A Better Approximation Ratio for the Minimum k-edge-connected Spanning Subgraph Problem

Cristina G. Fernandes *

Abstract

Consider the minimum k-edge-connected spanning subgraph problem: given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with minimum number of edges. This problem is known to be NP-complete. Khuller and Raghavachari presented the first algorithm with a performance ratio smaller than 2 for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm. We improve their analysis, proving that the performance ratio of their algorithm is smaller than 1.7 for large enough k, and that it is at most 1.75 for all k. Our analysis improves the best known ratios for any fixed $k \geq 4$, in particular, for $k = 4$ from 1.75 to 1.65, and for $k = 5$ from 1.733... to 1.68. Last, we show that the minimum k-edge-connected spanning subgraph problem is MAX SNP-hard, even for $k = 2$.

1 Introduction

The study of connectivity in graph theory has important applications in the areas of network reliability and network design. In this paper, we concentrate in the minimum k-edge-connected spanning subgraph problem: given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with minimum number of edges.

This problem is known to be NP-complete [GJ79], even for $k = 2$: if the graph G is Hamiltonian, a 2-edge-connected spanning subgraph of G with minimum number of edges must be a Hamiltonian cycle. So the goal is to look for good polynomial-time approximation algorithms for the problem.

The quality of an approximation algorithm is measured by its so called *approximation* or *performance ratio*. For the minimum k-edge-connected spanning subgraph problem, the approximation ratio of an algorithm is the infimum, over all possible inputs, of the ratio between the number of edges of the output of the algorithm and the number of edges of the optimum.

For a long time, 2 was the best approximation ratio achieved for all k. Just to illustrate, let us quickly describe how to get a ratio of 2. Assume we have a c-edge-connected spanning subgraph H of G, for some $c < k$. Observe that if we add to H the edges of a maximum forest in $G - E(H)$, the resulting spanning subgraph of G is $(c+1)$-edge-connected. Using this fact, we can easily derive a procedure to construct a k-edge-connected spanning subgraph of G. Start with $S = \emptyset$, and repeatedly add to S the edges of a maximum forest in $G-S$. After k iterations, S has at most $k(n-1)$ edges, where n is the number of vertices in G, and it is the edge set of a k-edge-connected spanning subgraph of G. Now, note that any k-edge-connected spanning subgraph of G must have at least $kn/2$ edges, since each vertex must have degree at least k. Thus this procedure leads to a polynomial-time algorithm with approximation ratio at most 2. In fact, there are examples which prove that the performance ratio is exactly 2.

Khuller and Raghavachari [KR95] presented the first algorithm with a performance ratio smaller than a constant smaller than 2 for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm. In this paper, we improve their analysis, proving that the performance ratio of their algorithm is smaller than 1.7 for large enough k, and that it is at most 1.75 for all k.

Previously to [KR95], Karger [K94] presented an algorithm with performance ratio $1 + O(\sqrt{(\log n)/k})$. This is smaller than 2 only when $k \gg \log n$. Also, there were algorithms with approximation ratio smaller than 2, for some particular values of k. An algorithm for $k = 2$ with 1.5 ratio was presented in [KV94]. As observed in [KR95], by combining the biconnectivity algorithm in [KV94], and the sparse certificate algorithm in [CKT93], one can easily obtain a ratio of $2 - 1/k$.

The bound on the approximation ratio given in [KR95] for small values of k is actually better than 1.85: it is 1.5 for $k = 2$, 1.666... for $k = 3$, 1.75 for $k = 4$, 1.733... for $k = 5$, etc. To our knowledge, these were the best known. The bound in [KR95] on the performance ratio of Khuller and Raghavachari's algorithm is tight for $k = 2$ and 3. Our analysis improves their bound for any fixed $k \geq 4$. In particular, we get 1.65 for $k = 4$, and 1.68 for $k = 5$.

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280. E-mail: cris@cc.gatech.edu. Research supported in part by the CNPq (Brazil), under contract 200975/92-7.

Khuller and Vishkin [KV94] have introduced the following concept: a *tree-carving in a graph* $G = (V, E)$ is a partition of the vertex set V into subsets $V_1, \ldots, V_t$ with the following properties. Each subset constitutes a node of a tree Γ. For every vertex v in V_j, all the neighbors of v in G belong either to V_j itself or to V_i, where V_i is adjacent to V_j in the tree Γ. They have used this to prove the 1.5 bound on the ratio of their algorithm for $k = 2$. We generalize the concept of tree-carving, and from this generalization, we derive that the performance ratio of Khuller and Raghavachari's algorithm is smaller than 1.7 for large enough k.

Finally, we show that the minimum k-edge-connected spanning subgraph problem is MAX SNP-hard. This implies that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1 + \epsilon$ would imply that $P = NP$ [ALMSS92].

In the next section, we review some results proved in [KR95], the algorithm, and the analysis. In section 3, we present a better analysis and prove a new upper bound of 1.75 for all k on the performance ratio of the algorithm. Section 4 generalizes the concept of a tree-carving, and shows some properties which are then used to prove that the performance ratio of the algorithm is less than 1.7 for large enough k. The proof of MAX SNP-hardness appears in section 5. Finally, the conclusions are presented in the last section.

2 Preliminaries

Consider a graph G, a maximum depth first search forest F in G, and the post-order numbering of the vertices of G given by F. For each edge e in G, call *top (bottom)* the endpoint of e with biggest (smallest) post-order number. Recall that any edge of G not in F is a *back edge*, that is, its top endpoint is an ancestor in F of its bottom endpoint.

A *cut in a graph* $G = (V, E)$ is denoted by $(X, \overline{X})$, where X is a subset of V and $\overline{X}$ stands for $V - X$. A cut $(X, \overline{X})$ in G consists of the set of all edges of G which have one endpoint in X and the other in $\overline{X}$.

The following lemma is a restatement of lemmas 3.1, 3.2, 3.3 and 3.7 in [KR95]. It permits a better understanding of the algorithm and thus of our analysis. We omit its proof (which is not very complicated).

