Better Performance Bounds for Finding the Smallest k-Edge Connected Spanning Subgraph of a Multigraph

Harold N. Gabow *

Abstract

Khuller and Raghavachari [12] present an approximation algorithm (the *KR algorithm*) for finding the smallest k-edge connected spanning subgraph (k-*ECSS*) of an undirected multigraph. They prove the KR algorithm has approximation ratio < 1.85. We prove the KR algorithm has approximation ratio $\leq 1 + \sqrt{1/e} < 1.61$; for odd k this requires a minor modification of the algorithm. This is the best-known performance bound for the smallest k-ECSS problem for arbitrary k. Our analysis also gives the best-known performance bound for any fixed value of $k \geq 3$, e.g., for even k the approximation ratio is $\leq 1 + (1 - \frac{1}{k})^{k/2}$. Our analysis is based on a laminar family of sets (similar to families used in related contexts) which gives a better accounting of edges added in previous iterations of the algorithm. We also present a polynomial time implementation of the KR algorithm on multigraphs, running in the time for $O(nm)$ maximum flow computations, where n (m) is the number of vertices (edges, not counting parallel copies). This complements the implementation of [12] which uses time $O((kn)^2)$ and is efficient for small k.

1 Introduction

Given a k-edge connected graph, we seek a spanning subgraph with the fewest possible number of edges that is still k-edge connected. This is a natural problem in network design. The problem is MAX SNP-hard for any fixed $k \geq 2$ [4, 7]. A number of approximation algorithms have been proposed. This paper achieves the best-known bounds for the case of undirected multigraphs. We do this by improving the analysis of the algorithm of Khuller and Raghavachari[12]. For the rest of the paper we refer to Khuller and Raghavachari's algorithm as the *KR algorithm*, and k-*ECSS* stands for k-edge connected spanning subgraph.

Previous work The problem of finding a smallest 2-ECSS has been widely investigated [13, 2, 15]. The best known approximation ratio is 5/4, achieved by Jothi, Raghavachari and Varadarajan [9] building on the approach of Vempala and Vetta [19]. For 2-ECSS parallel edges pose no problem and can essentially be ignored. However this is not the case for any higher connectivity $k \geq 3$. Gabow [6] gives a 3/2 approximation for the smallest 3-ECSS of a multigraph. Most of these 2- and 3-ECSS algorithms are based on depth-first search.

Cheriyan and Thurimella [3] present an elegant $1 + 2/(k + 1)$ approximation algorithm for the smallest k-ECSS that is valid for arbitrary k under the assumption that the given graph is simple. Their approach is based on an analog of a theorem of Mader on k-node connected graphs. This analog holds for simple graphs. In Appendix A we show that on multigraphs the algorithm of [3]

*Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado 80309-0430, USA. e-mail: hal@cs.colorado.edu

has performance ratio exactly 2. (More precisely 2 is an upper bound on the approximation ratio, and for any fixed $k \geq 2$ there are arbitrarily large graphs where the ratio is arbitrarily close to 2.)

Karger [10] gives a random-sampling algorithm for the smallest k-ECSS that is valid for multigraphs and has performance ratio $1+O(\sqrt{(\log n)/k})$. This bound is of interest when $k >> n$. Here and throughout this paper n and m denote the number of vertices and edges of the given graph, respectively.

The simplest known way to approximate the smallest k-ECSS is by a collection of k maximal spanning forests. This algorithm achieves performance ratio 2 on any multigraph. Nagamochi and Ibaraki [16] show how to find the desired forests in time $O(n(m+n\log n))$ on multigraphs.

Khuller and Raghavachari [12] gave the first algorithm for smallest k-ECSS that achieves approximation ratio better than 2. The KR algorithm is based on depth-first search. They prove the approximation ratio is strictly less than 1.85 on any multigraph. Their implementation uses time $O((kn)^2)$. Fernandes [4] improves the analysis of the KR algorithm on simple graphs, showing the approximation ratio is at most 1.75, and at most 1.7 for large enough k. Both proofs use the assumption of simplicity in crucial ways [4, see Facts 3.3 and 4.4].

Our contribution This paper improves the analysis of the KR algorithm. We show the approximation ratio for smallest k-ECSS is at most $1+\sqrt{1/e} < 1.61$ for arbitrary multigraphs. For any fixed connectivity k the ratio is better, e.g., $1+(1-\frac{1}{k})^{k/2}$ when k is even. Our approach uses a laminar family of sets introduced in [1, 5] and also used in Cheriyan and Thurimella's analysis [3]. This enables more accurate accounting than [12, 4]. Specifically our estimate of the number of edges added in a given step of the KR algorithm takes every previously added edge into account.

To achieve the desired performance bound for odd k we use the recent 3-ECSS algorithm of [6]. The $1+\sqrt{1/e}$ upper bound is proved using a simple bound from [6]. Our best bound for fixed odd k depends on extending the analysis of the 3-ECSS algorithm. Since this extension involves detailed knowledge of the algorithm it has been added as an appendix to [6].

We also provide a polynomial-time implementation of the KR algorithm on weighted graphs, i.e., multigraphs where each edge has an integral value specifying its multiplicity. The running time is dominated by $O(nm)$ max flow computations, specifically

$$O(\min\{(nm)^2\log_b n, \mu n m^2 \log(n^2/m)\log k\})$$

where $b = m/(n\log n)+2$, $\mu = \min\{m^{1/2}, n^{2/3}\}$. The analysis depends on the use of a potential function to show the number of "distinct" iterations of the KR algorithm is polynomial regardless of k.

Organization of the paper Section 2 is devoted to a recurrence that arises in our analysis. Section 3 reviews the KR algorithm and presents our analysis. Section 4 gives the polynomial-time implementation for multigraphs. Appendix A gives the analysis of the algorithm of Cheriyan and Thurimella. This section closes with terminology and notation.

Terminology We often denote a singleton set $\{x\}$ by x. A family of sets is *laminar* if every two sets in the family are either disjoint or one contains the other.

All graphs are undirected. Multiple edges are always allowed in a graph (but not self-loops). We denote edges by juxtaposing the two vertices, as in vw. For multigraphs this notation need not designate a unique edge but this will not cause any confusion. We do not distinguish between a subset of edges and the spanning subgraph induced by those edges. For a simple graph G and an integer c, $c \cdot R$ denotes the multigraph R with every edge having multiplicity c.

Take a (multi)graph $G = (V, E)$. Let X be a set of vertices. An edge with exactly one vertex in X *leaves* X. $d(X)$ denotes the number of edges leaving X. In this notation every edge is counted according to its multiplicity. For distinct vertices x, y, $\lambda(x, y)$ denotes the smallest number of edges whose removal disconnects x and y. If S is a set of edges then appending S as the last argument to d or λ means the graph of interest is the spanning subgraph with edge set S, e.g., $d(X, S), \lambda(x, y; S)$.

Graph G is *k-edge connected* if $\lambda(x, y) \geq k$ for all $x \neq y$. For a given G and k with G k-edge connected, OPT is a smallest set of edges forming a k-ECSS of G.

Given a spanning tree, a nontree edge *covers* every edge in its fundamental cycle, while a tree edge covers itself. A *(maximal) dfs forest* consists of a depth-first spanning tree of each connected component. It is sometimes convenient to assume back edges are directed upwards, i.e., from the deeper end to the shallower end.

