# A 5/4-approximation algorithm for minimum 2-edge-connectivity[*]

Raja Jothi Balaji Raghavachari Subramanian Varadarajan[†]

**Abstract**

A 5/4-approximation algorithm is presented for the minimum cardinality 2-edge-connected spanning subgraph problem in undirected graphs. This improves the previous best approximation ratio of 4/3. It is shown that our ratio is tight with respect to current lower bounds, and any further improvement is possible only if new lower bounds are discovered.

## 1 Introduction

A network design problem which requires the underlying network to be resilient to link failures is known as the *Edge Connectivity Survivable Network Design Problem* (EC-SNDP). 2-edge-connectivity (2EC) is a major feature in today's fast and reliable communication networks as single transmission failure could cause intolerable losses. 2-edge connectivity between any two terminals in a network is needed to guarantee network connectivity in case of a main transmission link failure. A graph is said to be *2-edge-connected* if the deletion of any single edge does not disconnect it. In other words, a link failure continues to allow communication between functioning sites.

Most network optimization problems that require finding minimal subgraphs satisfying given connectivity constraints are NP-hard. Hence, it has become imperative to design approximation algorithms for such problems. The problem of finding a minimum cardinality 2-edge-connected spanning subgraph of a given undirected graph is known to be NP-hard and MAXSNP-hard [4, 6]. In this paper, we present a 5/4-approximation algorithm for the 2-edge-connected spanning subgraph problem in undirected graphs.

**1.1 Problem Statement.** The input is an arbitrary, undirected, 2-edge-connected graph $G = (V, E)$. On such a graph, the 2-edge-connectivity problem (2-ECSS) is to find a minimum cardinality subset of edges $H \subseteq E$ such that the graph $G = (V, H)$ is 2-edge-connected. The problem is known to be NP-hard [10, 14]. In this paper we present an approximation algorithm for the problem that achieves a ratio of 5/4. Throughout this paper, "2-connectivity" refers to 2-edge-connectivity.

**1.2 Previous results.** In their ground-breaking paper, Khuller and Vishkin [16] demonstrated a 3/2-approximation algorithm for 2-ECSS. They also presented an algorithm for the analogous vertex connectivity problem, 2-VCSS, whose approximation ratio is 5/3. Their algorithms used a depth-first search (DFS) tree of the graph as a starting point and added carefully chosen back edges that cross bridges and cut vertices. Developing on this work further, Garg, Santosh and Singla [11] gave an improved algorithm for 2-VCSS whose performance ratio is 3/2. Cheriyan, Sebő and Szigeti [1] improved the approximation ratio to 17/12 for 2-ECSS using an ear decomposition of the graph in which the number of odd-length ears is maximal.

Vempala and Vetta [19] designed new algorithms for both 2-ECSS and 2-VCSS with 4/3 approximation ratio. The following key concepts introduced by them have paved the way to design better algorithms. They introduced the novel concept of a *beta-structure* to avoid hard configurations. They also preprocessed the graphs to remove cut vertices and adjacent degree two nodes. In addition, they used a solution to the "D2 problem" (minimal subgraph in which each vertex is incident to at least two edges) as the starting point instead of a DFS tree. Recently, Krysta and Kumar [17] improved the ratio for 2-ECSS to about 1.3326 by employing a smart charging scheme.

**1.3 Our contributions.** We describe an improved approximation algorithm for 2-ECSS with an approximation ratio of 5/4. There are examples to show that the analysis of our algorithm is tight. In addition, one of the examples in [11] shows that improving beyond our algorithm requires new lower bounding techniques. We combine key ideas from Vempala and Vetta [19] and Krysta and Kumar [17], and introduce new ideas to obtain the 5/4 bound.

[*]Research supported by the National Science Foundation under grant CCR-9820902.

[†]Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75080. E-mail: {raja, rbk, sxv010010}@utdallas.edu.

**1.4 Other related work.** Hartvigsen [12, 13] has presented polynomial-time algorithms for finding 3-cycle-free maximal 2-matchings in general graphs, and 4-cycle (square) free maximal 2-matchings in bipartite graphs. The problem of finding a minimum-weight $k$-edge-connected spanning subgraph ($k$-ECSS) is known to be NP-hard [10]. Khuller and Vishkin [16] obtained a 2-approximation algorithm for weighted $k$-ECSS for $k > 1$.

Nagamochi and Ibaraki [18] gave an efficient algorithm for finding a sparse $k$-connected subgraph of a given graph. Khuller and Raghavachari [15] demonstrated an algorithm for $k$-ECSS with approximation ratio of 1.85 and an algorithm for $k$-VCSS with a performance ratio of $2 + \frac{1}{n}$ in graphs satisfying the triangle inequality. Fernandes [6] improved the approximation ratio to 1.75 for $k$-ECSS. Cheriyan and Thurimella [3] presented an elegant algorithm (which we call the CT algorithm) that achieves a performance ratio of $1 + \frac{2}{k+1}$ for unweighted $k$-ECSS. They also presented an algorithm for the unweighted k-vertex-connectivity problem ($k$-VCSS) with $1 + \frac{1}{k}$ as the approximation ratio. Cheriyan, Vempala and Vetta [2] presented an algorithm for weighted $k$-VCSS with approximation ratio $O(\log k)$ (as long as $|V| \geq 6k^2$).