LEMMA 2.1. *Let k be a positive integer, $G = (V, E)$ be a k-edge-connected graph, i be an integer, $0 \leq i < k - 1$. Let S and F be two disjoint subsets of E such that (V, S) is an i-edge-connected spanning subgraph of G and (V, F) is a maximum depth first search forest in $(V, E - S)$. Then the following hold:*

1. *$(V, S \cup F)$ is $(i + 1)$-edge-connected.*
2. *Any cut in G with exactly $(i + 1)$ edges in $S \cup F$ contains exactly one edge of F.*
3. *Consider a cut in G with $(i + 1)$ edges in $S \cup F$. Let e be the edge of F in this cut. The edges of $E - S$ in this cut are exactly e plus any back edge in $E - S - F$ which has e on the unique path in F between its endpoints.*

2.1 Khuller and Raghavachari's algorithm.

The input for Khuller and Raghavachari's algorithm is a positive integer k and a k-edge-connected graph $G = (V, E)$. Their algorithm works in phases. It consists of $\lfloor k/2 \rfloor$ phases, plus an extra final phase when k is odd. For now, let us assume that k is even. Later we comment on the final phase for odd k.

After phase i, the algorithm will have selected a set of edges S_i of G which induces a $2i$-edge-connected spanning subgraph of G. Hence at the end of $k/2$ iterations, the current set $S_{k/2}$ induces a k-edge-connected spanning subgraph of G which is the output of their algorithm.

How does each phase work? Let $S_0 = \emptyset$. In phase $i \geq 1$, the algorithm chooses two disjoint edge sets F_i and B_i, both disjoint from S_{i-1}. The set F_i is simply the edge set of a maximum depth first search spanning forest in the graph $(V, E - S_{i-1})$. For each edge e in F_i, we call an edge f in $E - (S_{i-1} \cup F_i)$ a *back edge of* e if e lies on the unique path in (V, F_i) between the two endpoints of f. (Note that any edge in $E - (S_{i-1} \cup F_i)$ must have the two endpoints in the same component of F_i, because (V, F_i) is a maximum forest in $(V, E - S_{i-1})$.) The edge set B_i is built as follows: initially $B_i = \emptyset$. Then each edge e in F_i is scanned in post-order. If a minimum cut in G separating the endpoints of e contains exactly $2i - 1$ edges in $S_{i-1} \cup F_i \cup B_i$, then add to B_i a back edge of e which has the biggest post-order numbered top endpoint. (Because $k \geq 2i$, and G is k-edge-connected, there must be some edge of G not in $S_{i-1} \cup F_i \cup B_i$ which is in this minimum cut. And, by 3 of lemma 2.1, this is a back edge of e.)

Next lemma was proved in [KR95] and is basically a corollary of lemma 2.1, so we omit the proof. The correctness of the algorithm is a direct consequence of it.

LEMMA 2.2. *For all i, $0 \leq i \leq k/2$, S_i is a $2i$-edge-connected spanning subgraph of G.*

We need some extra notation for the analysis. Given an edge e in B_i, let t_e be the edge in F_i which made e to be included in B_i. Let $(X_e, \overline{X_e})$ be the cut in G containing t_e with $2i - 1$ edges in $S_{i-1} \cup F_i \cup B'_i$, where B'_i is B_i just before e was included in B_i by the algorithm. Let P_e be the set of edges in $(X_e, \overline{X_e})$ which are in $E - S_{i-1}$. Denote by OPT a k-edge-connected spanning subgraph of G with minimum number of edges, and by *opt* the number of edges in OPT.

Note that, by 3 in lemma 2.1, the cut $(X_e, \overline{X_e})$ contains exactly $2(i-1)$ edges in S_{i-1}, and the rest of the edges are in P_e. Because G is k-edge-connected, G must contain at least $k-2(i-1)$ edges in P_e, for all $e \in B_i$. Next lemma states a key fact, proved in [KR95], essential for the analysis.

LEMMA 2.3. *For any distinct edges e and f in B_i, the edge sets P_e and P_f are disjoint.*

From this lemma, we get that

$$(2.1)\quad (k-2(i-1))|B_i| \leq opt, \quad \text{for all } i, 1 \leq i \leq k/2.$$

We also have that $opt \geq nk/2$, $|F_i| \leq n$, and $|B_i| \leq n$. From all this, Khuller and Raghavachari's upper bound on the performance ratio for even k follows:

$$\begin{aligned}
ratio &= \frac{\sum_{i=1}^{k/2}(|F_i| + |B_i|)}{opt} \\
&= \frac{\sum_{i=1}^{k/2}|F_i| + \sum_{i=1}^{\lceil k/4\rceil}|B_i| + \sum_{i=\lceil k/4\rceil+1}^{k/2}|B_i|}{opt} \\
&\leq \frac{\frac{k}{2}n + \lfloor\frac{k}{4}\rfloor n}{\frac{kn}{2}} + \sum_{i=1}^{\lceil k/4\rceil}\frac{1}{k-2(i-1)} \\
&= 1 + \frac{\lfloor\frac{k}{4}\rfloor}{\frac{k}{2}} + \sum_{i=1}^{\lceil k/4\rceil}\frac{1}{k-2(i-1)} \\
&\leq \frac{3}{2} + \frac{\ln 2}{2} \leq 1.85.
\end{aligned}$$

This completes the review of Khuller and Raghavachari's algorithm and analysis for even k. For odd k, the algorithm runs $\lfloor k/2\rfloor$ phases, and in the end, it runs a "half" phase, computing $F_{\lceil k/2\rceil}$ only (not $B_{\lceil k/2\rceil}$). The final $S_{\lceil k/2\rceil}$ is simply $S_{\lfloor k/2\rfloor} \cup F_{\lceil k/2\rceil}$. The analysis for odd k is similar , so we omit it.

For fixed k, the above analysis gives bounds smaller than 1.85. More specifically, it gives 1.5 for $k=2$, 1.666... for $k=3$, 1.75 for $k=4$ and 1.733... for $k=5$. These bounds are tight for $k=2$ and 3.

In figure 1, we present our tight examples for $k=2$ and 3. Our results originated from attempts of generalizing these examples for $k>3$, and we think they can give some intuition. There are two essentially different types of examples for $k=2$: G_1 and G_2. Denoting by n the number of vertices in each of these graphs, we have that in both cases $opt = n$. Also, if the algorithm chooses as F_1 the dark edges, then the output of the algorithm will have $n + (n-1)/2$ edges (in both cases). As n grows, the ratio between the size of the output and *opt* approaches 1.5. Note that both these examples can be extended for large values of n. Graph G_3 is our tight example for $k=3$ (only

