Alcom-FT Technical Report Series
ALCOMFT-TR-01-127

Approximation Algorithms for Minimum Size 2-Connectivity Problems*

Piotr Krysta** and V. S. Anil Kumar

Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany.
{krysta,kumar}@mpi-sb.mpg.de.

Abstract. We study some versions of the problem of finding the minimum size 2-connected subgraph. This problem is NP-hard (even on cubic planar graphs) and MAX SNP-hard. We show that the minimum 2-edge connected subgraph problem can be approximated to within $\frac{4}{3} - \epsilon$ for general graphs, improving upon the recent result of Vempala and Vetta [14]. Better approximations are obtained for planar graphs and for cubic graphs. We also consider the generalization, where requirements of 1 or 2 edge or vertex disjoint paths are specified between every pair of vertices, and the aim is to find a minimum subgraph satisfying these requirements. We show that this problem can be approximated within $\frac{3}{2}$, generalizing earlier results for 2-connectivity. We also analyse the classical local optimization heuristics. For cubic graphs, our results imply a new upper bound on the integrality gap of the linear programming formulation for the 2-edge connectivity problem.

1 Introduction

Graph connectivity is an important topic in theory and practice. It finds applications in the design of computer and telecommunication networks, and in the design of transportation systems. Networks with certain level of connectivity, which intuitively means that they provide certain number of connections between sites, are able to maintain reliable communication between sites, even when some of the network elements fail. For a survey and further applications, see Grötschel *et al.* [9].

Problem statement. Given a graph with weights on its edges, and an integral *connectivity requirement* function r_{uv} for each pair of vertices u and v, the *vertex connectivity* (*edge connectivity*, respectively) *survivable network design problem* (SNDP) is to find a minimum weight subgraph containing at least r_{uv} vertex (edge, respectively) disjoint paths between each pair u, v of vertices. If $r_{uv} \in X$ for some set X, for each pair u, v, we denote the problem as X-VC-SNDP (X-EC-SNDP, respectively). The term survivable refers to the fact that the network is tolerant to the failures of sites and links (in case

* Partially supported by the IST Program of the EU under contract number IST-1999-14186 (ALCOM-FT).

** The author was supported by Deutsche Forschungsgemeinschaft (DFG) Graduate Scholarship. Part of the work by this author was done while he was visiting the Combinatorics & Optimization Dept., University of Waterloo, Ontario, Canada, during January-March, 2000, and was partially supported by NSERC grant no. OGP0138432 of Joseph Cheriyan.

of VC-SNDP) or links (for EC-SNDP). Even the simplest versions of these problems are NP-hard, and so approximation algorithms[1] are of interest.

Previous results for general cases. For $\mathcal{N}_{\geq 0}$-EC-SNDP with arbitrary edge weights, there is a $O(\log(r_{max}))$-approximation by Goemans *et al.* [6] ($r_{\max} = \max_{u,v}(r_{uv})$), which was recently improved to 2 by Jain [10]. No algorithm with a non-trivial guarantee is known for the general version of VC-SNDP. For $\{0, 1, 2\}$-VC-SNDP with arbitrary edge weights, Ravi and Williamson [13] gave a 3-approximation algorithm.

Unweighted low-connectivity problems. The case of low connectivity requirements is of particular importance, as in practice networks have rather small connectivities. There has been intense research for problems with low connectivity requirements and all the edge weights being equal to one (*unweighted problems*) [2, 5, 11, 14]. We focus on the special unweighted cases of this problem where each $r_{uv} \in \{1, 2\}$. These are the simplest non-trivial versions of this problem and have been studied for a long time, but tight approximation guarantees and inapproximability results are not fully understood.

For the unweighted $\{2\}$-EC-SNDP (or 2-EC) Khuller and Vishkin [11] gave a $\frac{3}{2}$-approximation, which was improved by Cheriyan *et al.* [2] to $\frac{17}{12}$, and recently to $\frac{4}{3}$ by Vempala and Vetta [14]. For the unweighted $\{2\}$-VC-SNDP (or 2-VC) Khuller and Vishkin [11] gave an algorithm with approximation guarantee of $\frac{5}{3}$, which was improved to $\frac{3}{2}$ by Garg *et al.* [5] and to $\frac{4}{3}$ by Vempala and Vetta [14]. Both unweighted 2-VC and 2-EC problems are NP-hard even on cubic planar graphs, and also MAX SNP-hard [3]. For both unweighted problems $\{1, 2, \ldots, k\}$-VC-SNDP and $\{1, 2, \ldots, k\}$-EC-SNDP, the results of Nagamochi and Ibaraki [12] imply k-approximation algorithms.

The Linear Programming (LP) relaxation for the 2-EC problem and the subtour relaxation for TSP are very closely related [2]. The approximation ratio of $\frac{4}{3}$ obtained by Vempala and Vetta [14] has a special significance, because of the connections with the $\frac{4}{3}$ conjecture for metric TSP [1, 2]. In this regard, the issue of whether $\frac{4}{3}$ can be improved for 2-EC is an interesting question. Also, the integrality gap[2] of the LP relaxation for 2-EC is not well understood. Vempala and Vetta [14] say that their result does not imply the same bound on the integrality gap for 2-EC. We prove a bound of better than $\frac{4}{3}$ on the integrality gap for cubic graphs, i.e. graphs with maximum degree at most 3.

