An Improved Approximation Algorithm for Finding the Minimum Biconnected Subgraph

Naveen Garg* Vempala S. Santosh* Aman Singla*

Abstract

We consider the problem of finding the minimum 2-vertex connected spanning subgraph in a given graph. The problem is **NP**-hard. We provide a better technique to lower bound the size of the minimum subgraph. This allows us to achieve an approximation guarantee of $\frac{3}{2}$ thereby improving on an earlier algorithm of Khuller and Vishkin that avhieved a $\frac{5}{3}$-approximation [9].

1 Introduction

Problem [2VC] Given an undirected graph $G = (V, E)$ we wish to find a set of edges, S, such that the subgraph (V, S) is 2-vertex connected and $|S|$ is minimum.

We give algorithms for approximating the minimum 2-vertex connected (2VC) and 2-edge connected (2EC) spanning subgraphs of a given undirected graph. Our algorithms achieve approximation factors of $\frac{3}{2}$ and $\frac{5}{4}$ respectively, whereas the previous best factors were $\frac{5}{3}$ and $\frac{3}{2}$ respectively [9]. These problems have many practical applications, *e.g.* in network design. However, both are NP-hard [8], and hence designing good approximation algorithms is important.

Notice that for each of these problems, 2VC and 2EC, any solution must pick at least n edges - because every vertex must have degree at least 2 in the picked subgraph. An easy scheme for picking the solution with at most $2n - 2$ edges is to do a depth first search with an arbitrary vertex as root, pick all tree edges, and in addition pick the deepest back edge from each non-root vertex. This scheme clearly achieves a factor of 2. Khuller and Vishkin [9] refined the strategy by picking back edges more judiciously. Yet, on certain instances they may end up picking as many as $2n - 2$ edges. However, they introduce elegant concepts of graph carving and tree carving which enables them to give a better lower bound on the optimal, thereby enabling them to prove that their algorithms achieve better factors than 2.

Our algorithm for 2VC does not use the notion of graph carving for lower bounding the optimal solution; we resort to a more familiar graph-theoretic notion for doing this. We note that $\max(n, 2I)$ is a lower bound on the OPT, where I is an independent set in G (Lemma 2.1). While solving 2VC our algorithm also picks an independent set I in G - if the solution found has more than $\frac{3n}{2}$ edges, then the independent set produced is appropriately large so that the solution found is within a factor of $2I$ and hence within a factor of $\frac{3}{2}$ of OPT. We also give an example to show that the

*Dept. of Computer Science and Engg., Indian Institute of Technology, New Delhi

factor claimed is the best possible using this lower bounding technique. This naturally makes one wonder whether there are more sophisticated algorithms that achieve better factors. Indeed, for several natural problems people have hit barriers beyond which no better algorithms have been found, despite much effort. Are these barriers inherent in the problems? We show an example graph G, on n vertices, for which OPT has n edges, and on removing just 1 edge from G, OPT jumps to $\frac{3n}{2}$. Thus small changes in the input graph lead to substantial changes in the the size of the minimum subgraphs. On the other hand, our lower bounding mechanisms are not so sensitive to minor changes. This suggests that we shall need lower bounding techniques that are much more sensitive to do any better.

2 Approximating 2VC

The previous best known algorithm for approximating the minimum 2-vertex connected subgraph achieves a factor $\frac{5}{3}$ approximation [9]. This algorithm uses *graph carvings* to lower bound the optimum. An important contribution of this paper is to lower bound the optimum by a more natural graph-theoretic object viz. the independent set. The notion of graph carvings is used only to pick the edges of the subgraph.

Lemma 2.1 *If a graph has an independent set of size I then the minimum 2-vertex connected spanning subgraph has at least* $\max(n, 2I)$ *edges.*

Proof: The minimum 2-vertex connected spanning subgraph has at least 2 edges incident at each vertex of the independent set. Thus it contains at least $2I$ edges. Since n is also a lower bound, the minimum 2-vertex connected subgraph has at least $\max(n, 2I)$ edges. ■

2.1 The Algorithm.

The algorithm proceeds in two phases. In the first phase we pick a set of edges S, that form a 2-vertex connected spanning subgraph in G. This phase is identical to the algorithm in [9] for picking a 2-vertex connected spanning subgraph. Phase 1 also provides a partition of the vertices into *blocks* which is used in Phase 2 to modify the set S.