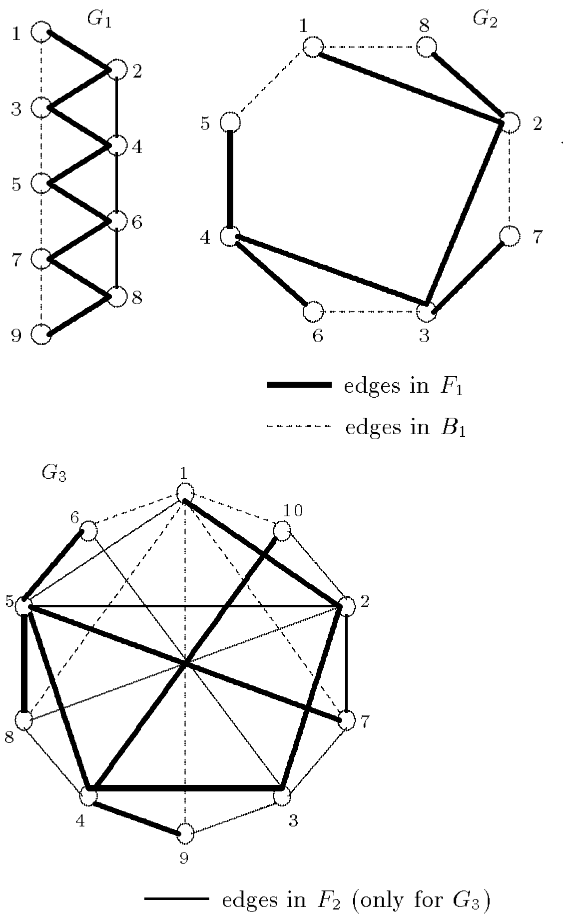

Figure 1: Graphs G_1, G_2 and G_3, with the depth first search numbering given by F_1.

for even n). In this case, OPT can be the outer cycle and the chords from each vertex in the outer cycle to the opposite vertex in the outer cycle. We have that $opt = 3n/2$. Also, assume that the algorithm chooses as F_1 the dark edges. In general, F_1 is built as in G_2: it is a path starting at any vertex in the cycle, and passing by each other vertex in the cycle, until it has visited half of the vertices. Then the remaining vertices are leaves of F_1. Note that F_1 is in fact a depth first search tree. Consider that all edges from the leaves in F_1 to the root are present in G_3, so that these edges are chosen as B_1. In our example, the dashed edges would be B_1. Besides that, the remaining edges of G_3 should contain a tree. In our example, all the remaining edges would be F_2. The output of the algorithm will have $2(n-1) + n/2$ edges. As n grows, the ratio between the size of the output and *opt* approaches $5/3 = 1.666...$ This example can be extended, as suggested, for any large value of n such that $n = 2(2t+1)$ for some positive integer t.

The intuition we would like to get of these examples comes from the following reasoning. For the previous bound to be tight, *opt* must be approximately $kn/2$, and the initial B_i's must have approximately $opt/(k-2(i-1))$ edges. But, for large k, the only way to expect that $|B_i| \approx opt/(k-2(i-1))$, for the initial B_i's, is if F_i has many leaves. However, as we will see, this cannot happen if $opt \approx kn/2$.

3 The 1.75 upper bound on the performance ratio

In this section, we improve the analysis of Khuller and Raghavachari's algorithm by presenting a better upper bound on $|B_i|$.

For each e in B_i, recall that t_e is the edge in F_i which made e to be included in B_i. Denote by b_e the bottom endpoint of t_e. For each i, $1 \leq i \leq \lfloor k/2 \rfloor$, let l_i be the number of b_e's which are leaves of F_i.

Our improvement is first based on the following stronger version of (2.1):

LEMMA 3.1. $(k-2(i-1))|B_i|+(i-1)l_i \leq \mathrm{opt}$, *for all* i, $1 \leq i \leq \lfloor k/2 \rfloor$.

Proof. The number of pairs (u, h), where u is a vertex of G and h is an edge of OPT incident to u, is $2opt$. Next, we compute a lower bound for this number.

For each edge e in B_i, OPT has at least $k-2(i-1)$ edges in P_e. So there are at least $2(k-2(i-1))$ pairs associated to e: two for each of the $k-2(i-1)$ edges of OPT in P_e.

In fact, if b_e is a leaf of F_i, e can be associated to $(k-2(i-1))+k$ pairs. The first $k-2(i-1)$ pairs would be the ones mentioned above whose first coordinate is not b_e. The remaining k pairs have b_e as first coordinate: since OPT is k-edge-connected, there are k edges of OPT incident to b_e.

Let us show that no two edges in B_i have an associated pair in common. As said in lemma 2.3, the sets P_e's are pairwise disjoint, so the pairs coming from P_e's are certainly distinct. The k pairs corresponding to b_e cannot be repeated also, since the only edges in P_f's with b_e as endpoint are in P_e.

Therefore, we must have $l_i((k-2(i-1))+k)+(|B_i|-l_i)2(k-2(i-1)) \leq 2opt$, which is equivalent to the statement of the lemma, concluding the proof. ∎

We will present a lower bound on l_i, for $1 \leq i \leq \lfloor k/2 \rfloor$, which will imply an improved upper bound on $|B_i|$. Let us fix i such that $1 \leq i \leq \lfloor k/2 \rfloor$. For such an i, $|P_e| \geq k-2(i-1) \geq 2$ for any e in B_i. Denote by B the set $\{b_e : e \in B_i\}$. The following fact will be used in the next lemma.

FACT 3.1. *For any* $e, f \in B_i$ *such that* b_f *is a child of* b_e*, there is a child of* b_f *not in* B.

Proof. Consider $e, f \in B_i$ such that b_f is a child of b_e. Because $P_e \cap P_f = \emptyset$, all the edges in P_f must have b_e as their top endpoint. Since $|P_f| \geq 2$, there is at least one edge g in $P_f - \{t_f\}$. The bottom endpoint b_g of g is not b_f (otherwise g would have the same two endpoints of f). Because b_f is a child of b_e, we must have that f was inserted in B_i before e. At the moment f was included in B_i, there could not be other edge h in B_i such that t_h is on the path in F_i between the two endpoints of f. This is true because if, by contradiction, we assume there is an h in B_i such that t_h is on the path in F_i between the two endpoints of f, we must have that $f \in P_h$. But $f \in P_f$ too, contradicting $P_f \cap P_h = \emptyset$. So, the child of b_f on the path from b_f to the bottom endpoint of g is not in B. ∎

The lower bound on l_i is given by the following lemma.

LEMMA 3.2. $l_i \geq 2|B_i|-n$, *for any* i, $1 \leq i < k/2$, *and, for even* k, $l_{k/2} \geq 3|B_{k/2}|-2n$.

Proof. Fix i such that $1 \leq i < k/2$. Denote by N the set of b_e's in B which are not leaves in F_i. We will associate to each b_e in N a distinct vertex z_e in $V-B$. The association is made in two phases:

Phase 1. For each b_e in N which has a child in $V-B$. Let z_e be a child of b_e in $V-B$.