Little is known about vertex-connectivity generalizations where arbitrary requirements are allowed, even for unweighted graphs. The simplest such generalization is to allow requirements of 1 or 2, instead of 2 for every pair. It should be noted that if the requirement can also take value of *zero*, the unweighted and weighted problems are essentially identical: an edge with an integer weight w can be replaced by a path of Steiner vertices of length w. For instance, unweighted $\{0, 1, 2\}$-VC-SNDP is equivalent to the weighted $\{0, 1, 2\}$-VC-SNDP considered by Ravi and Williamson [13].

Our Contributions. We give improved approximation algorithms for the 2-edge connectivity (2-EC) and the $\{1, 2\}$-connectivity problems. We show a $(\frac{4}{3} - \epsilon)$-approximation

[1] A polynomial time algorithm is called an α-*approximation algorithm*, or is said to achieve an *approximation (or performance) guarantee* of α, if it finds a solution of weight at most α times the weight of an optimal solution. α is also called an *approximation ratio (factor)*.

[2] The LP for the unweighted 2-EC is $\min\{\sum_{e \in E} x_e : \sum_{e \in \delta(S)} x_e \geq 2, \forall S \subset V, S \neq \emptyset; x_e \geq 0, \forall e \in E\}$, where $\delta(S)$ is the set of edges with exactly one end vertex in S. The ratio of the optimum integral solution value of 2-EC to the value of the LP, is called the *integrality gap*.

algorithm for 2-EC on general graphs, where $\epsilon = \frac{1}{1344}$. Our algorithm extends the technique of Vempala and Vetta [14] and removes the bottlenecks mentioned in their paper. The main new ideas are a better charging scheme, a refined lower bound, and the use of a local search heuristic. We show a tight example for their lower bound and their analysis, which leads naturally to the lower bound we use. The improved approximation is obtained by proving that paths can be connected for better charge by a more careful analysis, which is of comparable complexity as theirs, but is more uniform because it only deals with paths. Since, it is unlikely that an approximation scheme exists for the 2-EC problem (because of the MAX SNP-hardness), finding the best possible approximation guarantee for this classical problem is interesting (Section 3).

We achieve better guarantees for special classes of graphs, on which 2-EC is still NP-hard. For planar graphs, we show for 2-EC a $\frac{17}{13}$-approximation, but in quasi-polynomial time, by computing a stronger lower bound (Section 4). For 2-EC on cubic graphs we obtain a $\frac{21}{16}$-approximation using a simple local search heuristic. This implies an integrality gap of at most $\frac{21}{16}$ for the standard LP on cubic graphs (Section 5).

For the $\{1,2\}$-VC-SNDP and $\{1,2\}$-EC-SNDP (henceforth denoted by $\{1,2\}$-VC and $\{1,2\}$-EC), we give a $\frac{3}{2}$ approximation. This improves on straightforward 2-approximation. Our algorithms are generalizations of the algorithms of Garg *et al.* [5] and of Khuller and Vishkin [11]. The lower bounds used in [5, 11] do not apply to our problems and we generalize them appropriately. We also analyse the performance of the classical local optimization heuristics (Section 6). Finally, Section 7 has some conclusions. Most of the details and proofs are missing in this abstract, and will appear in the full version.

2 Preliminaries

Graph Theory. We consider only undirected simple graphs. Given a graph $G = (V, E)$, we write $V(G) = V$ (*vertices*) and $E(G) = E$ (*edges*). Sets of vertices or contracted subgraphs will sometimes be called *(super) nodes*. C_l denotes a cycle of length l. A $u{-}v$ *path* is a path with end vertices u, v. $d_G(v)$ is the degree of vertex v in G. For definitions of the following standard graph theory notions the reader is referred to [7]: *cut vertex*, *two vertices separated by a cut vertex*, *bridge* (or *cut edge*), *ear decomposition* $\mathcal{E} = \{Q_0, Q_1, \ldots, Q_k\}$ (Q_0 is just one vertex), *ears* (Q_i's), *ℓ-ear* (an ear with ℓ edges), *open/closed ear* and *ear decomposition*. If a graph is 2-vertex(edge)-connected, then we write that it is 2-VC(EC). It is well known, that a graph is 2-EC (2-VC) iff it has no bridge (cut vertex, resp.), and iff it has an (open, resp.) ear decomposition. Also, an (open) ear decomposition can be found in polynomial time.

Let $\mathcal{E}$ be an ear decomposition of a 2-connected graph. We call an ear $S \in \mathcal{E}$ of length ≥ 2 *pendant* if none of the internal vertices of S is an end vertex of another ear $T \in \mathcal{E}$ of length ≥ 2. Let $\mathcal{E}' \subseteq \mathcal{E}$ be a subset of ears of the ear decomposition $\mathcal{E}$. We say that set $\mathcal{E}'$ is *terminal* in $\mathcal{E}$ if: (1) every ear in $\mathcal{E}'$ is a pendant ear of $\mathcal{E}$, (2) for every pair of ears $S, T \in \mathcal{E}'$ there is no edge between an internal vertex of S and an internal vertex of T, and (3) every ear in $\mathcal{E}'$ is open.