2.1.1 Phase 1.

1. Do a depth first search of the graph, label edges as tree and back edges in the usual manner. Let T be the DFS tree. For ease of notation we direct all edges towards the root of T.

2. All the tree edges are included in the subgraph. We now pick an appropriate set of back edges to ensure that the subgraph is 2-vertex connected. This is done as follows

   When backtracking from a vertex v, check if u, the parent of v in T, threatens to be a cut vertex in the currently chosen subgraph. If yes, pick the deepest going back edge from the subtree rooted at v. Also form a block out of all such vertices in the subtree rooted at v that do not belong to any other block.

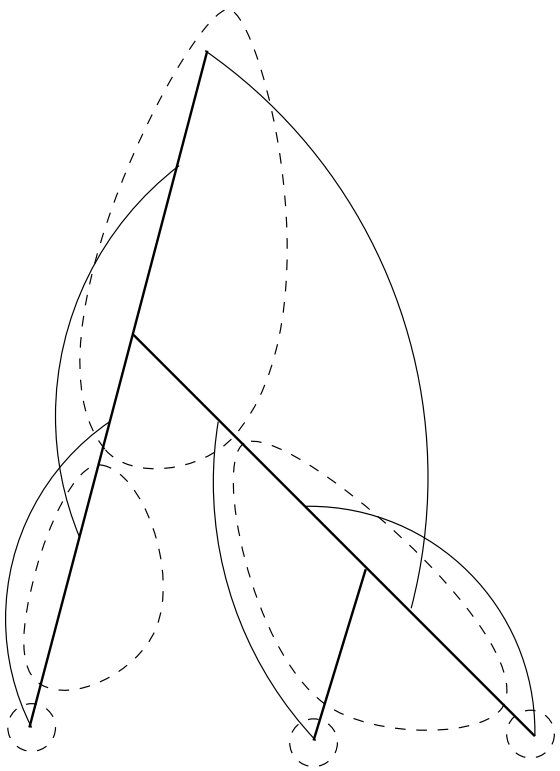

Figure 1:

3. At the end of the DFS, form a block out of all vertices not included in any other block. This block is called the *root block*.

Fig. 1 shows the blocks formed and the subgraph picked at the end of Phase 1.

Lemma 2.2 *The set of edges picked form a 2-vertex connected spanning subgraph.*

We now consider some properties of the blocks.

1. The vertices of a block occur 'together' in the DFS tree. Formally, the tree edges form a spanning tree over each block.

2. Each leaf of T occurs in a block by itself. The root of T is contained in the root block.

Consider the DFS tree T. Coalesce each block into a single vertex. Let T' be the tree so formed (that this is a tree follows from property 1). The vertex corresponding to the root block is the root of T'. We use T' to define parent-child relationships between blocks.

Definition 2.1 The *parent vertex of a block* B, is that vertex in the parent block of B which has a tree edge from B. Clearly, each block except the root has a unique parent vertex.

Lemma 2.3 *All edges of G run, either*

1. *within a block.*
2. *between a block and its parent block.*
3. *between a block and a unique vertex in its grandparent block. This unique vertex is* the parent vertex of the parent block.

The statements of Lemma 2.2 and Lemma 2.3 can be verified from Fig. 1. These Lemmata are proved in [9].

Lemma 2.4 *If a vertex in a block has no edge from any child block then it has no edge from any descendant block.*

Proof: Since the vertex has no edge from any of the child blocks, it is not the parent vertex of any of the child blocks. Hence it shall have no edge from any of the grandchild blocks (Lemma 2.3). ■

2.1.2 Phase 2.

Lemma 2.5 *Let G be a 2-vertex connected graph, C a simple cycle in G, and $e = (u, v)$ a chord in C. Then $G - \{e\}$ is also 2-vertex connected.*

Proof: Assume, for contradiction, that $G - \{e\}$ is not 2-vertex connected. Let w be a cut vertex in $G - \{e\}$ whose deletion forms components G_1 and G_2. Since G is 2-vertex connected, u and v do not belong to the same component G_i, $i = 1, 2$. But now, all paths between u and v use vertex w contradicting the fact that (u, v) is a chord in a simple cycle, C. ■

After the first phase the set S contains a tree edge and a back edge out of each block other than the root. The back edge chosen, is the one that goes deepest. The second phase of the algorithm modifies this choice of back edges in a manner that allows us to drop a tree edge from each block (Lemma 2.5). In case this is not possible, we can find a large independent set to lower bound the optimum.