Phase 2. For each b_e in N whose children are all in B. Let $h \in B_i$ be such that b_h is one of b_e's children. By fact 3.1, b_h has a child not in B.

- Case 1: b_h has two or more children not in B. Let z_e be any child of b_h not in B and distinct of z_h.
- Case 2: b_h has exactly one child not in B. Note that b_h satisfies the statement of phase 1, which means that z_h was assigned in phase 1. So this child is in fact z_h. But $i \leq (k-1)/2$, which implies that $|P_h| \geq k-2(i-1) \geq 3$. This means that there is at least one edge g in P_h which has a descendent of z_h as bottom endpoint. Let z_e be the child of z_h on the path from z_h to the bottom endpoint of g. Observe that z_e cannot be in B, otherwise the edge inserted in B_i by the edge whose bottom endpoint is z_e would forbid b_h to be in B.

Let us verify that distinct b_e's are assigned distinct z_e's. In the first phase, clearly no two distinct b_e's are assigned the same z_e. Also no two z_e's assigned in phase 2, case 1, coincide. The same for two z_e's assigned in phase 2, case 2. We explicitly make sure that a z_e assigned in phase 2, case 1, does not coincide with a

z_e assigned in phase 1. A z_e assigned in phase 2, case 2, cannot coincide with a z_e assigned in phase 1, because $z_h \notin B$. Finally, note that in phase 2, case 1, z_e is a grandchild of b_e, while in phase 2, case 2, z_e is a grandgrandchild of b_e. Thus, the only way of phase 2, case 1 and phase 2, case 2 select the same z_e is if b_h in case 2 had its z_h assigned in phase 2, case 1. But, as we have said, b_h had its z_h assigned in phase 1. This completes the proof that we can assign a different vertex in $V - B$ to each vertex in N.

Let $Z = \{z_e : e \in B_i \text{ and } b_e \in N\}$. We have that $|Z| = |N|$, and $|B| = |B_i|$. Moreover, $N \subseteq B$ and $Z \subseteq V - B$. Thus $l_i = |B - N| = |B| - |N| = |B| - |Z| \geq |B| - |V - B| = |B| - (n - |B|) = 2|B| - n = 2|B_i| - n$, completing the proof of the first part of lemma 3.2.

The proof of the second part differs from the above proof only at phase 2, case 2. For $i = k/2$ (even k), we just have $|P_h| \geq 2$. Therefore, z_h might not have a child to be selected as z_e. So, instead of associating to each b_e in N a *distinct* z_e in $V - B$, we associate a z_e which might have already been selected, but at most once. In other words, each $z \in Z = \{z_e : e \in B_i \text{ and } b_e \in N\}$ will correspond to at most two b_e's in N. More specifically, in phase 2, case 2, we select $z_e = z_h$. This implies $|N| \leq 2|Z|$, and we have $l_{k/2} = |B| - |N| \geq |B| - 2|Z| \geq |B| - 2(n - |B|) = 3|B| - 2n = 3|B_{k/2}| - 2n$, completing the proof of the lemma. ∎

Now, using lemma 3.2 and $n \leq 2opt/k$ in lemma 3.1, follows a new upper bound on $|B_i|$.

Corollary 3.1.

$$\begin{aligned} |B_i| &\leq \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) opt, \quad \textit{for} \quad 1 \leq i < k/2. \\ |B_{k/2}| &\leq \frac{2(3k-4)}{k(3k-2)}\, opt. \end{aligned}$$

This bound is not the best we can get using this technique, as we will show in the next section. But it implies the better upper bound of 1.75 on the performance ratio of the algorithm, as stated below.

Theorem 3.1. *The performance ratio of Khuller and Raghavachari's algorithm is at most 1.75 for all k. More specifically, it is at most*

$$1.75 - \frac{3k^2 - 6k + 8}{2k^2(3k-2)}, \textit{ for even } k, \textit{ and}$$

$$1.75 - \frac{2k-3}{4k^2}, \textit{ for odd } k > 1.$$

Proof. For $k = 1$, the algorithm is optimum, i.e., the ratio is 1. For odd $k > 1$,

$$\begin{aligned} ratio &= \frac{\sum_{i=1}^{(k+1)/2} |F_i| + \sum_{i=1}^{(k-1)/2} |B_i|}{opt} \\ &\leq \frac{\frac{k+1}{2} n}{\frac{kn}{2}} + \sum_{i=1}^{(k-1)/2} \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) \\ &= \left(1 + \frac{1}{k}\right) + \frac{k-1}{2k} + \frac{2}{k^2} \sum_{i=1}^{(k-1)/2} (i-1) \\ &= \frac{3}{2} + \frac{1}{2k} + \frac{(k-1)(k-3)}{4k^2} \\ &= \frac{3}{2} + \frac{k^2 - 2k + 3}{4k^2} = \frac{7}{4} - \frac{2k-3}{4k^2}. \end{aligned}$$

The account is similar for even k.

$$\begin{aligned} ratio &= \frac{\sum_{i=1}^{k/2} |F_i| + \sum_{i=1}^{k/2} |B_i|}{opt} \\ &\leq \frac{\frac{k}{2} n}{\frac{kn}{2}} + \sum_{i=1}^{k/2-1} \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) + \frac{2(3k-4)}{k(3k-2)} \\ &= 1 + \left(\frac{1}{2} - \frac{1}{k}\right) + \frac{2}{k^2} \sum_{i=1}^{k/2-1} (i-1) + \frac{2(3k-4)}{k(3k-2)} \\ &= \frac{3}{2} + \frac{k^2 - 10k + 8}{4k^2} + \frac{2(3k-4)}{k(3k-2)} \\ &= \frac{3}{2} + \frac{1}{4} - \frac{5k-4}{2k^2} + \frac{2(3k-4)}{k(3k-2)} \\ &= \frac{7}{4} - \frac{3k^2 - 6k + 8}{2k^2(3k-2)}. \end{aligned}$$

Observe that $2k - 3 \geq 0$ for all $k \geq 3$, $4k^2 > 0$ for all positive k, $3k^2 - 6k + 8 \geq 0$ always, and $2k^2(3k-2) > 0$ for all positive k. Hence the ratio is in fact always smaller than 1.75 for all values of k. ∎

4 Generalized tree-carvings

In this section, we generalize some of the results from the previous section. In particular, we present stronger versions of both lemmas 3.1 and 3.2. From these, we get an even better upper bound on the $|B_i|$'s, and, consequently, on the performance ratio of the algorithm for large values of k.