2 A recurrence

For arbitrary real numbers α, a, b and for $c \neq 1$, consider the recurrence

$$\begin{aligned} R_1 &= \alpha \\ R_{i+1} &= a + bi + cR_i \quad \text{for all } i \geq 1. \end{aligned} \tag{1}$$

The solution is

$$R_i = c^{i-1}\alpha + a\frac{1-c^{i-1}}{1-c} + \frac{b}{1-c}\left(i - \frac{1-c^i}{1-c}\right).$$

This solution can be derived using the identities for $j \geq 1$, $\sum_{i=0}^{j-1} c^i = \frac{1-c^j}{1-c}$ and $\sum_{i=0}^{j-1}(j-i)c^i = (j+1-\frac{1-c^{j+1}}{1-c})/(1-c)$. Alternatively it can be verified by an easy induction.

We are interested in the case

$$a = \frac{1}{k} + \frac{\epsilon}{k^2} \text{ for } \epsilon \in \{0, 2\}, \; b = \frac{2}{k^2}, \; c = 1 - \frac{1}{k}. \tag{2}$$

Using $1/(1-c) = k$ and some algebra we get for this case

$$R_i = c^{i-1}\left(1 + \alpha - \frac{2+\epsilon}{k}\right) + \frac{2i+\epsilon}{k} - 1. \tag{3}$$

Equation (4) below gives 3 particular values of R_i. (4a) is for k even and (4b-c) are for k odd. We are still assuming (2), as well as $k > 1$. The values (4) all follow from (3) by simple algebra.

$$R_i = \begin{cases} (1-\frac{1}{k})^{k/2} & i = k/2, \quad a = \alpha = 1/k \quad \text{(4a)} \\ (1-\frac{1}{k})^{(k-1)/2} - \frac{1}{k} & i = (k-1)/2, \quad a = \alpha = 1/k \quad \text{(4b)} \\ (1-\frac{1}{k})^{(k-3)/2}\left(1 + \alpha - \frac{4}{k}\right) + \frac{1}{k} & i = (k-1)/2, \quad a = 1/k + 2/k^2 \quad \text{(4c)} \end{cases} \tag{4}$$

Now assume we have $\alpha = r/k$ for some constant $r \leq 8/3$. This includes all the values we are interested in. Then all 3 quantities of (4) satisfy $\lim_{k\to\infty} R_i = \sqrt{1/e} \approx 0.606^+$. The quantity of (4a) increases with k. For (4b) and (4c) we will be interested in those quantities ignoring the last term $\pm 1/k$. The numerical values in the next section (Table 1) will show that for $k \geq 3$, the (4b) quantity (without $1/k$) decreases with k while the (4c) quantity (without $1/k$) increases with k. Our theorem requires a proof of this last relation for (4c). The proof is omitted since the numerical evidence is convincing and more meaningful. The interested reader can supply the argument, using logarithmic differentiation and Taylor series to show the computed derivative (w.r.t. k) is positive for $k \geq 3$ (and $r \leq 8/3$).

3 Analysis of the KR algorithm

We begin this section by reviewing the KR algorithm. Its input is a k-edge connected multigraph $G = (V, E)$. The algorithm returns a k-ECSS having a small number of edges.

We call the basic operation of the KR algorithm a *2-step*. The purpose of a 2-step is to increase the edge-connectivity of the current solution graph by 2. More precisely a 2-step starts with an h-ECSS H and adds a set of edges $F \cup B$ to get an $(h+2)$-ECSS $H \cup F \cup B$. F is a maximal depth-first spanning forest of $G - H$. It is easy to see that $F \cup H$ is $(h+1)$-edge connected. B is a set back edges of the depth-first search. It is formed by traversing the forest F bottom-up, adding edges according to the following rule: When we go from a vertex v to its parent p, check if $\lambda(v, p; H \cup F \cup B) < h + 2$ (here B refers to the current set B). If so, add to B a back edge that goes from a descendant of v to a vertex closest to the root.

[12] proves that such a back edge always exists, and furthermore $H \cup F \cup B$ is $(h+2)$-edge connected at the end of the 2-step.

The overall KR algorithm starts with a spanning subgraph of no edges and performs $\lfloor k/2 \rfloor$ 2-steps. If k is odd then an additional spanning forest is added at the end.

As just described every 2-step in the KR algorithm starts with h even. However the analysis of [12] remains valid if h is odd, and we shall use 2-steps that start with h odd. Also, it is important to bear in mind that in general F will not be a spanning tree but rather a forest. Certainly in the first iteration F is a spanning tree, but even in the second iteration F may consist of a large number of trees.

The analysis of [12] uses a property similar to the idea of "tree carvings" that was introduced in [13] for approximating the 2-ECSS problem. To state it, in any 2-step, let F^* be the set of edges vp of F that force a back edge e to be added to B (i.e., v and p are only $(h+1)$-edge connected when vp is traversed).

Lemma 3.1 (Khuller and Raghavachari) *In any 2-step, every edge of $G - H$ covers at most one edge of F^*.*

Proof: This follows from the fact that each edge of B is chosen to be directed to the shallowest possible vertex. For details see [12, proof of Lemma 3.6]. □

Our analysis, like previous ones, centers around estimating the number of back edges added by each 2-step. We use a laminar family of sets that covers every edge of F^*. A similar family is used in [3] but our family enjoys additional structural properties. For the following lemma fix an arbitrary 2-step. See Fig.1 for an example.

Lemma 3.2 *There is a laminar family of vertex sets $\mathcal{X} = \{X_f : f \in F^*\}$ such that*

(a) each edge $f \in F^*$ leaves X_f and $d(X_f, H \cup F) = h + 1$;

(b) each edge of $G - H$ leaves at most one set of $\mathcal{X}$;

(c) f is the unique edge of F leaving X_f; the edge of B that covers f leaves X_f and no other set of $\mathcal{X}$.

Proof: We start by invoking a well-known fact discovered by Cai [1] and, independently in more general form, Frank [5]: Every k-edge connected (multi)graph has a laminar family $\mathcal{L}$ of vertex sets such that every set $X \in \mathcal{L}$ has $d(X) = k$ and every edge xy with $\lambda(x, y) = k$ leaves at least one set of $\mathcal{L}$. (This is proved by first applying an uncrossing argument and then converting the cross-free family to laminar.) Apply the fact to the $(h+1)$-edge connected graph $H \cup F$. Take any edge $f = xy \in F^*$. It is easy to check that $\lambda(x, y; H \cup F) = h + 1$. Hence f leaves some set $X_f \in \mathcal{L}$. (If f