Rule Each block decides on the back edges going out of its child blocks.

We start with the root block which picks the deepest back edge from each of its child blocks. With the back edge going out of it now decided, the child block, in turn, chooses (in some manner) the back edges going out of its child blocks. In this manner we march from the root towards the leaves picking back edges along the way.

So the key idea is *the manner in which a block decides on the back edges out of its child blocks.*

Consider a block B, other than the root or the leaf blocks (Fig. 2).

Let the back edge out of B, say (v, u), be incident on vertex v in B. Consider the tree path from v to p, the parent vertex of block B. Let s be the first vertex (excluding v) along this path that has no tree edge (other than the edges on the $v - p$ path) incident at it. In case there is no such vertex then s is the same as p. In what follows we denote by

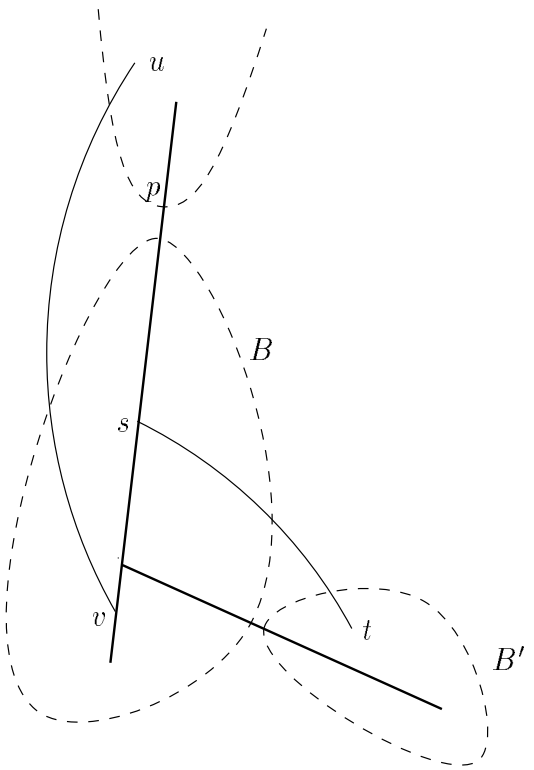

Figure 2:

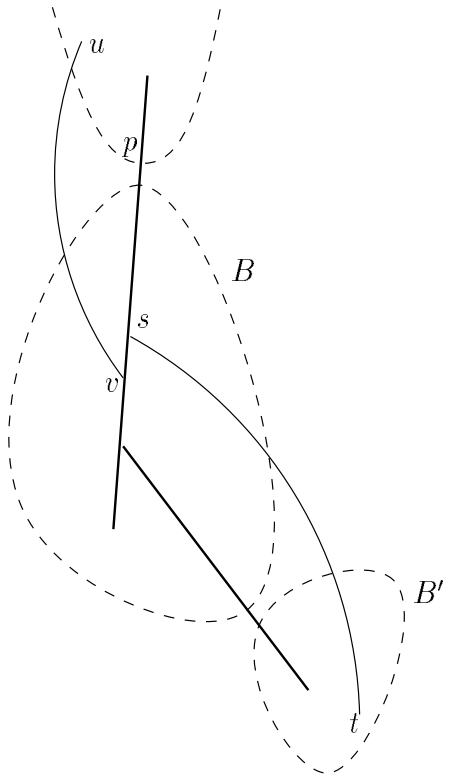

Figure 3:

$u - v$ the edge (u, v).

$u \overset{T}{-} v$ the path from u to v in the tree T.

$u \overset{A}{-} v$ the path from u to v over the vertices in the ancestor blocks (u and v belong to the ancestor blocks).

case 1

s has a back edge (t, s) from a child block B' (Fig. 2).

Instead of picking the deepest going back edge from B' we pick the back edge (t, s). Clearly the subgraph is still 2-connected. For all other child blocks of B we retain the choice made in phase 1 i.e. the deepest going back edge.

case 1.1

s is the parent of v (Fig 3).

Then (s, v) is a chord of the cycle $u - v \overset{T}{-} t - s \overset{T}{-} p \overset{A}{-} u$ and hence can be removed.

case 1.2

s is not the parent of v.