We start by introducing a generalization of the tree-carving concept and of some results given in [KV94].

Consider a positive integer k, a k-edge-connected graph $G = (V, E)$, a non-negative integer $c < k$, and a subset S of E such that (V, S) is a c-edge-connected spanning subgraph of G.

A *c-tree-carving in G with respect to S* is a partition of the vertex set V into subsets $V_1, V_2, \ldots, V_t$ with the following properties.

1. Each subset consists of a node of a rooted tree Γ. (We will refer to vertices of Γ as *nodes*, and to its edges as *arcs*.)

2. For every vertex v in V_j, all of the neighbors of v in $(V, E-S)$ belong to either V_j itself or to V_i, where V_i is adjacent to V_j in the tree Γ.
3. Each arc e in Γ defines a partition of V into two sets X_e and $\overline{X_e}$ (the deletion of arc e breaks Γ into two trees, Γ_1 and Γ_2, where V_1 belongs to Γ_1. The set X_e is defined to be the union of the sets V_y that belong to Γ_1). The cut $(X_e, \overline{X_e})$ in G must contain exactly c edges of S. Denote by P_e the set of edges in $(X_e, \overline{X_e})$ not in S.

When $c = 0$ and $S = \emptyset$, this definition is the same as the tree-carving definition.

How does this c-tree-carving relate to the algorithm? In fact, the algorithm can be modified so that, at each phase, a $2(i-1)$-tree-carving of G with respect to S_{i-1} is computed. This can be done as follows. Each time an edge f in F_i makes an edge e to be included in B_i, a new set V_j of the $2(i-1)$-tree-carving is defined. This set V_j consists of b_f (the bottom endpoint of f) and all of its descendents in F_i which do not appear in previously V_y's defined in this phase. The parent of this set V_j will be the set V_i containing the top endpoint of f. At the end of a phase, we define the final V_j, the root, to be the set of vertices remaining (i.e., which do not appear in V_y's defined previously in this phase). Observe that the tree corresponding to this $2(i-1)$-tree-carving has exactly $|B_i|$ arcs.

Consider a c-tree-carving $V_1, \ldots, V_t$ of G with respect to S. And let Γ, $(X_e, \overline{X_e})$ and P_e be as in the definition above.

FACT 4.1. *Any k-edge-connected spanning subgraph of G has at least $k-c$ edges in each P_e.*

Proof. Let H be a k-edge-connected spanning subgraph of G. For each arc e in Γ, there must be at least k edges of H in $(X_e, \overline{X_e})$. Exactly c edges in $(X_e, \overline{X_e})$ are in S. Hence, there must be at least $k-c$ edges of H in $(X_e, \overline{X_e})$ not in S, that is, in P_e. ■

FACT 4.2. *Any edge in P_e, for $e = (V_i, V_j)$, has one endpoint in V_i and the other in V_j.*

Proof. Since $P_e \subseteq E - S$, each of its edges has endpoints either in the same V_y or in V_y's adjacent in Γ. But P_e is a subset of $(X_e, \overline{X_e})$, and either $V_y \cap X_e = \emptyset$ or $V_y \cap \overline{X_e} = \emptyset$. So each of its edges cannot have the two endpoints in the same V_y. By the definition of $(X_e, \overline{X_e})$, V_i and V_j are the only nodes in Γ which are adjacent and such that one is a subset of X_e and the other of $\overline{X_e}$. Thus, each edge in P_e must have one endpoint in V_i and the other in V_j. ■

FACT 4.3. *For any distinct arcs e and f in Γ, the edge sets P_e and P_f are disjoint.*

Proof. The arcs e and f can have at most one common endpoint. Therefore, by fact 4.2, their sets P_e and P_f cannot intersect. ■

Denote by l the number of leaves of Γ. The following result is an extension of lemma 3.1.

LEMMA 4.1. *Any k-edge-connected spanning subgraph of G has at least $(k-c)(t-1) + cl/2$ edges.*

Proof. Let H be a k-edge-connected spanning subgraph of G. Let us count the number of pairs (V_y, h), where h is an edge of H in $(V_y, \overline{V_y})$.

For each V_j which is not the root of Γ, let V_i be its parent in Γ, and e be the arc (V_i, V_j). Recall that $P_e \subseteq E - S$. By fact 4.1, H has at least $k-c$ edges in P_e. So there are at least $2(k-c)$ pairs associated to e: two for each of the $k-c$ edges of H in P_e (one with V_i as first coordinate, the other with V_j).

In fact, if V_j is a leaf of Γ, arc e can be associated with $(k-c)+k = 2k-c$ pairs. The $k-c$ first pairs would be the ones mentioned above whose first coordinate is V_i. The remaining k pairs have V_j as first coordinate: since H is k-edge-connected, there are k edges of H in $(V_j, \overline{V_j})$.

Let us show that no two arcs have an associated pair in common. By fact 4.3, the sets P_e's are pairwise disjoint, so the pairs coming from P_e's are certainly distinct. The k pairs corresponding to a leaf V_j cannot be repeated also since V_j is the endpoint of only one arc in Γ.

Denoting by m the number of edges in H, we can have at most $2m$ pairs. Therefore, $2(k-c)(t-1-l) + (2k-c)l \leq 2m$, which implies that $m \geq (k-c)(t-1) + cl/2$, completing the proof of the lemma. ■

The generalization of lemma 3.2 we want to present gives a lower bound on l. But before stating the generalization, we need to prove one more fact.

FACT 4.4. *For each arc $e = (V_i, V_j)$ in Γ, $|V_i| + |V_j| \geq 2\sqrt{k-c}$.*

Proof. Consider the cut $(X_e, \overline{X_e})$ in G corresponding to e. Since G is k-edge-connected, and $(X_e, \overline{X_e})$ contains exactly c edges of S, there must be at least $k-c$ edges of $E-S$ in $(X_e, \overline{X_e})$. These edges must have one endpoint in V_i and other in V_j. The maximum number of edges with one endpoint in V_i and the other in V_j is $|V_i| \cdot |V_j|$. Thus $|V_i| \cdot |V_j| \geq k-c$. This implies that $|V_i| + |V_j| \geq 2\sqrt{k-c}$. ■

Now we can prove the following lower bound on l, the number of leaves of the tree Γ.

LEMMA 4.2.

$$l \geq \left(1 + \frac{2}{\lceil 2\sqrt{k-c} \rceil - 2}\right)t - \frac{2}{\lceil 2\sqrt{k-c} \rceil - 2}\, n.$$

Proof. The proof of this lemma uses basically the same technique used in lemma 3.2. We will associate to each V_i, a set Z_i of vertices in V. The sets will be pairwise disjoint and will contain $\lceil 2\sqrt{k-c}\rceil/2$ vertices, if V_i is not a leaf of Γ, and one vertex, if V_i is a leaf of Γ. The association is done in two phases.