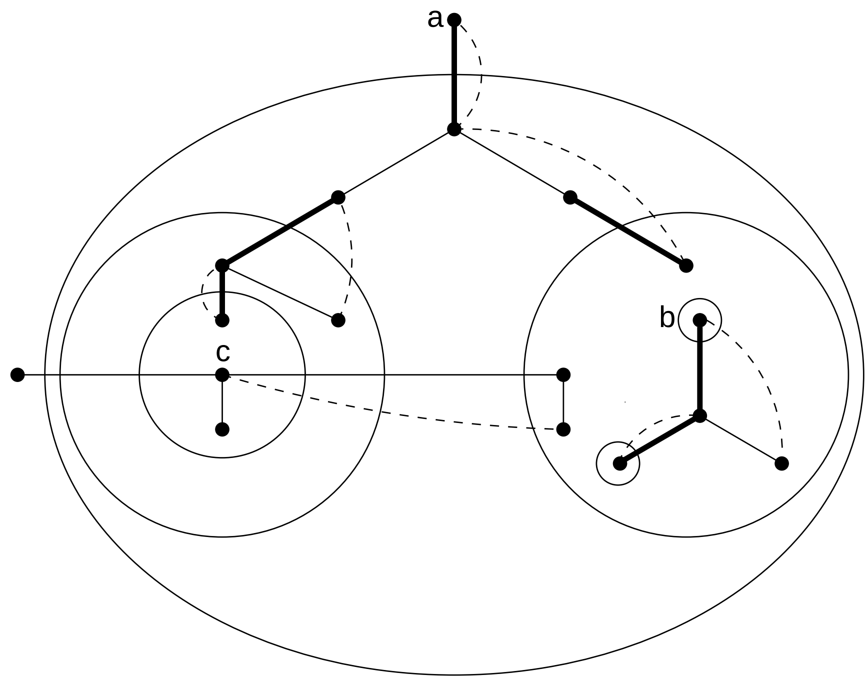

Figure 1: A laminar family $\mathcal{X}$ with 6 sets. Let T_x denote a tree rooted at x. F contains T_a and T_b. F^* contains the 6 heavy edges and B contains the corresponding 6 dashed edges. H contains many edges, including another copy of T_a and T_b, and also T_c. These 3 trees of H have $3, 1$ and 2 internal edges respectively.

leaves more than one set of $\mathcal{L}$ choose one of them arbitrarily to be X_f.) Define $\mathcal{X} = \{X_f : f \in F^*\}$. This laminar family satisfies (a).

Consider any edge $f \in F^*$. Since H is h-edge connected it is easy to see that $d(X_f, H) = h$ and f is the only edge of F leaving X_f. (This gives the first part of (c).) Since F is a maximal spanning forest of $G - H$, an edge of $G - H$ leaves X_f if and only if it covers f. Now (b) follows from Lemma 3.1. The rest of (c) also follows. □

We introduce some more notation concerning a 2-step. An edge of H is *external* if it leaves some set of $\mathcal{X}$ and *internal* if it does not. (See Fig.1). EXT (INT) denotes the set of external (internal) edges of H. Note that an external edge can leave an arbitrary number of sets of $\mathcal{X}$.

Lemma 3.3 $|OPT| \geq (k-h)|B| + |EXT|$.

Proof: Consider any set $X_f \in \mathcal{X}$. OPT contains $\geq k-h$ edges of $G-H$ that leave X_f (since Lemma 3.2 implies $d(X_f, H) = h$). Choose exactly $k-h$ such edges and add them to a set O. Doing this for every $f \in F^*$ makes O a subset of OPT with $|O| = (k-h)|B|$ and $d(X_f, O \cup H) = k$. This follows since an edge added for X_f does not leave any other set X_g (Lemma 3.2(b)). Furthermore $|F^*| = |B|$.

Initialize a set J to EXT. Observe that

$$d(X, O \cup J) = k \text{ for every } X \in \mathcal{X}. \tag{5}$$

We will now repeatedly remove 1 edge from J and add ≥ 1 new edge of OPT to O. Doing this for every edge of J will enlarge O to a subset of $\geq (k-h)|B| + |EXT|$ distinct edges of OPT. Clearly this implies the inequality of the lemma. In order to accomplish this we will maintain 2 invariants throughout the procedure, (5) and

$$OPT \subseteq O \cup J \cup \overline{EXT}. \tag{6}$$

Condition (6) obviously holds when J is initialized to EXT.

Consider an external edge e remaining in J. If $e \in OPT$ simply transfer e from J to O. Obviously (5)–(6) continue to hold.

So assume $e \notin OPT$. e leaves one or more sets $X \in \mathcal{X}$. For each such X choose 1 edge $d \in OPT - (O \cup J)$ that leaves X. Such a d exists by (5) and the fact that OPT is k-edge connected. Let D be the set of all chosen edges d. We will change O and J to $O' = O \cup D$ and $J' = J - e$ respectively.

Note that $|O'| > |O|$ as promised. O' and J' satisfy (6) since $e \notin OPT$. We must check that $O' \cup J'$ satisfies (5). It is obvious that any $X \in \mathcal{X}$ has $d(X, O' \cup J') \geq k$ but we must check that equality holds, i.e., each $d \in D$ leaves exactly one set of $\mathcal{X}$. (6) shows that $d \notin EXT$. Thus d is an edge of $G - H$. Now Lemma 3.2(b) shows that d leaves exactly one set of $\mathcal{X}$. $\Box$

We will look at the 2-steps that built up the h-ECSS H. If h is even let F_j and B_j, $j = 1, \ldots, h/2$ denote the sets of edges F and B added in the jth 2-step, respectively. So $H = \bigcup_{j=1}^{h/2} F_j \cup B_j$. For odd h we use the same definition except for some slight changes allowed when $j = 1, 2$, see below.

The notation $c(K)$ denotes the number of connected components of the graph K. By convention if K is a set of edges of G we consider K to be a spanning subgraph of G in the notation $c(K)$. As usual each isolated vertex contributes 1 to $c(K)$.

Observe that B_j is a forest. This follows from a simple fact about dfs trees:

Proposition 3.4 *A set of back edges of a dfs tree is a forest if it contains ≤ 1 back edge directed from each vertex.*

Proof: We prove the contrapositive. Let x be a vertex of greatest depth in a cycle of back edges. Both cycle edges incident to x are back edges directed from x. $\Box$

We will use a slightly stronger version of the next result when we treat odd values of k: The lemma holds even if A is any acyclic set of back edges of F_j.

Lemma 3.5 *For A an acyclic subset of $F_j \cup B_j$, $c(A \cap INT) \geq |B| + c(F_j)$.*

Example: In Fig.1 let F_j consist of copies of T_a, T_b and T_c; as before F consists of T_a and T_b. The right-hand side of the lemma is $|B| + c(F_j) = 6 + 3 = 9$. T_a and T_b contain 8 and 4 vertices respectively. For $A = T_c$, $c(A \cap INT) = 8 + 4 + 3 = 15$. For $A = F_j$, $c(A \cap INT) = 5 + 3 + 3 = 11$.

Proof: Consider the set of edges $S = (A \cap INT) \cup F^*$. We first show S is a forest. By way of contradiction assume S contains a cycle C. Since A is acyclic C contains edges of F^*. So $\mathcal{X}$ has a minimal set X_f with $f \in C$. f is the unique edge of C leaving X_f (Lemma 3.2(c)). But this is a contradiction since any cycle leaves X_f an even number of times.

Next we claim that every edge of S joins two vertices in the same tree of F_j. This holds by definition for A. For an edge $f = xy \in F^*$, x and y are in the same tree of F. Since F_j is chosen as a maximal spanning forest, x and y must also be in the same tree of F_j.