Gabow [8] has given a 3/2 approximation algorithm for 3-ECSS. This matches the ratio obtained by the CT algorithm, but Gabow's algorithm works on multigraphs as well, whereas the CT algorithm guarantees a ratio of 3/2 for 3-ECSS only on simple graphs. In fact Gabow [9] shows examples of multigraphs for which the ratio obtained by the CT algorithm is arbitrarily close to 2. In that paper he also shows how to modify an earlier algorithm of [15] for $k$-ECSS to multigraphs with a performance ratio of about 1.61.

## 2 Definitions

Let $G = (V, E)$ be the given graph with $|V| = n$. Let *Opt* be an optimal 2-ECSS of $G$. We will also use *Opt* to denote the cardinality of an optimal 2-ECSS of $G$, and this should cause no confusion. A node $v$ is called a *beta vertex* if the removal of some two nodes $x$ and $y$ from $G$ (along with their incident edges) results in at least three connected components with one of them containing just $v$. This definition extends easily to the concept of a *beta pair.* A subgraph $H = (V, E')$ with $E' \subseteq E$ is called *D2* if the degree of each node is at least 2. A *2-matching* is a subgraph, all of whose nodes have degree 2 or less. It is easily shown that the minimal D2 and the maximal 2-matching problems are closely related, and a solution for one can be easily converted into a solution for the other. Both problems are solved using algorithms for matching.

Vempala and Vetta [19] showed that there is no loss of generality in assuming that $G$ has no cut vertices, beta vertices or pairs, and adjacent degree-2 nodes. We will use their preprocessing steps and apply our algorithm only on such preprocessed graphs. In addition, as suggested by them, we will start with a minimal 3-cycle-free D2, or equivalently, a maximal 3-cycle-free 2-matching. It is easy to convert any minimal D2 into a maximal 2-matching by deleting all but two edges at every node whose degree is more than 2, and a similar transformation in the other direction is also possible.

## 3 Overview of the algorithm

We will assume that the graph $G$ has already been preprocessed to ensure that it has no cut-vertices, beta nodes and pairs, and adjacent degree-2 nodes (see [19] for more details). We then use Hartvigsen's algorithm [12] to find, in polynomial time, a maximal 3-cycle-free 2-matching (maximal subgraph without 3-cycles, in which each vertex is incident to at most two edges). Let the 2-matching that we obtain be a collection of cycles $C$ and a set of paths $P$.

PROPOSITION 3.1. *Let $G = (V, E)$ be a graph defined on $n \geq 4$ vertices, with no cut vertices. There exists an optimal 2-ECSS of $G$ that has no 3-cycles.*

*Proof.* The following procedure can be used to remove any 3-cycles from an optimal 2-ECSS (*Opt*) without increasing its cardinality. Suppose it has a 3-cycle $xyzx$. Since all edges of *Opt* must be critical, the removal of the 3 edges of this 3-cycle must break up *Opt* into three connected components, with exactly one of $\{x, y, z\}$ in each component. Otherwise, if for example, $x$ and $y$ are in the same component, we can find 3 paths between $x$ and $y$, and therefore the edge $(x, y)$ is redundant, contradicting the minimality of *Opt*. Since the graph has no cut vertices, there must be an edge $e$ in $G$ that connects the component containing $x$ to one of the other components, say $y$'s component. In this case, adding $e$ and removing $(x, y)$ gives an optimal 2-ECSS of $G$ that has one fewer 3-cycle than *Opt*. ∎

PROPOSITION 3.2. *Let $G = (V, E)$ be a graph defined on $n \geq 4$ vertices, with no cut vertices. Let $H$ be a maximal 3-cycle-free 2-matching of $G$, consisting of a collection of cycles $C$ and a set of paths $P$. Let Opt be the cardinality of an optimal 2-ECSS of $G$. Then Opt $\geq n + |P|$.*

*Proof.* A simple counting argument shows that $|H| = n - |P|$. Since $H$ is maximal and $n$ is fixed, $P$ is minimal for a 3-cycle-free 2-matching.

Proposition 3.1 showed that $G$ has an optimal 2-ECSS that does not have any 3-cycles. Consider an ear decomposition of such an optimal solution. Let $e$ be the number of its ears. Therefore $Opt = n + e - 1$. Since it is a minimal solution, it has no trivial ears (with just one edge). Deleting the first and last edge of each ear gives us a 2-matching with 1 cycle and $e - 1$ paths. Since we started with a 3-cycle-free optimal solution, we get a 3-cycle-free 2-matching. As observed earlier, the number of paths is minimal in $H$ and therefore, $|P| \leq e - 1$. Therefore, $Opt = n + e - 1 \geq n + |P|$. ∎

**3.1 Charging strategy.** Since our goal is to get an algorithm whose ratio is 5/4, we can use up to $\frac{5}{4}(n + |P|)$ edges. We split this into two parts. Each vertex is given a charge of 5/4, called its *vertex charge*. Each path in $P$ is also given a charge of 5/4, called its *path charge*. As the algorithm builds a solution, it has to pay for the edges through a combination of vertex and path charges. Our charging strategy has been adopted from the work of Krysta and Kumar [17].