Given a rooted tree T, $[a, b]$ denotes the $a - b$ path in tree T for some two vertices a, b such that b is an ancestor of a, and path $[a, b]$ contains both vertices a, b. We define $[a, b)$ (and $(a, b]$, resp.) similarly, but the path contains a (b, resp.) and does not b (a, resp.). If a is a proper descendant of b in T, we say that a is *below* or *lower than* b, and b is *above* or *higher than* a. A vertex in T is also descendant and ancestor of itself. *opt*(G) or *opt* denotes the size of an optimal solution on G to the problem under consideration.

Preliminary Reductions. Given a graph $G = (V, E)$, the problem of finding a minimum size subgraph G' of G in which every vertex of V has degree at least 2, is called $D2$. We refer to G' as $D2$, or $D2$ solution on G. This problem can be solved exactly in polynomial time [14], and it gives a lower bound for both 2-EC and 2-VC. The graph for 2-EC can be assumed to have no pair of adjacent degree 2 vertices (cf. [14]), and no cut vertices (else, we can solve 2-EC separately on each 2-VC component).

We define now *beta structures* [14]. A vertex u is a *beta vertex* if deleting some two adjacent vertices v_1, v_2 leaves at least 3 components, one of which is just u (Fig. $1(a)$). Two adjacent vertices u_1, u_2 are called *beta pair* if there are two other vertices v_1, v_2, whose removal leaves at least 3 components, one of which is just u_1 and u_2 (Fig. $1(b)$). Fig. $1(b)$ shows all the four edges (v_1, u_1), (v_1, u_2), (v_2, u_1), (v_2, u_2), but it may be the case for a beta pair that just three of them are present. A graph with no beta vertex or beta pair is called *beta-free*. Vempala and Vetta [14] show that any α-approximation algorithm for 2-EC on beta-free graphs can be turned into an α-approximation algorithm for 2-EC on general graphs. Thus, we can assume for 2-EC that the graph is beta-free.

Let $C = C_l$ ($l \geq 3$) be a given cycle in G. If any solution to 2-EC on G uses at least l' edges from the subgraph induced on $V(C)$, then we can *contract cycle* C (i.e. identify the vertices in $V(C)$ and delete self loops) into a super node and solve recursively 2-EC on the resulting graph, incurring a factor of $\frac{l}{l'}$ as in [14]. The solution to 2-EC on G will then be the union of $E(C)$ with the edges of the recursive solution. l' here is used as a "local" lower bound. The overall approximation ratio of such an algorithm is $\geq l/l'$.

Let T be a given rooted spanning tree, some of whose vertices might be super nodes corresponding to subgraphs of the input graph. For a given non-root super node N of T the tree edge to the parent of N is called the *upper tree edge* of N. If N is a non-leaf super node, then any of the tree edges to its children is called the *lower tree edge* of N.

Local Optimization Heuristics. We define here a general local optimization heuristic. Let Π be a minimization problem on $G = (V, E)$, where we want to find a spanning subgraph of G with minimum number of edges, which is feasible for (or w.r.t.) problem Π. Given a non-negative integer j and any feasible solution $H \subseteq G$ to problem Π, we define the *j-opt heuristic* as the algorithm which repeats, if possible, the following:

- if there are subsets $E_0 \subseteq E \setminus E(H), E_1 \subseteq E(H)$ ($|E_0| \leq j$, $|E_1| > |E_0|$) such that $(H \setminus E_1) \cup E_0$ is feasible w.r.t. Π, then set $H \leftarrow (H \setminus E_1) \cup E_0$.

The algorithm outputs H, if it can perform no such operation on H any more. We say that such output solution is *j-opt* (or *j-optimal*) w.r.t. Π. If $|E_0| = j$, then we call the operation above a *j-opt exchange*. The algorithm can be implemented to run in polynomial time when j is a fixed constant.

3 Approximating 2-Edge Connectivity: General Graphs

We start by considering the algorithm of Vempala and Vetta [14], which gives a $\frac{4}{3}$ approximation for 2-edge connectivity problem. They use $D2$ as the lower bound. The beta-free example in Fig. $1(c)$ shows that their lower bound is tight. $T_1, \ldots, T_k$ are 3-cycles connected to vertices v, w of C, and C is a small clique. Clearly, the optimum must use 4 edges for each T_i, and asymptotically, $|OPT| = \frac{4n}{3}$, where n is the number of vertices. This suggests that we should use a modified lower bound of $\max(|D2|, 4l)$, where l is the number of such disjoint triangles.

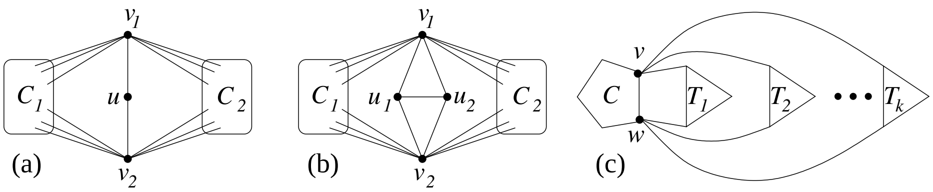

Fig. 1. (a), (b) Beta-structures. (c) A tight example for the lower bound in [14].