The claim implies that $c(S) \geq c(F_j)$. Thus $c(A \cap INT) \geq |F^*| + c(F_j)$, and the lemma follows. $\Box$

Corollary 3.6 *(a)* $|B_j \cap INT| + |B| + c(F_j) \le n$. *(b)* $|F_j \cap EXT| \ge |B|$.

Proof: Recall that any acyclic set of edges R has $|R| + c(R) = n$. To show part (b) apply Lemma 3.5 to $A = F_j$ to get

$$|F_j| + c(F_j) = n = |F_j \cap INT| + c(F_j \cap INT) \ge |F_j \cap INT| + |B| + c(F_j).$$

(b) follows since F_j is partitioned into $F_j \cap EXT$ and $F_j \cap INT$. Part (a) follows from a similar but simpler argument, using $A = B_j$. $\Box$

Lemma 3.7 *If k is even the KR algorithm achieves approximation ratio $1 + (1 - \frac{1}{k})^{k/2}$.*

Proof: Let S_i be the total number of back edges added in the first i 2-steps (i.e., the steps that achieve $2i$ connectivity). We shall prove the recurrence

$$(7) \qquad \begin{array}{rcl} S_1 & \le & \frac{1}{k}|OPT| \\ S_{i+1} & \le & \left(\frac{1}{k} + \frac{2i}{k^2}\right)|OPT| + \frac{k-1}{k} S_i \quad \text{for all } k/2 > i \ge 1. \end{array}$$

It is easy to see that (7) implies $S_i/|OPT|$ is upper-bounded by the quantity R_i of recurrence (1) with the values (2) and $\epsilon = 0$, $\alpha = 1/k$. So (4a) shows the total number of back edges added by the algorithm is $S_{k/2} \le (1 - \frac{1}{k})^{k/2}|OPT|$. Besides these back edges the algorithm adds $k/2$ dfs forests. The degree lower bound $|OPT| \ge kn/2$ implies the forests have total size at most $|OPT|$. The lemma follows.

It remains to prove (7). The base case $S_1 = |B_1| \le |OPT|/k$ is the carving lower bound [11, Theorem 6.2]. Alternatively the base case follows by taking $H = \emptyset$ in Lemma 3.1 and concluding that OPT contains $\ge k|B_1|$ edges.

Next we prove the upper bound on S_{i+1}. We will apply the preceding analysis of the 2-step that enlarges H from an h-ECSS to an $(h+2)$-ECSS. Take $h = 2i$. So

$$(8) \qquad \begin{array}{rcll} S_i & = & \sum_{j=1}^{i} |B_j| & \text{(8a)} \\ S_{i+1} & = & S_i + |B|. & \text{(8b)} \end{array}$$

The degree lower bound shows $|OPT| \ge kn/2$. Combining this with Corollary 3.6(a) shows

$$(9) \qquad |B_j \cap INT| + |B| + c(F_j) \le 2|OPT|/k, \quad j = 1, \ldots, i.$$

We will discard the term $c(F_j)$.

Using Corollary 3.6(b) in Lemma 3.3 gives

$$(10) \qquad (k - 2i)|B| + \sum_{j=1}^{i} (|B| + |B_j \cap EXT|) \le |OPT|.$$

Add the $i+1$ inequalities (9)–(10), and use the fact that B_j is partitioned into sets $B_j \cap INT$ and $B_j \cap EXT$, to get

$$(11) \qquad k|B| + \sum_{j=1}^{i} |B_j| \le (1 + 2i/k)|OPT|.$$

Add $k-1$ times (8a) to (11) and simplify using (8b) to get

$$kS_{i+1} \le (1 + 2i/k)|OPT| + (k-1)S_i.$$

k	2	4	6	8	10	20	100	1000	10000
ratio	1.5	1.562^+	1.578^+	1.586^+	1.590^+	1.598^+	1.605^+	1.6063^+	1.6065^+

k	3	5	7	9	11	21	101	1001	10001
KR	1.666^+	1.64	1.629^+	1.624^+	1.620^+	1.613^+	1.608^+	1.6066^+	1.6065^+
KRs	1.555^+	1.586^+	1.594^+	1.598^+	1.600^+	1.603^+	1.606^+	1.6064^+	1.6065^+
KR3	1.5	1.56	1.577^+	1.585^+	1.589^+	1.598^+	1.604^+	1.6063^+	1.6065^+

Table 1: Values of our approximation ratio for selected k.

This is equivalent to the desired relation of (7). $\square$

The bound of Lemma 3.7 increases monotonically with k and is always $< 1+\sqrt{1/e} = 1.60653+$. Selected values are given in the first 2 rows of Table 1.

We turn to odd values of k. The original KR algorithm for odd k executes the same 2-steps as the algorithm for $k-1$, after which it adds a spanning forest to achieve connectivity k. The analysis of Lemma 3.7 shows the number of back edges S_i satisfies recurrence (7). So the total number of back edges is $|OPT|$ times (4b). There are $(k+1)/2$ dfs forests, contributing a total of $\le (1+1/k)|OPT|$ edges. We conclude the approximation ratio is $1+(1-\frac{1}{k})^{(k-1)/2}$. This quantity is illustrated in the row labelled "KR" of Table 1. It approaches $1+\sqrt{1/e}$ from above.

To achieve an approximation ratio that is always below $1+\sqrt{1/e}$ we use the following version of the KR algorithm: Initialize the solution subgraph to a "good" 3-ECSS. Then execute $(k-3)/2$ 2-steps. It remains to choose the base 3-ECSS.

The simplest choice is to use the KR algorithm: Initialize H to a dfs tree of G. Then do a 2-step to enlarge H to a 3-ECSS. Our analysis of this variant leads to the same bound as the original KR algorithm. The analysis hinges on the fact that the first 2-step adds $\le |OPT|/k$ back edges. The total number of back edges remains $|OPT|$ times (4b). Further details are left to the reader.

To improve this we use the algorithm of [6] which we shall refer to as the ear algorithm. This algorithm finds a 3-ECSS A of a multigraph, achieving approximation ratio 3/2. The algorithm works in 3 phases which we now describe.

Phase I initializes A to a dfs tree, which we denote as F for consistency with a 2-step. Phase II begins by making A 2-edge connected by adding a set of back edges B'. To do this it first uses the 2-step procedure to construct B and the forcing edges F^*. It chooses the set B' in a top-down pass that guarantees each edge of B' covers a distinct edge of F^*. The edges of B' are called *long ear* edges. To describe the rest of Phase II note that the algorithm maintains a partition of V into sets of vertices that are known to be 3-edge connected in A, called "t-sets". The second part of Phase II adds edges called *short ears*. Each short ear merges at least 3 t-sets into 1. Phase III makes A 3-edge connected by adding a spanning forest of the t-sets.

For our analysis we partition the final A into the spanning tree F plus 2 forests: B_0 contains the long ear edges. It is acyclic, by Proposition 3.4 and Lemma 3.1. B_1 consists of the short ears plus the spanning forest of Phase III. It is acyclic, since each edge merged ≥ 2 distinct t-sets. Take a value α such that $|B_0|+|B_1| \le \alpha|OPT|$.