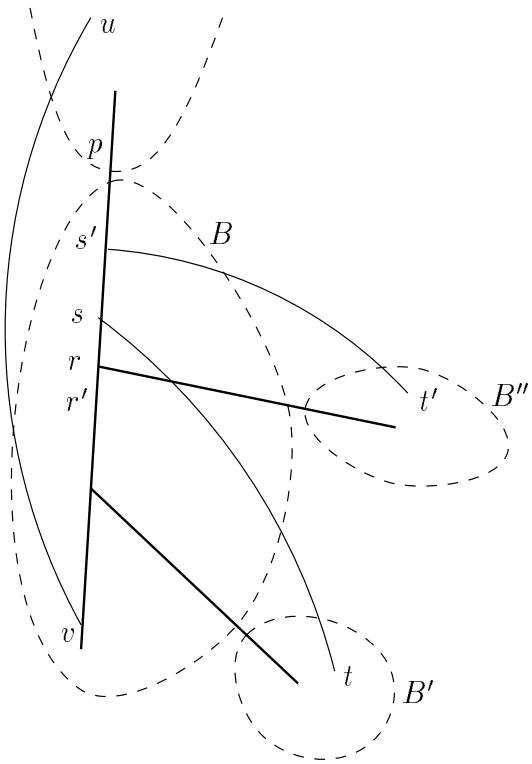

Figure 4:

Let $r' - r - s$ be the order of the vertices on the $v \overset{T}{-} p$ path. Clearly vertex r has a tree edge other than (r', r) and (r, s), incident at it.

case 1.2.1

The path $t \overset{T}{-} s$ uses the edge (r', r) (Fig 4).

Consider all such child blocks of B whose parent vertex lies in the subtree rooted at r but not in the subtree rooted at r'. At least one of these blocks, B'', has its deepest back edge, (t', s'), incident to some vertex s' on the $s \overset{T}{-} p$ path. Then (r, s) is a chord in the cycle $r \overset{T}{-} t - s \overset{T}{-} s' - t' \overset{T}{-} r$ and hence can be removed.

case 1.2.2

The path $t \overset{T}{-} s$ does not use (r', r) (Fig. 5).

Then (r, s) is a chord in the cycle $s \overset{T}{-} p \overset{A}{-} u - v \overset{T}{-} r \overset{T}{-} t - s$ and hence can be removed.

Therefore in this case we can always drop a tree edge. We charge the back edge going out of this block to the tree edge dropped. Thus the back edge going out of this block is for free and so we label the block FREE.

case 2

s has no back edges from any child block.

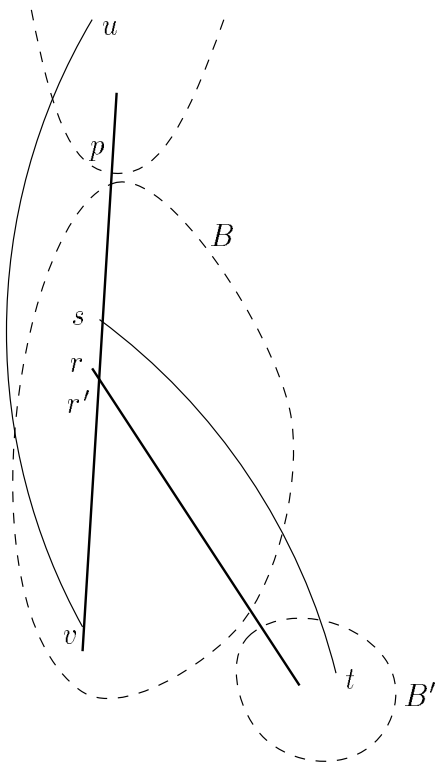

Figure 5:

By our choice of vertex s it follows that s has no tree edge from any child block. Thus s has no edge from any child block. Hence by Lemma 2.3, s has no edge from any offspring block. We mark the vertex s and label the block as MARKED.

The root block has no back edge going out of it and hence is labelled FREE. Each leaf block contains a single vertex which is a leaf of the DFS tree T. We mark this vertex and label the leaf block MARKED.

Lemma 2.6 *Each block is either labelled* MARKED *or* FREE.

Lemma 2.7 *The set of marked vertices form an independent set in G.*

Proof: No two marked vertices belong to the same block. Further, a marked vertex has no edge from any descendant block of the block that contains it. Thus no two marked vertices can have an edge between them. ■

2.2 The Analysis.

Theorem 2.8 *The total number of edges picked by this method is at most* $\frac{3}{2}$OPT.