With $C$ and $P$ on hand, we build a DFS tree of $G$ using the following strategy. When the first node $x_1$ of a cycle $X \in C$, where $X = \{x_1, x_2, \ldots, x_k\}$, is discovered, we will make the DFS go through the nodes of the cycle in sequence until it reaches the end of the cycle. Upon reaching a leaf node, we take the subgraph induced by the last 7 nodes and see if there is a different way of traversing them such that the new leaf vertex is adjacent to an unvisited vertex. If such a reorganization is possible, we do so, and the DFS is able to continue further. Later we will show that we need to look for specific configurations, and hence this search for a path in the 7-node induced subgraph can be done efficiently.

When a path $Y \in P$ is encountered by DFS, it is not necessary that the initial node encountered be an end vertex of the path. We use the following idea from [17] to handle paths, with a minor modification. If DFS enters $Y$ at one of its end vertices, then it goes through the vertices of the path in that sequence. That segment of the DFS tree inherits the 5/4 path charge of $Y$. On the other hand, if a node in the middle of $Y$ is encountered, we have the option of going in either direction to one of the ends of the path. The DFS chooses to search the segment of $Y$ that is longer, and allocate the 5/4 path charge to the smaller, unexplored segment, which is called the *residual* segment of $Y$. Inductively, there is one residual segment $Y' \subseteq Y$ that may still be unexplored with a path charge of 5/4. One exception to this method is when the DFS reaches the middle of a residual path of 8 nodes, and the longer segment has 5 nodes. In this case alone, we go the other way, expanding the DFS tree by 4 nodes and leaving a residual segment of 4 nodes. Whenever we consume a path segment of 5 nodes in the DFS tree, we assign that segment a path charge of 1/4.

When a residual segment gets small, we partition the available path charge to both segments to ensure the following. Each segment of three nodes or five nodes is allocated a path charge of 1/4, two nodes a charge of 1/2 and a singleton node a charge of 3/4. These path charges will be used to account for edges when a branch of the DFS tree has fewer than 4 nodes. The reason behind allocating a charge of 1/4 to path segments of 5 nodes will be discussed later. Since the 2-matching solution that we used has no 3 cycles, branches with fewer than 4 nodes can only be path segments.

Proposition 3.3. *The scheme described above distributes path charges as follows: 5-node and 3-node segments of a path receive at least 1/4 each, 2-node segments receive at least 1/2 each, and 1-node segments receive at least 3/4 each.*

We break the DFS tree into a sequence of paths in a natural way. The first path starts at the root and goes down to the first leaf encountered. Every time the DFS finds an unvisited vertex and starts a new search, we start building a new path that extends up to the first leaf node encountered by it. Let these DFS paths be $D_1, D_2, \ldots, D_p$. We will process the paths one at a time and 2-connect each path by adding carefully chosen back edges. Selected edges of the paths will be dropped in some cases. We will show that we can 2-connect each path using the vertex charge of 5/4 received by each vertex plus any path charge it may have been allocated.

**3.2 Additional operations.** In general, our strategy is to try to construct large cycles starting at a leaf. Some configurations are difficult to handle and we introduce additional operations that will enable our algorithm to avoid these difficult cases.

**Cycle shrinking:** During the course of the algorithm, we may find a cycle $C$ of 5 or more edges for which we are able to prove that $Opt$ uses at least $|C| - 1$ edges within the subgraph induced by $C$. In such cases, we shrink the cycle $C$ into a single node and solve the resulting problem recursively, and then add $C$ to its output. It is easy to show that this scheme works as long as $|C| \geq 5$ [19]. We call this operation as SHRINK($C$). We extend it to shrinking a larger 2-connected subgraph of $G$ as follows. Let $S$ be a set of 11 nodes that can be 2-connected using 12 edges (using 2 ears). If we can prove that any 2-ECSS of $G$ contains at least 10 edges within the subgraph induced by $S$, then we can apply the above shrinking strategy to $S$ and recursively solve the problem and still manage to get a 5/4 approximation ratio.

LEMMA 3.1. *Let $G = (V, E)$ be a 2-edge-connected graph that has neither cut vertices nor beta vertices and pairs. Suppose there is a subset $C \subset V$ of nodes for which it is possible to prove that any 2-ECSS of $G$ contains at least $|C| - 1$ edges within the induced subgraph of $C$, and $|C| \geq 5$. Suppose there exists a subgraph that 2-connects the nodes of $C$ using at most $|C| + \left\lfloor \frac{|C|-5}{4} \right\rfloor$ edges. Then we can shrink $C$ and solve the 2-ECSS problem and still achieve an approximation ratio of 5/4.*

*Proof.* Suppose we take an optimal 2-ECSS, *Opt*, and shrink $C$ into a single node, then we get a 2-connected graph that has $|V| - |C| + 1$ nodes and at most $Opt - |C| + 1$ edges. Suppose we find a 2-ECSS of the shrunken graph using a 5/4 approximation algorithm. The solution returned has at most $\frac{5}{4}(Opt - |C| + 1)$ edges. We now expand the nodes of $C$ and add the subgraph that 2-connects $C$ to it. We get a 2-connected subgraph of $G$ that has at most

$$\frac{5}{4}\Big(Opt - |C| + 1\Big) + |C| + \left\lfloor \frac{|C| - 5}{4} \right\rfloor$$

edges, which is at most $\frac{5}{4}Opt$. ∎

**Donation:** In certain cases, when we are processing a path $D$, if one of its nodes, $v$, has a child $c$ in the DFS tree outside $D$, i.e., $c$ would be the root of a path $D_k$ for some $k > i$, (see Figure 1(a)), we may 2-connect all nodes of $D$ except $v$. The path $D_k$ will be extended to include $v$ (see Figure 1(b)), and in this case, we say that $D$ donates $v$ to $D_k$. If it so happens that $D_k$ is a path of 4 nodes, then we would have to provide an extra charge of 1/4 to $D_k$, since we assumed that all 5-node path segments have a path charge of 1/4 available, unless it came from a 5-cycle. We will show that when $D$ donates a single vertex $v$, it has excess charge available, and therefore gives to $D_k$ a charge of 1/4 from its excess.