Our algorithm is an extension and refinement of that of Vempala and Vetta [14]. The main differences in our algorithm are:

- In a necessary preprocessing step, we use a simple local optimization heuristic to eliminate some complicated configurations.
- We observe that paths can be connected up with charge less than $\frac{1}{3}$ by partly charging their child nodes and by using the fact that neither beta-vertices nor cut vertices are present. The analysis of [14] mentions that one of the bottlenecks in their $\frac{4}{3}$ approximation is the charge for 2-connecting paths. The reason that they only get a charge of $\frac{1}{3}$ is because they only use the property that cut vertices are missing.
- We actually obtain a stronger property: paths can not only be connected up with charge less than $\frac{1}{3}$, they can also pay for an extra $\frac{5}{14}$, with charge remaining less than $\frac{1}{3}$ (in some cases, such paths can only pay for an extra $\frac{3}{14}$). The algorithm first computes some DFS spanning tree of the graph, different than the DFS spanning tree in [14]. Then it 2-connects the tree in a top-down manner ([14] does this bottom-up). This is critical, since it allows us to charge the added edges to children.
- Paths whose parent is incident at one end, can be 2-connected with charge better than $\frac{1}{3}$, even if one free edge is not available, by partly charging the children.
- Another bottleneck in [14] are 3-cycles and a $\frac{1}{3}$ charge for these seems inevitable in their analysis. We overcome this by either merging 3-cycles with their children to form longer paths, or charge them with their children. For leaf 3-cycles, we use a stronger lower bound of $\max(|D2|, 4l)$, where l is the number of such leaves.

3.1 The Algorithm

A description of our algorithm is given below.

1. Compute $D2$ solution and partition it into paths and 2-EC components as in [14]. See Section 3.2 below for details.
2. Run a 1-opt heuristic on the $D2$ solution w.r.t. $D2$, and replace that current $D2$ solution with the 1-opt one.

3. Transform each 2-EC component into a cycle of length at least 3 by contracting suitable vertices and edges. See Section 3.2 for details.
4. Shrink all paths in $D2$ into single nodes.
5. Perform DFS on the contracted graph with the following rule for deciding the order of exploring neighbors on entering a node.
   - On entering a path: Suppose that the parent enters this path node at vertex v. Then start DFS from the end of the path that is farther away from v, and proceed till the other end.
   - On entering a cycle $C = v_1, \ldots, v_k$: Traverse the whole C first in the order $v_1, v_2, \ldots, v_{k-6}, v_{\pi(k-5)}, \ldots, v_{\pi(k)}$, where $\pi(k-5), \ldots, \pi(k)$ is a permutation of $k-5, \ldots, k$. The π is chosen such that: (i) one can traverse the cycle in this order, and (ii) if possible, $v_{\pi(k)}$ is a vertex leading to an unvisited vertex. That is, of all the possible traversals of the last 6 vertices of the cycle, choose one that allows the last vertex to have a child. If the child is another cycle, we get a longer path. If no such $v_{\pi(k)}$ exists, any permutation is equally good.

   Uncontract the nodes representing paths in $D2$, and let T be the resulting DFS tree.
6. Decompose T into maximal paths in the following manner: paths in $D2$ remain paths now, and we call them *red paths*. All edges of a cycle are part of one path. If a collection of cycles or paths can be merged, they will form a new path. Paths corresponding only to cycles are called *blue paths*, whereas a path formed by merging cycles and paths of $D2$ is a red path. The only difference between these is that a red path has two free edges (cf. Section 3.2 and [14]). Consider the natural parent child relation between these paths. The parent of a blue path is incident to it at one end.
7. Consider the paths in a top down manner. Each path is connected using extra edges. For a blue path, 2-connecting up just means connecting it within, since one end of it is part of the parent. 2-connecting paths involves forming blocks (Section 3.3).

The next subsection describes the way edges are added and charged for each path. The charge on each edge of each path other than blue paths of length three at leaves is at most $\frac{37}{112} < \frac{1}{3}$. Since our lower bound is $\max(D2, 4l)$, where l is the number of blue paths of length three at leaves, the approximation factor is $1 + \frac{(D2-3l)\alpha+3l\cdot(1/3)}{\max(D2,4l)} \leq 1 + \frac{D2\alpha+l(1-3\alpha)}{\max(D2,4l)} \leq 1 + \alpha + \frac{1-3\alpha}{4} = 1 + \frac{1+\alpha}{4}$. For $\alpha = \frac{37}{112}$, the factor is at most $1 + \frac{149}{448} < \frac{4}{3}$. This gives us the following theorem.

Theorem 1. *The minimum 2-edge connected subgraph problem on unweighted graphs can be approximated within a factor of $\frac{4}{3} - \epsilon$, where $\epsilon = \frac{1}{1344}$, in time $O(n^{2.5})$.*

Remark. The running time, as in [14], is the time required to find the $D2$. If $D2$ has no C_3 or C_4, then our proof shows that the charge on $D2$ is at most $\frac{23}{70}$ (cf. Section 4).