Phase 1. For each V_i which is a leaf or such that $|V_i| \geq \lceil 2\sqrt{k-c}\rceil/2$.
Let Z_i be a subset of V_i with $\lceil 2\sqrt{k-c}\rceil/2$ vertices, if V_i is not a leaf of Γ, and with one vertex, if V_i is a leaf. (Note that $|V_i| \geq 1$ always.)

Phase 2. For each V_i which is not a leaf and such that $|V_i| < \lceil 2\sqrt{k-c}\rceil/2$.
Let $e = (V_i, V_j)$ be an arc in Γ joining V_i to one of its children V_j. By fact 4.4, $|V_i| + |V_j| \geq 2\sqrt{k-c}$, which implies $|V_i| + |V_j| \geq \lceil 2\sqrt{k-c}\rceil$. Thus we must have $|V_j| \geq \lceil 2\sqrt{k-c}\rceil - |V_i| > \lceil 2\sqrt{k-c}\rceil/2$, and so V_j had its set Z_j assigned in phase 1. Choose Z_i to be a subset of $V_i \cup (V_j - Z_j)$ with $\lceil 2\sqrt{k-c}\rceil/2$ vertices. (There is such a subset since $|V_i \cup (V_j - Z_j)| \geq |V_i| + |V_j| - \lceil 2\sqrt{k-c}\rceil/2 \geq \lceil 2\sqrt{k-c}\rceil/2$.)

In phase 1, clearly the assigned sets are pairwise disjoint. It is also clear that a set assigned in phase 2 does not intersect a set assigned in phase 1. By fact 4.4, two sets assigned in phase 2 cannot be associated to adjacent sets V_y's. So they are disjoint.

Since the union of the sets Z_i's is a subset of V, we must have $(t-l)\lceil 2\sqrt{k-c}\rceil/2 + l \leq n$. This means

$$\begin{aligned} l &\geq \frac{\lceil 2\sqrt{k-c}\rceil}{\lceil 2\sqrt{k-c}\rceil - 2} t - \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2}\, n \\ &= \left(1 + \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2}\right) t - \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2}\, n, \end{aligned}$$

concluding the proof of the lemma. ■

As we have said before, from the algorithm, we can extract at each phase a $2(i-1)$-tree-carving of G with respect to S_{i-1} with $|B_i|$ arcs. Recall that l_i is the number of b_e's which are leaves of F_i. Then note that this $2(i-1)$-tree-carving has at least l_i leaves. To simplify the formulas, consider the function h defined as $h(x) = \lceil 2\sqrt{k - 2(x-1)}\rceil$. Applying the lemma above to this $2(i-1)$-tree-carving, we get the following inequality:

$$l_i \geq \left(1 + \frac{2}{h(i)-2}\right)|B_i| - \frac{2}{h(i)-2}\, n. \tag{4.2}$$

Observe that, since $n \geq |B_i|$, we also have

$$l_i \geq \left(1 + \frac{2}{h(j)-2}\right)|B_i| - \frac{2}{h(j)-2}\, n, \tag{4.3}$$

for any $1 \leq i \leq j$.

Using (4.2), (4.3), and lemma 4.1 for OPT, we conclude the new upper bounds on $|B_i|$.

COROLLARY 4.1.

$$|B_i| \leq \frac{k(h(i)-2) + 4(i-1)}{k((k-(i-1))(h(i)-2) + 2(i-1))}\, opt, \tag{4.4}$$

and

$$|B_i| \leq \frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))}\, opt, \tag{4.5}$$

for any $1 \leq i \leq j$.

Using these new upper bounds, we derive our final result.

THEOREM 4.1. *The performance ratio of Khuller and Raghavachari's algorithm, for large enough k, is at most 1.7.*

Proof. We will show the proof for even k. The proof for odd k is similar.

We start by using, for some fixed r, bound (4.4) for $B_{\frac{k}{2}-r+1}, \ldots, B_{\frac{k}{2}}$. For the remaining B_i's, we use bound (4.5) with $j = \frac{k}{2} - r$. From this, we obtain that, for even k,

$$\begin{aligned} ratio &= \frac{\sum_{i=1}^{k/2}(|F_i| + |B_i|)}{opt} \\ &\leq 1 + \sum_{i=1}^{j} \frac{|B_i|}{opt} + \sum_{i=j+1}^{k/2} \frac{|B_i|}{opt} \\ &\leq 1 + \sum_{i=j+1}^{k/2} \frac{k(h(i)-2) + 4(i-1)}{k((k-(i-1))(h(i)-2) + 2(i-1))} \\ &\quad + \sum_{i=1}^{j} \frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))}. \end{aligned}$$

Note that $j+1 \leq i \leq k/2$ and $h(x)$ is decreasing for x, $j+1 \leq x \leq k/2$. Also, recall that $j = k/2 - r$. Therefore,

$$\begin{aligned} ratio &\leq 1 + \sum_{i=j+1}^{k/2} \frac{k(h(j+1)-2) + 4(k/2-1)}{k((k-(k/2-1))(h(k/2)-2) + 2j)} \\ &\quad + \sum_{i=1}^{j} \frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))} \\ &\leq 1 + r\frac{2(kh(j+1)-4)}{k(k+2+4j)} \\ &\quad + \sum_{i=1}^{j} \frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))} \\ &\leq 1 + \frac{2r(kh(j+1)-4)}{k(k+2+4j)} - \frac{4j}{k(h(j)-4)} \end{aligned}$$

$$+\frac{h(j)(h(j)-2)}{h(j)-4}\sum_{i=1}^{j}\frac{1}{(h(j)-2)k-(h(j)-4)(i-1)}$$
$$\leq 1+\frac{2r(kh(j+1)-4)}{k(k+2+4j)}-\frac{4j}{k(h(j)-4)}$$
$$+\frac{h(j)(h(j)-2)}{(h(j)-4)^2}\log\frac{k(h(j)-2)}{(h(j)-2)k-(h(j)-4)j}$$
$$\leq 1+\frac{2r(kh(j+1)-4)}{3k^2+2k-4r}-\frac{2k-4r}{k(h(j)-4)}$$
$$+\frac{h(j)(h(j)-2)}{(h(j)-4)^2}\log\frac{k(h(j)-2)}{h(j)k-2(h(j)-4)r}.$$

Since r is fixed, $h(j)=\lceil 2\sqrt{2r+2}\rceil$, $h(j+1)=\lceil 2\sqrt{2r}\rceil$ are also fixed. And, as k goes to infinity, the bound on the ratio converges to

$$1-\frac{2}{h(j)-4}+\frac{h(j)(h(j)-2)}{(h(j)-4)^2}\log\frac{2(h(j)-2)}{h(j)}.$$

Now, let us make r goes to infinity, which implies that $h(j)=\lceil 2\sqrt{2r+2}\rceil$ goes to infinity as well. The above expression converges to $1+\log 2<1.7$. From this, we finally get that, for large enough k, the performance ratio of the algorithm is at most 1.7. ∎

5 The complexity of the problem

In this section, we show that the minimum k-edge-connected spanning subgraph problem is MAX SNP-hard. This means that there is a constant $\epsilon>0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1+\epsilon$ implies that $P=NP$, by results of Arora et al. [ALMSS92].