**Plunder:** Consider an edge $(u, v) \in C$, where $C$ is a cycle that we are currently constructing from the leaf node of a path $D$. Suppose $u$ has an unprocessed descendant $d$ that is part of a path $D_k \neq D$, such

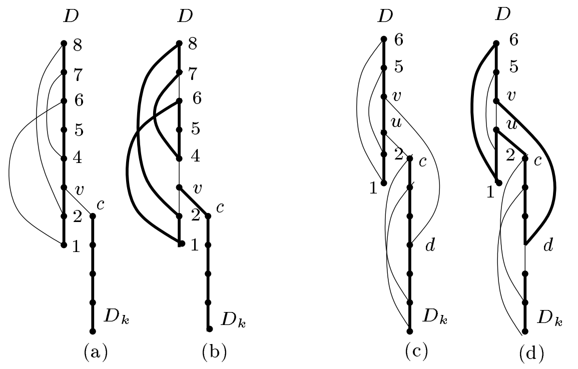

Figure 1: Donation and plundering

that there is some back edge $(d, v)$ (see Figure 1(c)). In this case, we can replace the edge $(u, v)$ by a path from $u$ to $v$ through $d$ (see Figure 1(d)). We refer to this operation as a *plunder*. During a plunder, we account for the vertex charges in the following way. We can show that if $D_k$ either has at most 4 nodes or is a 5-node cycle, then all its vertices are plundered. Otherwise, we guarantee that at least 4 nodes are plundered from $D_k$. If $p$ nodes are plundered from $D_k$ by $u$, they have a vertex charge of $5p/4$ available, out of which $u$ keeps $p + 1/2$ and returns the rest to $D_k$ as path charge allocated to $D_k$ by the plunderer. This allows enough charge to 2-connect the left over nodes in $D_k$. Note that in case $D_k$ originally had 5 nodes, and only one node is left after the plunder, then it still has a vertex charge of 5/4, an already available path charge of 1/4, and a path charge of 3/4 left by the plunderer, which together accounts for 2 edges that will be needed to 2-connect this single node to the core. Also, $D$ has gained a charge of 1/2, which is the equivalent of having two extra nodes in it. For nodes of any other $D_x$ in the path from $u$ to $D_k$, we ensure that the effective path length of $D_x$ stays the same. For each node that $u$ takes away from $D_x$, it takes away only 1 out of the 5/4 vertex charge available and it leaves behind a path charge of 1/4 to $D_x$.

Due to this accounting method, after plundering, the *effective* length of $C$ is equal to the number of nodes in it from $D$ plus two corresponding to the 1/2 it gained from $D_k$. In addition, paths which lost nodes due to a plunder have extra quarters to compensate their loss, and therefore, when we 2-connect them, we can act as if they still have their original length.

## 3.3 More details.

We first start by 2-connecting $D_1$, the first path in the DFS tree. We will try to build an ear decomposition of the nodes of $D_1$, starting from the leaf. Once the ear decomposition reaches the root vertex, all but the donated vertices of $D_1$ are two-connected, which we will call as the "core". We then proceed iteratively as follows. Out of the remaining paths, we select the path $D$ whose head is visited earliest by DFS and 2-connect their nodes to the core using the same ideas. Note that the paths may be processed in a order different than $D_2, \ldots, D_k$ since donations and plunders affect the order in which the paths are considered. Since there are no cross edges with respect to a DFS tree, we are able to 2-connect them one path at a time. At the end of the sequence, we get a solution that is 2-connected and, in fact, we compute an ear decomposition of the solution in which most ears contain at least 4 new nodes. There may be a few smaller ears generated by residual segments with three or fewer nodes, and we account for their edges by combining vertex charges and path charges. Since the paths $D$ are generated by DFS, non-tree edges of $G$ are all back edges.

We will show the following through a sequence of lemmas:

- Lemma 4.1 shows how to find a 5-cycle starting at the leaf of a given path $D$.
- Lemma 4.3 shows how to extend it to a 6-cycle.
- Lemma 4.5 shows how to extend it to a 7-cycle, donating nodes in some cases.

In all cases, if the path does not have enough nodes, then we will find a single cycle that includes all nodes. If certain configurations arise, then we will find a cycle $C$ in which *Opt* must use at least $|C| - 1$ edges within $C$, or a 11 node set which can be 2-connected using 12 edges within which *Opt* must use at least 10 edges. In these situations, we shrink the nodes into a single vertex and solve the problem recursively.

We now describe how our algorithm 2-connects a DFS path $D$ to the core. The behavior of our algorithm depends on the effective length of the path, which is the original number of vertices in it plus any vertices that may have been donated to it. Recall that when a path gets plundered and loses vertices, it gets to keep a 1/4 for each vertex that it loses, effectively maintaining the same length.