Proof: If k is the total number of blocks, F the number of blocks labelled FREE, and M the number labelled MARKED, then by Lemma 2.6,

$$k = M + F$$

The number of edges picked by the algorithm is

$$n - 1 + k - F = n - 1 + M$$

As G has an independent set of size M (Lemma 2.7), the optimal 2-connected spanning subgraph has at least $\max(n, 2M)$ edges (Lemma 2.1). Thus our algorithm achieves an approximation factor of

$$\frac{n - 1 + M}{\max(n, 2M)}$$

which is at most $\frac{3}{2}$. ■

It is easy to see that the running time complexity of the algorithm is $O(E)$.

3 Tight examples

In this section we present an example to show that the factor obtained is the best possible for the lower bounding technique used.

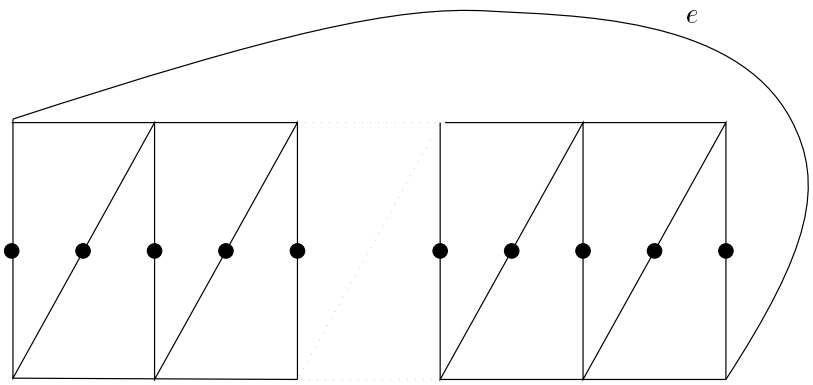

Figure 6:

The graph in Fig. 6 (without the edge e) is a minimal 2-vertex connected graph with $\frac{3n}{2}$ edges. The best lower bound on the OPT that we can show for this example is n, as we can get an independent set of at most $\frac{n}{2}$ vertices. Other methods of obtaining lower bounds are also of not much use. This is to be expected as by the addition of just one edge (e in Fig. 6) the graph becomes Hamiltonian. All known techniques for obtaining good lower bounds are local in the sense that small changes in the graphs do not effect the lower bound appreciably. Thus a better scheme for approximating the minimum 2-vertex connected subgraph shall require a much more sophisticated lower bounding technique that is more global in nature.

Acknowledgements The problem arose in a course on Approximation Algorithms at IITD. We thank Prof. Vijay Vazirani for his constant encouragement and the numerous pointers that helped us considerably. Prof. Vazirani also meticulously went through earlier drafts of the paper.

Numerous discussions with Vivek Arora, Prof. Huzur Saran and Suren Reddy provided us with useful insights. We gratefully acknowledge their help.

References

[1] N. Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*, Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[2] J. Edmonds, *Matroid intersection*, Annals of Discrete Mathematics, 4 (1979), pp. 185-204.

[3] K. P. Eswaran and R. E. Tarjan, *Augmentation problems*, SIAM Journal on Computing, Vol. 5, 4 (1976), pp. 653-665.

[4] G. N. Frederickson and J. JáJá, *On the relationship between biconnectivity augmentation and the travelling salesman problem*, Theoretical Computer Science, 19 (1982), pp. 189-201.

[5] A. Frank, *Augmenting graphs to meet edge connectivity requirements*, 31^{st} Annual Symposium on Foundations of Computer Science, (1990), pp. 708-718.

[6] A. Frank and E. Tardos, *An application of submodular flows*, Linear Algebra and its Applications, 114/115 (1989), pp. 320-348.

[7] H. N. Gabow, *A matroid approach to finding edge connectivity and packing arborescences*, 23^{rd} Annual Symposium on Theory of Computing, (1991), pp. 112-122.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, 1978.

[9] S. Khuller and U. Vishkin, *Biconnectivity approximations and graph carvings*, 24^{th} Annual Symposium on Theory of Computing, (1992).

[10] D. Naor, D. Gusfield and C. Martel, *A fast algorithm for optimally increasing the edge connectivity*, 31^{st} Annual Symposium on Foundations of Computer Science, (1990), pp. 698-707.

[11] D. P. Williamson, M. X. Goemans, M. Mihail, V. V. Vazirani, *A primal-dual approximation algorithm for generalized steiner network problems*, to appear, (1992).