3.2 Preprocessing Details

Partitioning $D2$. We partition the $D2$ solution as described in [14], as follows. Partition $D2$ into connected components. Each such component is formed by 2-EC components, connected by paths. By an easy reduction, we can assume that any 2-EC component with at least 4 $D2$ edges contains no C_3.

Let us fix now some connected component C in $D2$. We will partition any such connected component C as follows. Any maximal 2-EC component of C is called a *supernode*. Contract all the maximal 2-EC components of C into single nodes. Partition the resulting tree (from C) into maximal paths arbitrarily. Then, any of these paths is also called a *supernode*. The key point here is that any such path supernode has two associated "free" edges – the first and last edge of the path, which we call *free edges*.

Contracting into Cycles. We contract parts of each 2-EC component and solve the problem on the resulting graph. The following lemma describes properties of the resulting 2-EC components. The local search in step 2 of our algorithm is used in its proof.

Lemma 1. *The contracted final subgraph resulting from a 2-EC component is a simple cycle of length at least three. Moreover, each contracted super node contains inside at least four edges of the $D2$ solution.*

3.3 2-Connecting Paths

The paths are 2-connected by adding extra edges. A path is partitioned into blocks, and one extra edge is used for each block, and the cost of this edge is charged to the vertices of the block. This is sufficient if the blocks have length at least 4. For smaller blocks we charge the children also. At a high level, 2-connecting a path node N is as follows. We process the path in the direction opposite to the direction of DFS (the matter of direction is critical). Assume that vertices up to $u_0 \in N$ have been 2-connected already, and the subsequent vertices are $u_1, u_2, \ldots, u_k$, with u_k being the last vertex of N. Contract all the nodes not in the subtree rooted at N, and the vertices of N up to u_0 into u_0. Now consider the farthest going edge e from u_0. We have the following cases.

1. e is incident on $u_i, i \geq 4$: The collection of vertices from u_1 to the other end of e forms a block of size at least 4, and we add e to 2-connect this block.
2. e is incident on u_3: If $k = 3$, this is the last block of N and is connected differently (see below). Else, there must be an edge f from u_1 or u_2 beyond u_3, to prevent it from being a cut vertex. If f is from u_2 to some vertex beyond u_3, add e, f and delete (u_2, u_3). The vertices from u_1 till the other end of f form a block of size at least 4, and a total of 1 edge is used for this block. Now let f be from u_1 to a vertex beyond u_3, and let there be no edge from u_2 to u_0, or beyond u_3. Then u_2 is a beta-vertex, and there must be a child node C from u_2. Vertices u_1, u_2, u_3 form a block of length 3, and we add e, and partly charge C, to reduce the charge.
3. e is incident on u_2: If e goes to an ancestor, and the upper tree edge is incident on u_2, N has to be a red path and we form a block of length 2 (vertices u_1, u_2), which is connected by adding the upper tree edge and e. If $k = 2$, this is the last block, which is handled later. Otherwise, there must be an edge f from u_1, beyond u_2 (else u_2 is a cut vertex). If f goes beyond u_3, form a block of length at least 4 with the vertices from u_1 to the other end of f, and add e, f and delete (u_1, u_2). If $f = (u_1, u_3)$, there must be an edge f' from u_2, beyond u_3. The vertices from u_1 to the other end of f' form a block, and we add e, f, f' and delete (u_1, u_2), (u_2, u_3).

4. e is incident on u_1: If $k = 1$, this is the last block, which is handled later. Otherwise this is possible only if N is a red path, and e goes to an ancestor. In this case, we use part of the free edges available for red paths. The block will be of length 1, and gets connected by installing the upper tree edge and an upper back edge.

Remark. The above cases have been stated in a highly oversimplified manner. The arguments about cut vertices above might not hold if some vertex is a contracted supernode. But in this case, we can charge the $D2$ edges inside this supernode, using Lemma 1. For each extra edge to pay we have at least 4 $D2$ edges available, and $\frac{a+k}{b+4k} \leq \frac{a}{b}$ if $\frac{a}{b} \geq \frac{1}{4}$.

Last Block. The description above assumes that the other end of e is not the last vertex; the blocks formed above are called intermediate blocks. 2-connecting the last block poses problems if its size is at most 3. In the case of red paths, we use the fact that two free edges are available for connecting the path. We use up $\frac{3}{2}$ of these while 2-connecting the intermediate blocks, and $\frac{1}{2}$ is available while dealing with the last block. For the last block of a blue path, we do not have any free edge available and the algorithm adds edges based on the last and the second last block together.

The Charging Strategy. As mentioned before, the edges of all the paths, and two extra (free) edges per each red path are part of the lower bound ($D2$). We add extra edges to 2-connect each path, and the cost of these edges is charged to the edges of the paths. For each block, the cost of adding an edge to connect it is charged to the edges in the block. This works only for blocks of length at least 4, since we want to beat the $\frac{1}{3}$ barrier. Therefore, we will partly charge child nodes of the current node as well, to reduce the charge on the current node. The way the child node is charged depends on whether it is red or blue. This is due to the difference in the structure of red and blue paths: a blue path is connected to its parent at one end, while a red path need not be. For a red path, we shall make sure that it can not only get 2-connected for charge less than $\frac{1}{3}$, but can also pay for some extra charge, which will be used by the parent node.