$|\mathbf{D}| \leq \mathbf{3}$: Paths with less than 4 vertices are either path segments or paths that lost vertices in a plunder, since we don't have 3-cycles in our initial 2-matching that guides the DFS. As mentioned in our charging strategy, when $|D| \leq 3$, then $D$ has a path charge of $(4 - |D|)/D$. We are able to 2-connect $D$ using a single ear which can be paid using vertex and path charges.

$|\mathbf{D}| = \mathbf{4}$ **or 5**: A single ear that extends from the core through all the vertices of $D$ is constructed. The edges are paid using the vertex charges of $D$.

$|\mathbf{D}| = \mathbf{6}$ **or 7**: A single ear that extends from the core through all the vertices of $D$ is constructed. One of the vertices may be donated in this step. The edges are paid using the vertex charges of the vertices of $D$ in the single ear. If such an ear cannot be constructed, we show that we can call SHRINK($D$). The proof for cases $|D| = 6$ or 7 is similar to the proof of Lemma 4.5 and is not included due to lack of space.

$|\mathbf{D}| = \mathbf{8}$: First we find a cycle of length at least 5 starting at the leaf node. We then shrink this cycle into a single vertex, and find a single ear (containing at most 5 edges) starting at the core that includes all the nodes. Together at most 2 ears are used, and the edges can be paid using vertex charges.

$|\mathbf{D}| = \mathbf{9}$ **or 10**: Find a cycle of length at least 6 starting at the leaf. Shrink the cycle and connect the remaining nodes to the core using a single ear.

$|\mathbf{D}| = \mathbf{11}$: Find a cycle of length at least 6 starting at the leaf. Try to extend the cycle using methods of Lemma 4.5. If successful, shrink cycle and 2-connect the nodes using another ear. In some of the cases, we will donate up to 3 nodes to other paths, thus shrinking the length of $D$. We can then 2-connect the remaining nodes using one of the earlier cases. Otherwise we will show that any 2-ECSS has at least 10 edges in the subgraph induced by $D$ and therefore call SHRINK($D$).

$|\mathbf{D}| = \mathbf{12}$: It is easy to extend the method used for $|D| = 8$ to any multiple of 4 by 2-connecting $D$ using $|D|/4$ ears as follows. Find a 5-cycle from the leaf, shrink and recurse.

$|\mathbf{D}| > \mathbf{12}$: We find a cycle with 6 or more edges from the leaf, shrink it into a single node and repeat the process two more times. If there are any nodes left in $D$, find a cycle of length 5 or more, until all nodes of $D$ (except donated nodes) are 2-connected to the core.

## 4 Growing a cycle from the leaf

We now state and prove several interesting and useful claims. In the following, when we say a back edge *crosses* a node $v$, we mean that there is a back edge in $G$ that connects a proper descendant of $v$ to a proper ancestor of $v$. A node $x$ crosses (an ancestor) $v$ if either there is a back edge $(x, y)$ crossing $v$ or if there is a back edge $(d, y)$ that crosses $v$, where $d$ is a descendant of $x$ through a child $c$ not in the current path $D$. In such cases, if $(x, y) \notin E$, then we will add a *virtual* edge $(x, y)$ when we seek to grow a cycle.

Once we have identified the set of ears with which we 2-connect the nodes of $v_i$ to the core, we replace all the virtual edges by plundering the corresponding tree path from $x$ to $d$ and the back edge $(d, y)$. Since $d$ is a descendant of $x$ through a child $c$ that is not in $D$, the paths added to replace the virtual edges are all disjoint from each other. Therefore, in the following discussion, we will treat virtual edges the same as the real edges of $G$. Hence we will assume that if $x$ crosses $v$, then there is a back edge $(x,y)$ that crosses $v$. In addition, we will treat the core as an extra vertex (labeled as $v_q$ below) that is added to the end of the path, but the core does not have any charges available to pay for any edges.

## 4.1 Finding a 5-cycle

LEMMA 4.1. *Let $D$ be a DFS path whose vertices are labeled $v_1, v_2, ..., v_q$ starting from the leaf vertex. It is possible to find a cycle that includes all vertices in $\{v_1, v_2, \dots, v_k\}$ for some $k \geq \min(q, 5)$. The cycle may include plundered segments from the heads of unprocessed paths.*

*Proof.* We show that a cycle of five or more vertices can be formed based on one of the following cases. Observe that $v_1$ is a leaf vertex in the DFS tree and therefore has no children. The only edges from $v_1$ other than the tree edge $(v_1, v_2)$ are back edges.

*Case 1: The farthest back edge from vertex $v_1$ is $(v_1, v_k)$, $4 < k \leq q$.* Form the cycle $v_1 \dots v_k v_1$.

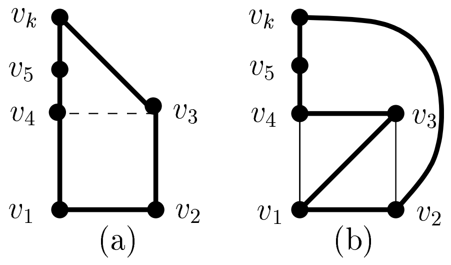

Figure 2: Lemma 4.1: Case 2

*Case 2: The farthest back edge from vertex $v_1$ is $(v_1, v_4)$.* In order that vertex $v_4$ is not a cut vertex, either $v_2$ or $v_3$ (or both) must cross $v_4$. If $v_3$ crosses $v_4$ with edge $(v_3, v_k)$, then form the cycle $v_1 v_2 v_3 v_k \dots v_4 v_1$ (see Figure 2 (a)). Otherwise if only $v_2$ crosses $v_4$, then the edge $(v_1, v_3)$ must exist, since without it, the removal of $v_2$ and $v_4$ generates three components — $\{v_5, \dots, v_q\}$, $\{v_3\}$, and $\{v_1\}$, which makes $v_1$ and $v_3$ beta vertices. Observe that $v_3$ cannot have a child $c$ other than $v_2$, since in that case we would then have continued the DFS as $v_4, v_1, v_2, v_3, c, \dots$ instead of accepting $v_1$ as a leaf vertex. Hence, we can form the cycle $v_1 v_2 v_k \dots v_4 v_3 v_1$ (see Figure 2 (b)).