As in [PY91], we use the concept of L-reduction, which is a special kind of reduction that preserves approximability. Let A and B be two optimization problems. We say *A L-reduces to B* if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

1. Algorithm f produces an instance $I'=f(I)$ of B, such that the optima of I and I', of costs denoted $opt_A(I)$ and $opt_B(I')$ respectively, satisfy $opt_B(I')\leq\alpha\cdot opt_A(I)$, and
2. Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c-opt_A(I)|\leq\beta\cdot|c'-opt_B(I')|$.

THEOREM 5.1. *The minimum k-edge-connected spanning subgraph problem is MAX SNP-hard.*

Proof. Denote by VC_7 the vertex cover problem restricted to graphs with maximum degree seven. Papadimitriou and Yannakakis [PY91] showed that VC_7 is MAX SNP-hard. We prove that VC_7 L-reduces to the minimum k-edge-connected spanning subgraph problem, here denoted by k-MECSS. The reduction comes from [KRY95], where a directed version of the 2-MECSS is proved to be MAX SNP-hard.

The first part of the L-reduction is a polynomial-time algorithm f and a constant α. Given any instance G of VC_7, f produces an instance H of the 2-MECSS such that the minimum number of edges in a 2-edge-connected spanning subgraph of H, denoted by $opt_{k-MECSS}(2,H)$, is at most α times the minimum size of a vertex cover in G, denoted by $opt_{VC_7}(G)$. In other words, $opt_{k-MECSS}(2,H)\leq\alpha\cdot opt_{VC_7}(G)$.

Let us describe algorithm f. Consider an instance G of VC_7. G is a graph with maximum degree seven. Here is a procedure to construct an instance H of the 2-MECSS. Similarly to [KRY95], start with a special vertex, the root. Each vertex in G will have a "current vertex", initially the root. For each edge uv, add a "cover-testing gadget" to H, as illustrated in figure 2. Specifically, add six new vertices x_1,x_2,x_3,y_1,y_2,y_3. Vertex x_2 is adjacent only to vertices x_1 and x_3. Analogously, vertex y_2 is adjacent only to vertices y_1 and y_3. There is an edge labeled u^- between x_1 and y_3, and an edge labeled v^- between y_1 and x_3. There is one more edge incident to x_1: an edge labeled u^+ between x_1 and the current vertex of u; and one more edge incident to y_1: an edge labeled v^+ between y_1 and the current vertex of v. Make y_3 the new current vertex of u, and x_3 the new current vertex of v. Finally, after all edges of G have been considered, for each vertex v in G, add an edge labeled v^+ between its final current vertex and the root. The gadgets are implicitly numbered in the order we have added them. Clearly H can be obtained in polynomial time in the size of G. This completes the description of f.

Next fact will be used in the proof of the existence of a constant α. Let m be the number of edges in G, and s be a positive integer.

FACT 5.1. *If G has a vertex cover with at most s vertices, then H has a 2-edge-connected spanning subgraph with at most $6m+s$ edges.*

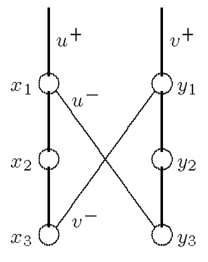

Figure 2: The gadget for edge uv.

Proof. Suppose G has a vertex cover S with at most s vertices. Consider the spanning subgraph H' of H with all edges incident to vertices of type x_2 and y_2, all edges labeled u^+ for $u \in S$, and all edges labeled u^- for $u \notin S$. Observe that all vertices in H'', but the root, have degree two. Besides, the only possible cycles in H' are of two types: either they are like $x_1x_2x_3y_1y_2y_3$, or they include the root. Because S is a vertex cover, no cycles of type $x_1x_2x_3y_1y_2y_3$ can occur. This implies that H is a collection of cycles intersecting only at the root, and spanning all vertices. Therefore, H' is a 2-edge-connected spanning subgraph of H. To count how many edges there are in H', just notice that H' has six edges per gadget plus one extra edge per vertex in S. Therefore, in total, H' has at most $6m + s$ edges. ■

LEMMA 5.1. *$opt_{k-MECSS}(2, H) \leq 43 \cdot opt_{VC_7}(G)$.*

Proof. By fact 5.1, $opt_{k-MECSS}(2, H) \leq 6m + opt_{VC_7}(G)$. Also, because G has maximum degree seven, $opt_{VC_7}(G) \geq m/7$. Putting these together, we get that

$$\begin{aligned} opt_{k-MECSS}(2, H) &\leq 6m + opt_{VC_7}(G) \\ &\leq (6 \cdot 7 + 1) opt_{VC_7}(G) \\ &= 43 \cdot opt_{VC_7}(G). \blacksquare \end{aligned}$$

So, we can consider $\alpha = 43$.

The second part of the L-reduction is the constant β and algorithm g. We may assume that $m \geq 1$. Note that H has $6m + 1$ vertices, therefore any 2-edge-connected spanning subgraph of H must have at least $6m+1$ edges. Consider a 2-edge-connected spanning subgraph H' of H with $6m + s$ edges, where s is some positive integer. Algorithm g will produce in polynomial time a vertex cover of size at most s, from H'. From this, and from fact 5.1, we will have that $opt_{k-MECSS}(2, H) = 6m + opt_{VC_7}(G)$. And then $|6m + s - opt_{k-MECSS}(2, H)| = |6m + s - 6m - opt_{VC_7}(G)| = |s - opt_{VC_7}(G)|$, meaning that $\beta = 1$ suffices.