Claim. Any red path can be 2-connected within and upwards, and also pay for an extra $\frac{3}{14}$, while incurring a charge less than $\frac{37}{112} < \frac{1}{3}$. A red path, such that no back edge from a descendant has already been installed, can pay for an extra $\frac{5}{14}$.

The extra charge for the red path is achieved by making each block pay an extra $\frac{1}{7}$. Finally, we need to prove that the algorithm produces a feasible solution. The path nodes are connected top-down, and each time a path is processed, it is 2-connected to its parent. Thus, if the edges added by the ancestors are not altered, the solution would indeed be 2-connected. This is true, because in most cases (except possibly when considering a red path with 2 internal edges), the upper tree edge from each of the paths is retained. Even in the case of a red path with 2 internal edges, the upper tree edge is not retained only if no back edge from any descendant has been installed. Thus, an ear decomposition of the solution edges can be done, which is a proof of its feasibility.

4 The Algorithm for Planar Graphs

Let $G = (V, E)$ be a given planar graph with $|V| = n$. As before, assume that G is 2-VC and has no beta-structures, as in [14]. We show how to compute the minimum subgraph $CD2$ which is connected and in which each vertex has degree at least two. This is a stronger lower bound than $D2$. We actually compute a $(1+\epsilon)$ approximation to $CD2$, worsening our overall approximation by an extra $(1+\epsilon)$ multiplicative factor. By a simple reduction from the Hamilton cycle problem, the $CD2$ problem is NP-hard, even on cubic planar graphs. Modifying the approach of Grigni *et al.* [8], we can obtain:

Lemma 2. *There is a $(1+\varepsilon)$-approximation algorithm for the $CD2$ problem on planar graphs (for any $\varepsilon > 0$), with running time of $n^{O((\log^3 n)/\varepsilon^4)}$.*

If $n \geq 5$, one can show that any $CD2$ solution can be transformed in polynomial time to another $CD2$ solution of no greater size, and without any C_3 or C_4. Thus, the analysis in Section 3 gives a better ratio, since all blue paths have length at least 5 now. This gives us the following.

Theorem 2. *For any given $\varepsilon > 0$, a $(\frac{93}{70}+\varepsilon)$-approximate solution to the minimum size 2-EC problem on a planar graph can be found in time $n^{O((\log^3 n)/\varepsilon^4)}$.*

Remark. A more careful analysis yields a $(\frac{17}{13}+\varepsilon)$-approximation for planar graphs with the same running time.

5 The Algorithm for Cubic Graphs

In this section we give a local search based approximation algorithm for the 2-EC problem on cubic graphs. Let $G = (V, E)$ be a given 2-VC cubic graph, with $|V| = n$. Our algorithm has two steps. The first step involves computing an ear decomposition H of G with the minimum number ϕ of even ears (i.e. of even length), using the algorithm of Frank [4, 2] (delete all 1-ears, since they are redundant). It is easy to see, that $n + \phi - 1$ is a lower bound on the size of the optimum 2-EC solution in G [2].

Let a j-opt exchange that does not increase the number of even ears in H be called a *parity-preserving* j-opt exchange. In the 2nd step, run parity-preserving 1-opt heuristic on H w.r.t. 2-EC. Let H' be the resulting ear decomposition. Given an ear S in an ear decomposition $\mathcal{E}$, we say that an internal vertex v in S is *free* if $d_{\mathcal{E}}(v) = 2$.

Claim. H' is an open ear decomposition with all 2- and 3-ears being terminal. Any 5-ear in H' has at least two free vertices.

Let p_ℓ be the total number of internal vertices in the ℓ-ears of H'. Then, $p_\ell/(\ell-1)$ is the number of ℓ-ears. We can estimate the size of the solution as: $|E(H')| \leq \sum_{i=2}^{8} \frac{i}{i-1} p_i + \frac{9}{8}(n - \sum_{i=2}^{8} p_i)$. The first summation in the right denotes the number of edges in ℓ-ears for $\ell = 2, 3, \ldots, 8$. This gives: $|E(H')| \leq (\frac{9}{8}n + \frac{7}{8}p_2 + \frac{5}{24}p_4 + \frac{3}{40}p_6 + \frac{1}{56}p_8) + (\frac{3}{8}p_3 + \frac{1}{8}p_5 + \frac{1}{24}p_7)$.

Since we have used only parity preserving exchanges, the bound of ϕ still applies to the number of even ears: $n + p_2 + \frac{1}{3}p_4 + \frac{1}{5}p_6 + \frac{1}{7}p_8 - 1 \leq n + \phi - 1$, thus $\frac{9}{8}(n + p_2 + \frac{1}{3}p_4 + \frac{1}{5}p_6 + \frac{1}{7}p_8) \leq \frac{9}{8}(n + \phi - 1) + \frac{9}{8} \leq \frac{9}{8}opt + \frac{9}{8}$. So the first term in the brackets in $|E(H')|$ can be upper bounded by $\frac{9}{8}opt + \frac{9}{8}$.