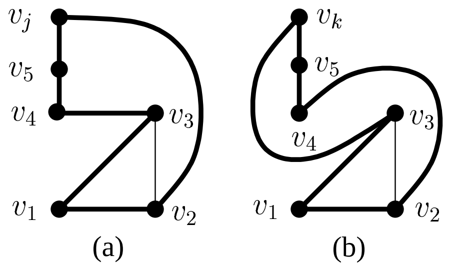

Figure 3: Lemma 4.1: Case 3

*Case 3: The farthest back edge from vertex $v_1$ is $(v_1, v_3)$.* As in the previous case, $v_2$ must cross $v_3$, since otherwise $v_3$ would be a cut vertex. If the farthest back edge from $v_2$ lands on $v_j$ with $j > 4$, then form the cycle $v_1 v_2 v_j \dots v_3 v_1$ (see Figure 3 (a)). Otherwise, if the farthest back edge from $v_2$ only reaches $v_4$, then $v_4$ threatens to be a cut vertex. The only possibility is that $v_3$ must cross $v_4$ with a back edge that lands on $v_k$, $k > 4$. In this case, we form the cycle $v_1 v_2 v_4 \dots v_k v_3 v_1$ (see Figure 3 (b)). ∎

The above lemma can be extended to the case when vertex $v_1$ is not a single node of $G$, but a 2-connected component that has been shrunk into a single node. Since the algorithm works by repeatedly finding a cycle starting from the leaf and shrinking it into a single node, it is necessary to prove an extended version of Lemma 4.1. The proof is more complicated we can no longer use the fact that certain nodes cannot have neighbors outside the current path because the algorithm reorients the last few nodes of a path if that allows it to be extended further. We are able to prove the lemma by introducing donations and plunders in carefully selected places. There are many different cases to consider, and we omit the proof here due to lack of space.

LEMMA 4.2. *Let $D$ be a DFS path labeled as in the previous lemma, $v_1, \dots, v_q$. The leaf node $v_1$ represents a 2-connected component formed by a set of ears starting at the original leaf of $D$, and $v_q$ is the core. We can find a cycle that includes all vertices in $\{v_1, v_2, \dots, v_k\}$ for some $k \geq \min(q, 5)$. At most one vertex is donated, and the cycle may include plundered segments from the heads of paths considered after $D$.*

**4.2 Finding a 6-cycle.** We now show that the operations in Lemma 4.1 can be continued further to find a cycle of length 6 or more.

LEMMA 4.3. *Let $D$ be a DFS path containing at least 7 nodes, labeled as in the previous lemma. It is either possible to find a 5-cycle $C$ on which we can call* SHRINK*($C$) or to find a cycle that includes all vertices in $\{v_1, v_2, \ldots, v_k\}$ for some $k \geq 6$. As before, the cycle may include plundered segments from the heads of paths considered after $D$.*

*Proof.* By Lemma 4.1, we can find a cycle through vertices $v_1$ to $v_5$. If this cycle also includes vertex $v_6$, we are done. Otherwise, we show how to extend the cycle further based on one of the following cases. We consider a case only if all the previous cases have failed.

To prevent $v_5$ from being a cut vertex, at least one of the vertices $\{v_1, v_2, v_3, v_4\}$ must cross $v_5$. In all cases below, let the back edge that crosses $v_5$ land on $v_k$, $k > 5$.

*Case 1: $v_1$ and/or $v_4$ can cross $v_5$.* In this case, we expand the cycle by taking either $v_1 \ldots v_k v_1$ or $v_1 v_2 v_3 v_4 v_k \ldots v_5 v_1$.

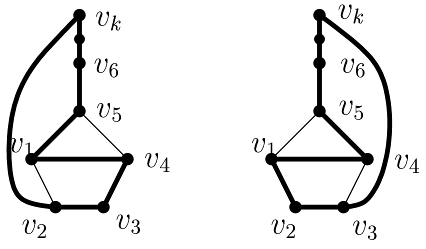

Figure 4: Lemma 4.3: Case 2

*Case 2: edge $(v_1, v_4)$ exists.* Either $v_2$ or $v_3$ crosses $v_5$. In this case, we expand cycle by taking either $v_1 v_4 v_3 v_2 v_k \ldots v_5 v_1$ or $v_1 v_2 v_3 v_k \ldots v_5 v_4 v_1$ (see Figure 4).