So let us see how algorithm g works. In a first phase, g produces another 2-edge-connected subgraph H'' of H with at most as many edges as H', and such that in H'' all vertices, but the root, have degree two. To get H'', each gadget is checked for vertices of degree three (all vertices in H, but the root, have degree at most three). If a vertex x has degree three in H', then the edges labeled u^+ and u^- incident to x appear in H'. Remove the edge labeled u^- incident to x and add (if not already there) the edge labeled u^+ incident to the vertex adjacent to x through the edge labeled u^-. See figure 3.

H'' is the graph obtained after applying this modification while there are vertices, other than the root, of degree three. We can make sure the modification is applied to the lowest numbered gadget, and inside the gadget, to x_1, y_1, x_3, y_3, in this order. This procedure takes polynomial time: the modification can be done in polynomial time in the size of G, and after each modification the number of edges labeled u^- decreases. Clearly H'' has at most as many edges as H'.

FACT 5.2. *H'' is a 2-edge-connected spanning subgraph of H.*

Proof. First, let us prove that all cycles in H'' contain the root. Consider a cycle C in H''. Assume, by contradiction, that C does not contain the root. This means that there are gadgets $D_1, D_2, \ldots, D_g$ such that C can be partitioned into paths $P_1, P_2, \ldots, P_g$, where P_i is a maximal (non-trivial) path using only edges of gadget D_i.

Since H' is connected and by the way H'' is constructed from H', for a gadget for an edge uv, we cannot have both edges labeled u^- and v^- in H''. This means C must contain vertices of at least two gadgets, that is, $g > 1$.

Observe that P_i must contain exactly one edge between the two edges labeled u^+ and v^+ in D_i. This is true because all vertices in the gadget have degree two in H'', and C is a cycle in H''. Besides, if the first edge of P_1 is labeled u^+, for some u, then, for all i, the first edge of P_i is labeled u^+, for some u. Analogously, if the last edge of P_1 is labeled u^+, for some u, then, for all i, the last edge of P_i is labeled u^+, for some u. The two cases are similar, so let us assume the first one holds. In this case, the last vertex of each P_i is a vertex of type x_3 or y_3.

Consider two gadgets D and D' such that there is an edge from x_3 or y_3 in D to x_1 or y_1 in D'. Then D' has a number higher than D (that is, D' was added to H after D).

We may assume without loss of generality that D_1 is the lowest numbered gadget among $D_1, D_2, \ldots, D_g$.

For $1 \leq i < g$, the last vertex of P_i, which is of type x_3 or y_3, is connected to a vertex of type x_1 or y_1 in

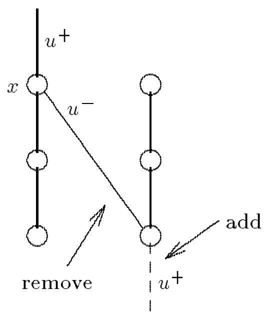

Figure 3: The modification of the gadget for edge uv.

D_{i+1}. Therefore, D_{i+1} has a number higher than D_i. From this, and because $g > 1$, we conclude that D_g has a number higher than D_1. But the last vertex of P_g, which is of type x_3 or y_3, is connected to a vertex of type x_1 or y_1 in D_1, a contradiction. This completes the proof that any cycle in H'' contains the root. And this, plus the fact that all vertices, but the root, have degree two, implies that H'' is a collection of cycles intersecting only at the root, and spanning all vertices of H. Consequently, H'' is a 2-edge-connected spanning subgraph of H. ■

Since all vertices in H'', but the root, have degree two, for each vertex in G, either all edges labeled u^+ or all edges labeled u^- appear in H''. In a second phase, a vertex cover of size at most s is computed: let S be the set of all vertices in H'' whose all edges labeled u^+ appear in H''. In any gadget, there must be edges labeled u^+ appearing in H'', otherwise H'' is not connected. Thus S is a vertex cover in G. Recall that H' had $6m + s$ edges. Hence H'' has at most $6m + s$ edges. But since all vertices but the root have degree two, H'' has $6m + t$ edges, for $t \leq s$, and t is the number of vertices in S. This finishes the description of algorithm g, completing the proof of theorem 5.1. ■

6 Conclusions

In this paper, we have investigated the minimum k-edge-connected spanning subgraph problem, which is known to be NP-complete. The goal is to look for good polynomial-time approximation algorithms. Our paper has two main contributions: first, the best known approximation ratio is reduced from 1.85 to 1.75; and second, the problem is proved to be MAX SNP-hard.

The best known ratio was 1.85 [KR95]. We present an improved analysis of Khuller and Raghavachari's algorithm [KR95], and prove an upper bound of 1.75 on its performance ratio. We also generalize the concept of a tree-carving, and use it to prove that the performance ratio of Khuller and Raghavachari's algorithm is less than 1.7 for large enough k. The MAX SNP-hardness proof guarantees that there is a small $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1+\epsilon$ implies P=NP. We believe that our analysis significantly contributes for a better understanding of the structure of the output of Khuller and Raghavachari's algorithm. We hope that this will help in the development of new algorithms with better performance ratios.

7 Acknowledgments

I would like to thank Vijay Vazirani for suggesting the problem, and S. Khuller for the main reference used on the MAX SNP-hardness proof. I am also grateful to Gruia Calinescu, Naveen Garg, Ion Mandoiu, and Robin Thomas for helpful discussions, comments and suggestions.

References

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Hardness of Approximation Problems," *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, 14–23, 1992.

[CKT93] J. Cheriyan, M. -Y. Kao and R. Thurimella, "Algorithms for Parallel k-vertex Connectivity and Sparse Certificates," *SIAM Journal of Computing*, 22 (1), 157–174, 1993.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., 1979.

[K94] D. Karger, "Random Sampling in Cut, Flow, and Network Design Problems," *Proc. 26th Annual ACM Symposium on the Theory of Computing*, 648–657, 1994.

[KR95] S. Khuller and B. Raghavachari, "Improved Approximation Algorithms for Uniform Connectivity Problems," *Proc. 27th Annual ACM Symposium on the Theory of Computing*, 1995.

[KRY95] S. Khuller, B. Raghavachari and N. Young, "Approximating the Minimum Equivalent Digraph," *SIAM Journal of Computing*, 24 (4), 859–872, 1995.

[KV94] S. Khuller and U. Vishkin, "Biconnectivity Approximations and Graph Carvings," *J. Assoc. Comput. Mach.*, 41 (2), 214–235, 1994.

[PY91] C. H. Papadimitriou and M. Yannakakis. "Optimization, Approximation, and Complexity Classes," *Journal of Computer and System Sciences*, 43:425–440, 1991.