprocessing step: Find a maximal forest decomposition and discard all edges except those in the first $\delta + 1$ forests. The new graph has the same minimum separators as the original and has $O(\delta n)$ edges. Using the algorithm of [15] the time is $O(m)$.

Now combining Lemmas 6.1 and 6.2 gives our main result.

Theorem 6.3 *In a digraph the vertex connectivity κ and a corresponding separator can be found in $O((n + \min\{\kappa^{5/2}, \kappa n^{3/4}\})m)$ time. In an undirected graph the same bound holds with m replaced by κn. The space in both cases is $O(m)$.*

Acknowledgments

We thank Noga Alon, Avi Wigderson and David Zuckerman for help with expander graphs.

References

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

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[4] N. Alon, M. Blum, A. Fiat, S. Kannan, M. Naor, and R. Ostrovsky. Matching nuts and bolts. In *Proc. 5th Annual ACM-SIAM Symp. on Disc. Algorithms*, pages 690–696, 1994.

[5] N. Alon, J.H. Spencer, and P. Erdös. *The Probabilistic Method.* Wiley Interscience, NY, 1992.

[6] S. Even. An algorithm for determining whether the connectivity of a graph is at least k. *SIAM J. Comput.*, 4(3):393–396, 1975.

[7] S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975.

[8] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comp. and System Sci.*, pages 261–272, 1995.

[9] M.R. Henzinger. A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity. *J. Algorithms*, 24(1):194–220, 1997.

[10] M.R. Henzinger, S. Rao, and H.N. Gabow. Computing vertex connectivity: new bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000.

[11] L. Lovász. *Combinatorial Problems and Exercises, 2nd Ed.* North-Holland, NY, 1993.

[12] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[13] G.A. Margulis. Explicit group-theoretical constructions of combinatorial schemes and their applications to the design of expanders and superconcentrators. *Problemy Peredachi Informatsii*, 24:51–60, 1988. In Russian; English translation in *Problems of Information Transmission 24*, 1988, pages 39–46.

[14] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, NY, 1995.

[15] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7:583–596, 1992.

[16] D.A. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proc. 27th Annual ACM Symp. on Theory of Comp.*, pages 388–397, 1995.

[17] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: explicit construction and applications. In *Proc. 25th Annual ACM Symp. on Theory of Comp.*, pages 245–251, 1993.