*Case 3: $(v_1, v_4) \notin E$.* Neither $v_1$ nor $v_4$ can have any children outside the current path $D$, since otherwise, the path would have been extended to include that vertex by reorienting the path, making $v_1$ or $v_4$ as the last vertex of the path, making it further extensible by DFS. Therefore, since $(v_1, v_4)$ is not an edge, any 2-connected solution must have at least 4 edges incident to $\{v_1, v_4\}$ and all these edges are only within $C = \{v_1, v_2, v_3, v_4, v_5\}$. In other words, there are at least 4 edges within $C$ in any optimal solution. Therefore we can shrink $C$ into a single node and recurse by calling SHRINK($C$) (with 5/4 as the target ratio). ∎

We now state the extended version of the above lemma. It is similar to Lemma 4.2. In some situations, we shrink a set of vertices in which we are able to prove that any 2-ECSS must include at least $|C| - 1$ edges. Up to three edges are donated in the process of finding a cycle.

LEMMA 4.4. *Let $D$ be a DFS path labeled as in the previous lemma, $v_1, \ldots, v_q$. The leaf node $v_1$ represents a 2-connected component formed by a set of ears starting at the original leaf of $D$, and $v_q$ is the core. It is either possible to find a set of nodes $C$ on which we can call* SHRINK*($C$) or to find a cycle that includes all vertices in $\{v_1, v_2, \ldots, v_k\}$ for some $k \geq \min(q, 6)$. At most three vertices are donated, and the cycle may include plundered segments from the heads of paths considered after $D$.*

## 4.3 Finding a 7-cycle.

We now consider the case when $|D| = 11$. Combining the vertex charges of all these nodes gives us a total of 13 3/4. Therefore in order to get a ratio of 5/4, we need to try to 2-connect $D$ to the core (which can be viewed as a 12th vertex) using at most 13 edges, i.e., using at most 2 ears. We will show how to process $D$ such that we will either 2-connect $D$ to the core using at most 2 ears, or find a way to donate at least one node, and up to three nodes, to unprocessed paths so that the remaining nodes of $D$ can be 2-connected to the core easily using up to two ears, or show that we can shrink $D$ by proving that any 2-ECSS must contain at least 10 edges in the induced subgraph of $D$.

LEMMA 4.5. *Let $D$ be a DFS path containing 11 nodes, labeled $v_1, \ldots, v_{11}, v_{12}$, where $v_{12}$ is the core. It is possible to do at least one of the following:*

- *2-connect $D$ to the core using 3 ears if it has a path charge of 1/4 or more available.*
- *2-connect $D$ to the core using at most 2 ears, donating at most 3 nodes to other paths that are still unprocessed by the algorithm.*
- *Find a set of nodes $C$ such that $C$ satisfies the conditions stated in Lemma 3.1.*

*As before, the cycle may include plundered segments from the heads of paths considered after $D$.*

*Proof.* By Lemma 4.3, we can find a cycle through vertices $v_1$ to $v_6$. If this cycle also includes vertex $v_7$, we are done. Otherwise, we show how to extend the cycle further based on one of the following cases. We consider a case only if all the previous cases have failed. Note that if we succeed in extending the cycle to be of length 7 or more, then we can 2-connect $D$ by shrinking the 7-cycle into a single node and then finding a cycle through the six or fewer remaining nodes (including the core).

To prevent $v_6$ from being a cut vertex, at least one of the vertices $\{v_1, v_2, v_3, v_4, v_5\}$ must cross $v_6$. In all cases below, let the back edge that crosses $v_6$ land on $v_k$, $k \geq \min(q, 7)$, and let it be a back edge that goes farthest up the tree.

*Case 1: $v_1$ and/or $v_5$ can cross $v_6$.* The cycle is extended to either $v_1 \ldots v_k v_1$ or $v_1 \ldots v_5 v_k \ldots v_6 v_1$.

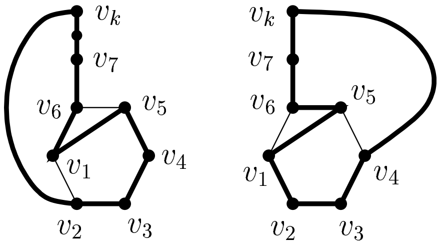

Figure 5: Lemma 4.5: Case 2

*Case 2: $v_2$ or $v_4$ can cross $v_6$ and $(v_1, v_5) \in E$.* The cycle is extended to either $v_1 v_5 v_4 v_3 v_2 v_k \ldots v_6 v_1$ or $v_1 v_2 v_3 v_4 v_k \ldots v_6 v_5 v_1$ (see Figure 5).

*Case 3: only $v_3$ crosses $v_6$ and $(v_1, v_5) \in E$.* Observe that nodes $v_2$ and $v_4$ can have no children in the DFS tree, because for example, if $v_2$ had a child $c$, then the algorithm would have reordered the current path to be $v_q, \ldots, v_6, v_1, v_5, v_4, v_3, v_2$ so that the DFS can be further extended to $c$ from $v_2$. Therefore the nodes $\{v_1, v_2, v_4, v_5\}$ have no edges that go out of the hexagon, and therefore there are at least 5 edges incident to them in any feasible solution to 2-ECSS. Hence, we can shrink the hexagon $H = \{v_1, \ldots, v_6\}$ in this case and call Shrink($H$).