By the claim above, each 3-ear (5-ear, 7-ear, resp.) has at least 4 (4, 2, resp.) vertices associated with it (2, (2, 0,resp.) free vertices, and the two end vertices). Therefore, we can lower bound the number of vertices as follows: $4\frac{p_3}{2} + 4\frac{p_5}{4} + 2\frac{p_7}{6} \leq n$, which gives us that $\frac{3}{8}p_3 + \frac{3}{16}p_5 + \frac{1}{16}p_7 \leq \frac{3}{16}n$. This we use to upper bound the second term in the brackets in $|E(H')|$. Finally, since $n \leq opt$, $|E(H')| \leq (\frac{21}{16} + \epsilon)opt$, for any $\epsilon > 0$.

Theorem 3. *There is a polynomial time local search* $(\frac{21}{16} + \epsilon)$*-approximation algorithm (for any* $\epsilon > 0$*) for 2-EC on cubic graphs.*

Application: Integrality gap of the LP. Let LP be the value of the standard linear program for 2-EC. Obviously, $n \leq LP$. [2] shows that $n + \phi - 1 \leq LP$. Thus, we obtain that the integrality gap of the LP is at most $\frac{21}{16} < \frac{4}{3}$ on unweighted cubic graphs.

6 Approximating $\{1, 2\}$-Connectivity

Due to the lack of space, we will make here a simplifying assumption that G is 2-vertex connected. Define function $\hat{r}$ as: $\hat{r}_u = \max\{r_{uv} : v \in V \setminus u\}$. We can assume that there exists a pair $u, v \in V$ with $r_{uv} = 2$, else the solution is a spanning tree.

6.1 Algorithm for $\{1, 2\}$-VC

We first present our modified lower bound for $\{1, 2\}$-VC. We assume that the input graph has no cut vertices.

Lemma 3. *If there is a pair* $u, v \in V$ *with* $r_{uv} = 2$*, then* $opt(G) \geq \max(n, 2|I|)$*, where* $I \subseteq V$ *is an independent set of vertices* v *with* $\hat{r}_v = 2$*.*

Our algorithm has two phases. The **first phase** is the same as in [5]. It computes a 2-VC spanning subgraph E': very roughly, it first finds a rooted DFS spanning tree, then adds extra back edges to 2-vertex connect the DFS tree, by processing it bottom-up. In the process, *blocks* are formed on addition of a back edge. The blocks form a partition of the vertices, finally. Our second phase is different from that in [5].

The algorithm – phase 2. After the 1st phase, set E' consisting of the edges of the DFS tree and of one back edge out of each non-root block has been output. $I = \emptyset$, initially. The 2nd phase will try for each block B to delete a tree edge within B, or if it is impossible to find a vertex s with $\hat{r}_s = 2$, add it to I, such that I remains independent. The 2nd phase will also modify the set of back edges.

Like in the second phase in [5], we traverse the blocks top-down. At each step we fix a block, say B, and B *decides* on the back edges going out of the child blocks of B. The first step is made with the root block: it chooses the farthest going back edge from each of its child blocks. Any child block having its back edge decided, decides on the back edges for its child blocks in a way we will specify. We proceed towards the leaves.

Let B be some non-root and non-leaf block for which the decision about the back edge $e = e(B)$ out of it has been made. Let v be an end vertex of e and $v \in B$, and $u = u(B)$ be the other end vertex of e. Let $p = p(B)$ be the parent vertex of B.

Block Property. We can assume that: (1) the back edge $e(B)$ goes higher than $p(B)$, and (2) there is an $u(B) - p(B)$ path that goes through the ancestor blocks of B.

The above property is obviously true for child blocks of the root block. We show that it is maintained during the algorithm, see Lemma 4.

Let w' be the highest vertex in path $[v, p)$ with $d_T(w') \geq 3$. If there is no such vertex in $[v, p)$, then let $w' = v$. Let w'' be the lowest vertex in $(w', p]$ which has a back edge from some child block of B. It can be shown that both w' and w'' are well defined. We note that the path $[w', w'']$ has length (i.e. the number of tree edges) at least one. Let q be the parent vertex of w'. The algorithm considers the following cases.

1. Assume that $q \neq w''$, and there is a vertex $s \in [q, w'')$ with $\hat{r}_s = 2$. Then, we label block B and vertex s with MARKED, and add s to the set I. B decides to retain the 1st phase choices of the farthest going back edges from all child blocks of B.
2. Assume that either $q \neq w''$, and for any vertex $s \in [q, w'')$, we have $\hat{r}_s = 1$, or $q = w''$. In this case the tree edge (q, w') is deleted from the current solution. We will show later that this step preserves the connectivities. Let e' be the back edge going from some child block B' of B into vertex w''. Then B decides not to take the farthest going back edge from B', but e' instead. For all other child blocks of B, B decides to retain the choice of the back edges made by the 1st phase.

In case 2 of the above algorithm when we delete a tree edge within block B, we pay this way for the back edge going out of B. Thus, in these cases the back edge is for free, and all such blocks are labeled FREE. Finally, the root block has no back edge going out of it and so is labeled FREE. Each leaf block is itself a vertex of requirement two, and so we label it MARKED, and choose all the leaf vertices into I.