Let *KR3* denote the k-ECSS approximation algorithm with the ear algorithm used for initialization. For $i \ge 1$ let S_i be the total number of back edges added in the initialization step and the first $i-1$ 2-steps. (These steps achieve $2i+1$ connectivity.) These quantities satisfy the recurrence

$$(12) \qquad \begin{array}{rcl} S_1 & \leq & \alpha|OPT| \\ S_{i+1} & \leq & \left(\frac{1}{k}+\frac{2i+2}{k^2}\right)|OPT|+\frac{k-1}{k}S_i \quad \text{for all } (k-1)/2 > i \geq 1. \end{array}$$

This is proved by essentially the same argument as Lemma 3.7. The differences stem from the fact that the initialization and first $i-1$ 2-steps add a total of i dfs forests and $i+1$ forests of back edges to the solution graph. In more detail, Lemma 3.5 and Corollary 3.6(a) remain valid for B_0 and B_1. (10) becomes

$$(k-2i-1)|B|+|B_0\cap EXT|+\sum_{j=1}^{i}(|B|+|B_j\cap EXT|)\leq|OPT|.$$

Finally in the last two equations the term $2i$ becomes $(2i+2)$.

It remains to determine α, i.e., we must bound the number of back edges in A in terms of $|OPT|$ (the size of the smallest k-ECSS). We will derive the main theorem using a simple bound for α: [6] gives a short proof that the ear algorithm achieves a 14/9 approximation ratio. We extend that proof as follows.

Lemma 3.8 *A contains $\leq \frac{8}{3k}|OPT|$ back edges.*

Proof: The analysis of [6] is based on 3 lower bounds on $|OPT|$: the degree lower bound, the carving lower bound, and the component lower bound. All 3 of these are actually lower bounds on the size of the smallest k-ECSS, which the argument of [6] specializes to $k=3$. Applying the argument that proves a 14/9 approximation ratio to general k gives the lemma – the only changes are in the algebra of [6, Lemma 4.5]. □

Now use (4c) to solve (12) with $\alpha = 8/(3k)$, $\epsilon = 2$. This gives an upper bound on the number of back edges added by KR3. In addition KR3 adds $(k-1)/2$ dfs forests. The degree lower bound shows they contain a total of $\leq \frac{k-1}{k}|OPT|$ edges. Summing the two bounds shows the size of KR3's subgraph is at most

$$\left(1+(1-\frac{1}{k})^{(k-3)/2}(1-\frac{4/3}{k})\right)|OPT|.$$

This upper bound is illustrated in the row labelled "KRs" in Table 1. As can be seen, the upper bound is less than $1+\sqrt{1/e}$ (Section 2 describes a proof).

Theorem 3.9 *The KR algorithm has approximation ratio $\leq 1+\sqrt{1/e}$ on multigraphs.* □

The above bound for odd k can be improved, for any fixed k, to the row labelled "KR3" in Table 1. This is done by extending the proof of the 3/2 performance ratio for the ear algorithm [6] to show we can take $\alpha = \frac{5}{2k}$ in (12). The extended proof depends on detailed knowledge of [6] and so has been added as an appendix to [6]. Assuming that result we get the following corollary. Note the corollary's bound for $k=3$ is 3/2 as expected.

Corollary 3.10 *For $k>1$ the KR algorithm has approximation ratio at most*

$$\begin{array}{ll} 1+(1-\frac{1}{k})^{k/2} & k \textit{ even;} \\ 1+(1-\frac{1}{k})^{(k-3)/2}(1-\frac{3/2}{k}) & k \textit{ odd.} \end{array}$$

□

4 Efficient implementation

When the desired connectivity k is small the implementation of [12] is efficient. This section concentrates on implementing the KR algorithm efficiently for graphs with large multiplicities. Specifically we assume a weighted graph representation, where each edge is given with an (arbitrarily large) integer multiplicity. We implement the KR algorithm in time dominated by $O(nm)$ maximum flow computations.

The algorithm begins with a simple initialization: If k is odd the solution graph H is initialized to the 3-ECSS provided by the ear algorithm; if k is even H is $\emptyset$. The ear algorithm runs in time $O(m\alpha(m,n))$ [6] which is dominated by our desired time bound. (It is also dominated by the bound of [12], so this does not change.) For the rest of the section the parity of k is irrelevant.

The implementation is organized into "phases." Each phase simulates a number of 2-steps, until either the current solution graph H becomes k-edge connected or the multiplicity of one or more edges of $G-H$ decreases to 0 or 1. This implies there are $\leq 2m+1$ phases.

Each phase simulates a sequence of consecutive 2-steps that use the same forest F. The phase consists of a number of "multisteps." A multistep simulates the execution of as many consecutive 2-steps as possible that each add a copy of the same set $F \cup B$.

Fix a 2-step with current graph H and dfs forest F. An edge $xy \in F$ is *critical* if $\lambda(x,y;H \cup F) = h+1$. We shall see that the critical edges determine the set B. To implement a multistep we need to know when an edge becomes critical. The following lemma answers this question. It also implies several other facts that we need about critical edges.

Take an edge $xy \in F$. The graph HF_{xy} is constructed by starting with H and contracting every connected component of $F - xy$. We use the term "node" to refer to a vertex of HF_{xy}, and node x (node y) refers to the node of HF_{xy} that contains vertex x (vertex y) of G. For a set of edges S, $cov(xy, S)$ denotes the number of edges of S that cover xy. We count each edge according to its multiplicity in S.

To state the lemma suppose we are executing the KR algorithm (not our implementation). Let H be an h-ECSS of the k-edge connected graph G. Let F be a maximal spanning forest of $G-H$. Take any $c \geq 0$ so $h+2c \leq k-2$. Suppose that c consecutive 2-steps enlarge H to the $(h+2c)$-ECSS $K = H \cup (c \cdot F) \cup B_c$. (In general B_c is a union of a number of different B sets.)

Lemma 4.1 *Suppose a $c+1$st 2-step is to be done, adding a $c+1$st copy of F. The critical edges in this 2-step are the edges $xy \in F$ satisfying*

$$\lambda(x,y;HF_{xy}) + cov(xy, B_c) = h + c.$$

Proof: Edge $xy \in F$ is critical for the next 2-step if $\lambda(x,y;K \cup F) = h+2c+1$. This means there is a set S of vertices containing x but not y such that

$$d(S, K \cup F) = h + 2c + 1.$$

If $d(S,F) \geq 2$ then $d(S,K\cup F) \geq d(S,H) + (c+1)d(S,F) \geq h+2c+2$. Hence we can assume $d(S,F) \leq 1$. Since xy leaves S, we can assume $d(S,F) = 1$ and xy is the unique edge of F leaving S.

Now we have $d(S,K\cup F) = d(S,H)+(c+1)+d(S,B_c)$. So the condition for criticality becomes

$$d(S,H) + d(S,B_c) = h + c. \tag{13}$$

Since every edge of B_c joins vertices in the same tree of F, an edge of B_c leaves S if and only if it covers xy, i.e., $d(S,B_c) = cov(xy,B_c)$. So if S satisfies (13) it must minimize $d(S,H)$. (It is easy to see that any S satisfies (13) with $\geq$, by tracing back to the original criticality condition.) Since xy is the unique edge of F leaving S, S corresponds to a set of nodes in HF_{xy}, and S contains node x but not node y. So (13) is equivalent to the equation of the lemma. $\square$

The special case of the lemma with $c = 0$, $B_c = \emptyset$ is of interest. For instance consider the following:

Fact 1. Suppose an edge xy is critical at the start of a 2-step. During the bottom-up traversal of the 2-step the quantity $\lambda(x, y; H \cup F \cup B)$ changes from $h+1$ to $h+2$ precisely when the first edge covering xy gets added to B.

To prove Fact 1 use the special case of the lemma. Observe that the first paragraph of the proof of Lemma 4.1 shows $\lambda(x, y; H \cup F \cup B)$ remains equal to $h + 1$ as long as there is a set S containing x but not y such that xy is the unique edge of F leaving S and no edge of B covers xy.

Fact 1 implies that the set of critical edges C at the start of a 2-step determines the set of back edges B that get added in the 2-step. So we use the notation $B(C)$. The set of "forcing edges" F^* is also determined by C, and we use the notation $F^*(C)$. Note from the bottom-up procedure of a 2-step that $B(C) = B(C')$ for any set C' satisfying $F^* \subseteq C' \subseteq C$.

We need two more facts about critical edges. Consider 2 consecutive 2-steps, where the first 2-step is done for critical edges C and the second 2-step uses the same forest F as the first.

Fact 2. An edge of C covered by exactly one edge of $B(C)$ is critical for the second 2-step.

Fact 3. An edge of $F - C$ that is critical for the second 2-step is not covered by any edge of $B(C)$.

Facts 2–3 follow from the Lemma 4.1 by first taking $c = 0$, $B_c = \emptyset$ and then $c = 1$, $B_c = B(C)$.

We can now give the stopping criterion for a multistep. Recall that a multistep simulates the execution of as many consecutive 2-steps as possible that each add a copy of the same set $F \cup B$. In greater detail it simulates 2-steps until either

(a) the number of available copies of some edge of F or B drops to 0, or

(b) some edge of F changes from noncritical to critical.

Let us prove that as long as (a) and (b) do not occur, each 2-step adds the same set $B(C)$. Let $F^* = F^*(C)$. Each edge of F^* is covered by exactly 1 edge of B. Fact 2 shows the set C' of edges that are critical in the next 2-step includes F^*. Assuming (b) does not occur $C' \subseteq C$. This implies $B(C) = B(C')$ as desired.

If (a) occurs the phase ends. (Note that we have described a phase as ending when the number of available copies of some edge drops to 0 or 1, which seems slightly different from (a). The difference is due to the situation where the multiplicity of an edge in $F \cap B$ drops to 1.)

For (b) to occur the new critical edge e must not be covered by B (Fact 3). Now it is easy to see that in the next 2-step the deepest such e enters the set F^* and causes the set B to change.

This justifies the following detailed statement of a multistep: The multistep begins with its set of critical edges C. It determines the set $B = B(C)$. Then it adds the greatest number of copies of $F \cup B$ to H until (a) or (b) occurs. Lemma 4.1 is used to determine when (b) will occur.

It is not hard to see this gives a polynomial time implementation of a multistep. (Implementation details are given below.) It remains to bound the number of multisteps in a phase. This is nontrivial because in a sequence of consecutive 2-steps with the same forest F, a given edge of F can change from critical to noncritical and back again many times.

It is convenient to extend some tree terminology from vertices to tree edges. An *edge-ancestor* of a vertex v is a tree edge whose deeper vertex is an ancestor of v. An edge-ancestor of a tree edge e is an edge-ancestor of the deeper vertex of e. If e is a tree edge then $depth(e)$ is the depth of the deeper vertex of e. We start by describing how F^* changes when a new edge becomes critical.

Lemma 4.2 *Let C be a set of critical edges. Let c be a tree edge that is not covered by $B(C)$ but is covered by some back edge. Then for some edge f covering c either*

(a) $F^*(C \cup c) = F^*(C) + c$ and $B(C \cup c) = B(C) + f$, or

(b) for some set $D \subseteq C \cup c$, some proper edge-ancestor $\underline{a}$ of c, and some edge e covering $\underline{a}$, $F^*(D) = F^*(C) - a + c$ and $B(D) = B(C) - e + f$.

Proof: Let f be a back edge that covers c and is directed to the shallowest possible vertex, say w. Let a be the deepest edge-ancestor of c belonging to $F^*(C)$, if such exists. We will show that (b) holds if a exists and (a) holds if it does not. If a exists let $e \in B(C)$ be the edge covering a and let v be the head of e. Since a is an edge-ancestor of c, v is an ancestor of w. Let P_{wv} consist of all edges on the tree path from w to v. If v does not exist then take $v = w$ and $P_{wv} = \emptyset$. Let

$$D = C - P_{vw} + c.$$

Note that $D = C \cup c$ if a does not exist.

We will execute the bottom-up procedure to construct both $B(C)$ and $B(D)$. The desired relations (a)–(b) result from the fact that the two executions differ only in 1 step.

Let F_- consist of all edges of F except the edge-ancestors of c. Clearly $C \cap F_- = D \cap F_-$. Hence the bottom-up procedure works the same for C and D on F_-, placing the same edges of F_- into both $F^*(C)$ and $F^*(D)$ and the same back edges into $B(C)$ and $B(D)$.

The next step of the bottom-up procedure adds f to $B(D)$ and c to $F^*(D)$. If a does not exist then both procedures are done, and part (a) of the lemma holds. Suppose a exists. The bottom-up procedure adds e to $B(C)$ and a to $F^*(C)$.

The remaining steps of both executions are exactly the same. This follows since the remaining uncovered edges of C are edge-ancestors of v, by definition. The remaining uncovered edges of D are edge-ancestors of w, and since $D \cap P_{wv} = \emptyset$, they are edge-ancestors of v. C and D contain exactly the same edge-ancestors of v. So the remaining steps of both executions are the same, and we have verified (b) of the lemma. $\Box$

Consider the potential function

$$\Phi = \sum\{depth(e) : e \in F^*\}.$$

For Φ we compute depths in the forest F. We will prove that every multistep in a phase except possibly the last increases Φ by at least 1. Since $\Phi \le n^2$ this implies that a phase has $O(n^2)$ multisteps.

If C is a set of critical edges, let $\Phi(C)$ denote the above expression with $F^* = F^*(C)$. When we apply Lemma 4.2 it is convenient to unify the two cases by taking D to be $C \cup c$ in case (a). Observe that $\Phi(D) > \Phi(C)$ for both cases (a) and (b) of Lemma 4.2 (since $\Phi(D) = \Phi(C) + depth(c) - depth(a)$ where we take $depth(a) = 0$ in (a)).

Lemma 4.3 *Every multistep in a phase except the first or last increases Φ.*

Proof: Consider a multistep that is not the first or last of its phase. Let C_0 be the set of edges that were critical in the previous multistep and remain critical at the start of the current multistep. Let C_1 be the set of edges that have changed from noncritical to critical. Fact 2 implies that B of the previous multistep equals $B(C_0)$. Fact 3 implies no edge of C_1 is covered by $B(C_0)$.