Lemma 4. *(1) Each case of the 2nd phase of the algorithm maintains the Block Property, and preserves the feasibility of the solution w.r.t. r. (2) The set I of MARKED vertices is an independent set in G. Also all the vertices in I have the requirement of 2.*

Theorem 4. *The above algorithm is a linear time $\frac{3}{2}$-approximation algorithm for the unweighted $\{1, 2\}$-VC problem.*

6.2 Algorithms for $\{1, 2\}$-EC

Application of the previous algorithm. A straightforward application of the algorithm of Section 6.1 gives a $\frac{3}{2}$-approximation algorithm for $\{1, 2\}$-EC.
Simple algorithm. A simple modification of the algorithm of Khuller and Vishkin [11] leads to a $\frac{3}{2}$-approximation algorithm for $\{1, 2\}$-EC. Let $G = (V, E), r$ be a given instance of $\{1, 2\}$-EC. Find a DFS spanning tree of G and keep all the tree edges in our solution subgraph H. Whenever the DFS backs-up over a tree edge e, check if e is a cut-edge of current H (i.e. none of the back edges in H covers e). If yes, and if the cut $(S, \bar{S})$ given by e separates a vertex pair x, y with $r_{xy} = 2$, add the farthest going back edge covering e into H. Also, "mark" the cut $(S, \bar{S})$, where S is the vertex set of the subtree below e, and $\{x, y\}$ is separated by $(S, \bar{S})$ if S has exactly one of x, y.

The number of tree edges in H is at most $n-1$ which is at most $opt(G)$. The number of the back edges in H is equal to the number of "marked" cuts $(S, \bar{S})$. Because of the DFS tree property and the way back edges were chosen, any two such cuts are edge disjoint. Thus, the optimal solution to $\{1, 2\}$-EC must have at least 2 edges in each of these cuts. So, the number of these cuts is at most $\frac{1}{2}opt(G)$, and $E(H) \leq \frac{3}{2}opt(G)$.
Local optimization heuristics. Generalizing the local optimization techniques used in Section 5, we can obtain the following results.

Theorem 5. *There are local search based algorithms which achieve a $\frac{7}{4}$-approximation for the $\{1,2\}$-VC problem and a $\frac{5}{3}$-approximation for the $\{1,2\}$-EC problem.*

Remark. Theorem 5 generalizes a result for the 2-VC communicated by J. Cheriyan.

7 Conclusions

While our improvement for 2-EC is small, it shows that $\frac{4}{3}$ can certainly be improved. With a more careful analysis, it should be possible to improve the approximation factor further. We think that similar methods can also improve the approximation ratio for 2-VC. We have also presented approximation algorithms for $\{1,2\}$-VC and $\{1,2\}$-EC. The algorithms are based on depth-first-search methods and on local search heuristics. It is important to note that almost all our algorithms use local search as the main method or as a subroutine. We think it would be interesting to further develop applications of the local search paradigm in the area of approximating connectivity problems.

References

1. R. Carr and R. Ravi. A new bound for the 2-edge connected subgraph problem. In the *Proc. 6th IPCO,* LNCS **1412**, 1998.
2. J. Cheriyan, A. Sebő and Z. Szigeti. An Improved Approximation Algorithm for Minimum Size 2-Edge Connected Spanning Subgraphs. In the *Proc. 6th IPCO*, LNCS **1412**, 1998.
3. C.G. Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *Journal of Algorithms*, **28**, pp. 105–124, 1998.
4. A. Frank. Conservative weightings and ear-decompositions of graphs. *Combinatorica*, **13**, pp. 65–81, 1993.
5. N. Garg, V. Santosh and A. Singla. Improved Approximation Algorithms for Biconnected Subgraphs via Better Lower Bounding Techniques. In the *Proc. 4th ACM-SIAM SODA*, pp. 103–111, 1993.
6. M.X. Goemans, A. Goldberg, S. Plotkin, D.B. Shmoys, È. Tardos and D.P. Williamson. Improved Approximation Algorithms for Network Design Problems. In the *Proc. 5th ACM-SIAM SODA*, pp. 223–232, 1994.
7. R.L. Graham, M. Grötschel and L. Lovász, editors. *Handbook of Combinatorics. Volume I.* North-Holland, 1995.
8. M. Grigni, E. Koutsoupias and C.H. Papadimitriou. An approximation scheme for planar graph TSP. In the *Proc. of the IEEE FOCS*, 1995.
9. M. Grötschel, C. Monma and M. Stoer. Design of survivable networks. In *Handbook in Operations Research and Management Science,* Volume on Networks. North-Holland, 1995.
10. K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. In the *Proc. of the IEEE FOCS*, 1998.
11. S. Khuller and U. Vishkin. Biconnectivity Approximations and Graph Carvings. *Journal of the ACM*, **41**(2), pp. 214–235, 1994.
12. H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, **7**, pp. 583–596, 1992.
13. R. Ravi and D.P. Williamson. An Approximation Algorithm for Minimum-Cost Vertex-Connectivity Problems. *Algorithmica*, **18**(1), pp. 21–43, 1997.
14. S. Vempala and A. Vetta. Factor 4/3 Approximations for Minimum 2-Connected Subgraphs. In the *Proc. 3rd International Workshop APPROX,* LNCS **1913**, 2000.