We will construct sets $D_i \subseteq C_0 \cup C_1$, $i = 0, \ldots, I$ for some $I \geq 1$, with $\Phi(D_i) > \Phi(D_{i-1})$, $D_0 = C_0$ and $D_I = C_0 \cup C_1$. Clearly this implies the lemma. We will also have each D_i maximal in the sense that D_i contains every edge of $C_0 \cup C_1$ that is covered by $B(D_i)$. Clearly D_0 is maximal.

Supposing D_{i-1} has been constructed, construct D_i as follows. Take an edge $c \in C_0 \cup C_1 - D_{i-1}$ that has maximum possible depth. (For $i = 1$, $C_1 - C_0 \neq \emptyset$ since this is not the last multistep of the phase.) By maximality c is not covered by $B(D_{i-1})$. Apply Lemma 4.2 to $C \equiv D_{i-1}$ and c. The lemma gives set $D \equiv D_i$ with $D_i \subseteq C \cup c \subseteq C_0 \cup C_1$ and $\Phi(D_i) > \Phi(D_{i-1})$. We can enlarge D_i so it contains all edges of $C_0 \cup C_1$ that are covered by $B(D_i)$, without changing $F^*(D_i)$ or $B(D_i)$. (For edge f this depends on the choice of c to have maximum depth. For other edges, which belong to D_{i-1}, this depends on the maximality of D_{i-1}.) Now D_i is maximal. If now $D_i = C_0 \cup C_1$ then take $I \equiv i$ and we are done. This eventually occurs since Φ always increases. $\Box$

It is easy to see that the lemma guarantees each phase runs in polynomial time and we have a polynomial-time algorithm. We now show the time for a phase is dominated by $O(n)$ maximum flow computations. We first assume flows are computed by the algorithm of [14] which uses time $O(nm \log_b n)$ for $b = m/(n \log n) + 2$.

A phase starts by doing a dfs to determine F. It then computes two sets of values that remain constant during the phase: First the values $\lambda(x, y; HF_{xy})$, $xy \in F$. These values do not change since F does not change. The values are computed using $O(n)$ flow computations, each on a graph of $\leq n$ vertices and $\leq m$ edges.

The second set of values are the *lowpoint* values of the vertices of F [12, 17]. These values do not change since the set of nontree edges does not change. The *lowpoint* values are computed in $O(m)$ time. This enables each multistep to determine the new set B in a bottom-up traversal of F in $O(n)$ time.

The phase also maintains the values $cov(xy, B_c)$ for each $xy \in F$. Here B_c is the set of nontree edges added so far in the phase. Initially all *cov* values are 0.

Now we show that each multistep can be executed in $O(n)$ time. Use Lemma 4.1 to determine the set of critical edges C for this multistep. All quantities needed have already been computed, so this takes $O(n)$ time. Compute $B = B(C)$ in $O(n)$ time using *lowpoint* values.

Determine the greatest number of 2-steps that can be executed, as follows. Compute the values $cov(xy, B)$ for $xy \in F$ and the new B in time $O(n)$. Edges with $cov(xy, B) = 0$ are candidates for becoming critical. For each such xy the equation of Lemma 4.1 gives the value of c at which xy becomes critical. Let c_0 be the least such value (if it exists).

Let the current solution graph have edge connectivity $h + 2c$. Let a be the fewest number of available copies of an edge of F or B. The multistep adds $d = \min\{c_0, a, (k - (h + 2c))/2\}$ copies of $F \cup B$. Finally the multistep increases each value $cov(xy, B_c)$ by $d \times cov(xy, B)$.

Lemma 4.3 shows the total time for all multisteps in a phase is $O(n^3)$. This is within the time bound for a phase.

Theorem 4.4 *The KR algorithm can be implemented on multigraphs in time $O((nm)^2 \log_b n)$ for $b = m/(n \log n) + 2$.* $\Box$

Now we assume that maximum flows are computed by the algorithm of [8]. If all capacities are integers $\leq k$ this algorithm runs in time $O(\mu m \log(n^2/m) \log k)$ for $\mu = \min\{m^{1/2}, n^{2/3}\}$.

A phase starts as before, computing F, the values $\lambda(x, y; HF_{xy})$ for $xy \in F$, and all *lowpoint* values. In the flow computations we can assume all capacities in HF_{xy} are integers $\leq k$, since flow values larger than k are irrelevant. Hence the desired time bound for computing a flow applies.

We must speed up the implementation of a multistep. We achieve time $O((n \log n)^2)$ in total for all multisteps in a phase. Since this quantity is $O(\mu nm)$ it suffices for our desired time bound. The approach is based on a stronger version of Lemma 4.3. Recall that Lemma 4.2 shows how to make a new edge c critical, by adding an edge to both F^* and B and perhaps dropping an edge from both of these sets. Call this change to F^* and B an *edge swap*. The proof of Lemma 4.3 shows how to update the set B from one multistep to the next, using a total of $O(n^2)$ edge swaps in the entire phase. Our algorithm updates B using this procedure. We will show that each edge swap can be done in time $O(\log^2 n)$. We sketch the implementation, leaving some details to the interested reader.

We first describe the mechanism for determining critical edges. For $xy \in F$ define $t(xy) = \min\{\lambda(x, y; HF_{xy}) + cov(xy, B_c), k\}$. Lemma 4.1 shows that if xy is not covered by the current B set, $t(xy) < k$, and every following 2-step adds the current B, then xy becomes critical after step $t(xy) - h$, where we have indexed the 2-steps of the current phase starting at 1.

Define a key $T(xy)$ for each $xy \in F$ by

$$T(xy) = K^2 cov(xy, B) + Kt'(xy) + (n - depth(xy)).$$

Here $K = \max\{k, n\}$. B denotes the current set B (even in the middle of the bottom-up procedure). $t'(xy)$ is the quantity $t(xy)$ excluding the contribution of all current edges B to $cov(xy, B_c)$. Observe that an edge xy that is not covered by B has $T(xy) = Kt(xy) + (n - depth(xy))$. So it is easy to see that the edge xy with smallest key $T(xy)$ is the next edge to turn critical, and if there is a tie for this edge xy is as deep as possible. (The latter is needed since we follow the proof of Lemma 4.3 to update B.)

Our data structure is based on the heavy path decomposition [18] of the forest F. Recall this notion gives a partition of the edges of F into "heavy paths." Each edge of a given heavy path has, to within a factor of 2, the same number of vertex descendants in F. The path from any vertex to the root of its tree in F intersects at most $\log n$ heavy paths.

The edges of each heavy path are stored in a binary search tree, where symmetric order is the same as order in the heavy path. Each node of the search tree keeps track of the smallest key $T(xy)$ in its subtree, and each node has a displacement quantity that is added to all the keys in its subtree. (This enables all keys in a subtree to be increased by the same value in time $O(1)$.) There is a priority queue Q that contains the edge of smallest key from each heavy path. Finally each edge of F^* is marked in the heavy path decomposition, so that the deepest edge-ancestor in F^* of a given tree edge can be found in $O(\log^2 n)$ time.

At the start of a multistep we find the next new critical edge using the queue Q in time $O(\log n)$. We update the set B by following the procedure of Lemma 4.3. Each swap is done in $O(\log^2 n)$ time. The number of 2-steps to be simulated is computed as before. We achieve time $O((n \log n)^2)$ for a phase as desired.

Corollary 4.5 *The KR algorithm can be implemented in time* $O(\mu nm^2 \log(n^2/m) \log k)$ *for* $\mu = \min\{m^{1/2}, n^{2/3}\}$. $\square$

Appendix A The Cheriyan-Thurimella algorithm on multigraphs

Throughout this section take any fixed $k \geq 2$. The algorithm of Cheriyan and Thurimella [3] consists of two steps: First set M to a minimum cardinality set of edges in which every vertex has degree $\geq k$. Next take C to be a minimal set of edges that makes $M \cup C$ k-edge connected. Return the k-ECSS $M \cup C$.

We will exhibit multigraphs that have $|M \cup C| \geq (2-\epsilon)|OPT|$, for any $\epsilon > 0$. On the other hand we will prove the approximation ratio is always ≤ 2. This upper bound holds even for the weaker version of the algorithm where M is chosen to be a minimal set rather than a minimum cardinality set. [3] proves much stronger upper bounds when the graph is restricted to be simple.

To prove the upper bound suppose that the edges of $M \cup C$ can be partitioned into k forests. Then the degree lower bound gives $|M \cup C| \leq k(n-1) < kn \leq 2|OPT|$ as desired. Hence our upper bound follows from this lemma:

Lemma A.1 *The edges of $M \cup C$ can be partitioned into k spanning forests.*

Proof: We first partition M into spanning forests $F_1, \ldots, F_k$. To do so repeatedly remove a maximal spanning forest from M until k forests are obtained. Let $N = \cup_{i=1}^{k} F_i$. We claim $N = M$. If not take an edge $xy \in M - N$. Vertices x and y are in the same tree in each forest F_i, by maximality. Hence $d(x,N), d(y,N) \geq k$. But this contradicts the minimality of M.

Next take each edge $e \in C$ and add it to a forest F_i where it joins 2 different trees. We claim this is always possible. If not suppose $e = xy \in C$ cannot be added to any forest. Thus each forest F_i contains an xy-path. This implies $\lambda(x, y; M \cup C - e) \geq k$. This contradicts the minimality of C. □

We turn to the example graphs. Let C_n denote the cycle on n vertices. Index its vertices from 0 to $n-1$ in cyclic order.

First suppose k is even. Let $G = k \cdot C_n$. For simplicity assume n is even (a similar construction works if n is odd). $(k/2) \cdot C_n$ is a k-ECSS of G with exactly $kn/2$ edges. Since this matches the degree lower bound this graph can be taken as OPT.

Execute the approximation algorithm on G as follows. In the first step choose $M = k \cdot \{(2i, 2i+1) : i = 0, \ldots, n/2-1\}$. (This is valid since every vertex has degree exactly k.) In the second step choose $C = k \cdot \{(2i-1, 2i) : i = 1, \ldots, n/2-1\}$. (This is valid since it makes $M \cup C$ a minimal k-edge connected graph.) We have $|M \cup C| = kn/2 + k(n/2-1) = k(n-1)$. Thus $|M \cup C|/|OPT| = 2(n-1)/n$ which approaches 2 as $n \to \infty$.

A similar example works if k is odd and ≥ 3. The graph is $k \cdot C_n$ plus the matching $A = \{(2i, 2i+3) : i = 0, \ldots, n/2-1\}$. Here we assume n is even and ≥ 4, and addition is modulo n. It is easy to see that for $k = 1$ this graph is 3-edge connected. (See [6] for other uses of this 3-edge connected graph.) The subgraph $\lfloor k/2 \rfloor \cdot C_n \cup A$ is k-edge connected (since $C_n \cup A$ is 3-edge connected and $(\lfloor k/2 \rfloor - 1) \cdot C_n$ is $2(\lfloor k/2 \rfloor - 1)$ edge-connected). Each vertex has degree k so again $|OPT| = kn/2$. The approximation algorithm can choose exactly the same sets M, C as before, so again the approximation ratio approaches 2.

References

[1] M. Cai, "The number of vertices of degree k in a minimally k-edge-connected graph," *J. Comb. Theory B, 58*, pp.225–239, 1993.

[2] J. Cheriyan, A. Sebö and Z. Szigeti, "Improving on the 1.5-approximation of a smallest 2-edge connected spanning subgraph," *SIAM J. Disc. Math., 14*, 2, pp.170–180, 2001.

[3] J. Cheriyan and R. Thurimella, "Approximating minimum-size k-connected spanning subgraphs via matching," *SIAM J. Comput., 30*, 2, pp.528–560, 2000.

[4] C.G. Fernandes, "A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem," *J. Algorithms, 28*, 1, pp.105–124, 1998.

[5] A. Frank, "Submodular functions in graph theory," *Discrete Math., 111*, pp.231–243, 1993.

[6] H.N. Gabow, "An ear decomposition approach to approximating the smallest 3-edge connected spanning subgraph of a multigraph," *Proc. 13th Annual ACM-SIAM Symp. on Disc. Algorithms*, 2002, pp.84–93; complete version available at `http://www.cs.colorado.edu/~hal`, submitted for publication.

[7] H.N. Gabow, "On the difficulty of k-connected spanning subgraph problems," unpublished notes.

[8] A.V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *J. ACM 45*, 5, pp.783–798, 1998.

[9] R. Jothi, B. Raghavachari and S. Varadarajan, "A 5/4-approximation algorithm for minimum 2-edge-connectivity," *Proc. 14th Annual ACM-SIAM Symp. on Disc. Algorithms*, 2003, to appear.

[10] D.R. Karger, "Random sampling in cut, flow, and network design problems," *Math. of OR 24*, 2, pp.383–413, 1999.

[11] S. Khuller, "Approximation algorithms for finding highly connected subgraphs," in *Appoximation Algorithms for NP-hard Problems*, D.S. Hochbaum, Ed., PWS Publishing, Boston, MA, 1997.

[12] S. Khuller and B. Raghavachari, "Improved approximation algorithms for uniform connectivity problems," *J. Algorithms 21*, 2, pp.434–450, 1996.

[13] S. Khuller and U. Vishkin, "Biconnectivity approximations and graph carvings," *J. ACM 41*, 2, pp.214–235, 1994.

[14] V. King, S. Rao and R. Tarjan, "A faster deterministic maximum flow algorithm," *J. Algorithms 17*, 3, pp.447–474, 1994.

[15] P. Krysta and V.S. Anil Kumar, "Approximation algorithms for minimum size 2-connectivity problems," in *Proc. 18th Int. Symp. on Theoretical Aspects of Comp. Sci.*, Springer-Verlag, Lecture Notes in Computer Science 2010, pp.431–442, 2001.

[16] H. Nagamochi and T. Ibaraki, "Computing edge-connectivity in multigraphs and capacitated graphs," *SIAM J. on Discrete Math., 5*, 1, pp.54–66, 1992.

[17] R.E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput., 1*, 2, pp.146–160, 1972.

[18] R.E. Tarjan, "Applications of path compression on balanced trees", *J. ACM, 26*, 4, pp.690–715, 1979.

[19] S. Vempala and A. Vetta, "Factor 4/3 approximations for minimum 2-connected subgraphs," in *Appoximation Algorithms for Combinatorial Optimization*, K. Jansen and S. Khuller, Eds., Springer-Verlag, Lecture Notes in Computer Science 1931, pp.262–273, 2000.