*Case 4: only $v_3$ crosses $v_6$ and $(v_1, v_5) \notin E$.* Nodes $v_1$ and $v_5$ have no neighbors outside the hexagon $H = \{v_1, \ldots, v_6\}$ as explained in earlier cases. If $(v_1, v_4) \in E$, then expand cycle to $v_1 v_2 v_3 v_k \ldots v_6 v_5 v_4 v_1$ (see Figure 6 (a)). If $(v_2, v_5) \in E$, then expand cycle to $v_1 v_2 v_5 v_4 v_3 v_k \ldots v_6 v_1$ (see Figure 6 (b)). If $(v_3, v_5) \in E$, then we could have reordered the path as $v_q, \ldots, v_6, v_1, v_2, v_3, v_5, v_4$, and therefore $v_4$ has no neighbors outside the current path. Therefore in this case, any 2-ECSS must have at least 5 edges incident to $\{v_1, v_4, v_5\}$, and hence we call Shrink($H$).

We have considered all possible neighbors for $v_5$. If $v_5$ is a degree-2 node, then $v_4$ does not have a child $c$ other than $v_3$, because, since $v_4$ is not a cut vertex, there must be a back edge from a descendant of $c$ that goes farther than $v_4$. The back edge cannot land on $v_5$ because $v_5$ has no neighbors outside $H$, nor can the back edge land on any vertex beyond $v_6$, since in this case $v_4$ crosses $v_6$. Therefore any back edges can only land on $v_6$ and this makes $v_5$ a beta vertex

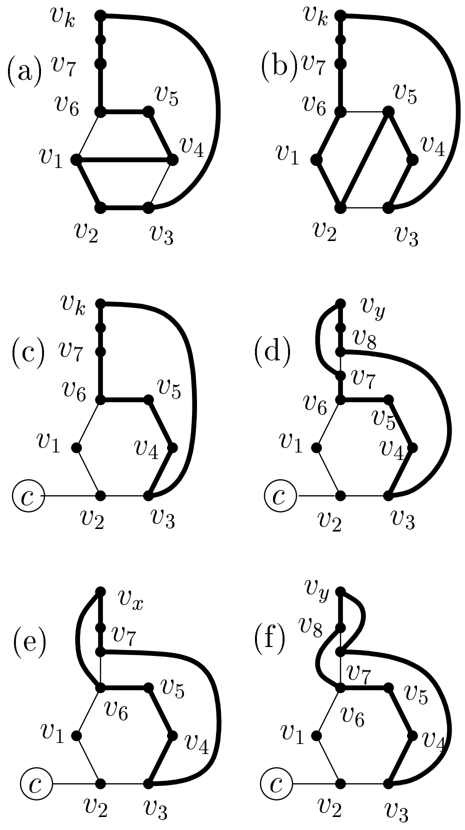

Figure 6: Lemma 4.5: Case 4

on the removal of $v_4$ and $v_6$ (the third component is the subtree with $c$ as the root). If $v_2$ has no child other than $v_1$, then we can call Shrink($H$), since there must be at least 5 edges incident to $\{v_1, v_2, v_5\}$ in any 2-ECSS.

In the only remaining possibility, let $c$ be a child of $v_2$ outside $H$. By arguing that the graph has no cut vertices as in Case 3 above, at least one of the following cycles exists; if needed, $v_2$ and $v_1$ are donated to $c$.

- Replace $(v_2, v_3)$ by plundering $c$.
- $v_5 v_4 v_3 v_k \ldots v_8 v_7 v_6 v_5$ (see Figure 6 (c)).
- $v_5 v_4 v_3 v_8 \ldots v_y v_7 v_6 v_5$ (see Figure 6 (d)).
- $v_5 v_4 v_3 v_7 v_8 \ldots v_x v_6 v_5$ (see Figure 6 (e)).
- $v_5 v_4 v_3 v_7 v_y \ldots v_8 v_6 v_5$ (see Figure 6 (f)).
- Call Shrink($H$).

*Case 5: only $v_4$ crosses $v_6$.* If $v_5$ has no other neighbors in $G$, it will be a beta vertex. As noted earlier, $v_5$ cannot have any children other than $v_4$.

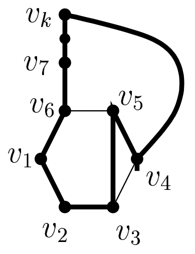

Figure 7: Lemma 4.5: Case 5

Therefore at least one of the edges $(v_1, v_5)$, $(v_2, v_5)$, or $(v_3, v_5)$ must exist. Edge $(v_1, v_5)$ is covered under Case 2. If $(v_3, v_5)$ exists, then we expand the cycle to $v_1v_2v_3v_5v_4v_k \dots v_6v_1$ (see Figure 7). The only case left to consider is when edge $(v_2, v_5)$ exists. In this case $v_3$ has no neighbors outside the path because the algorithm would rearrange the path as $v_q, \dots, v_6, v_1, v_2, v_5, v_4, v_3$ and extend the path further. Hence there are at least 5 edges incident to $\{v_1, v_3, v_5\}$ in any 2-ECSS, and therefore we call Shrink$(H)$.

*Case 6: only $v_2$ crosses $v_6$.* This case is symmetric to Case 5 above.

*Case 7: At least two consecutive nodes in the set $\{v_2, v_3, v_4\}$ have back edges that cross $v_6$.* We will discuss the case when we have back edges $(v_2, v_w)$ and $(v_3, v_x)$. The other cases will be handled similarly. If $w = 12$ (i.e., $v_2$ has a back edge to the core), then we can 2-connect $D$ using 2 ears: $v_{12}v_2v_1v_6 \dots v_{12}$ followed by $v_2v_3v_4v_5v_6$. The case $x = 12$ is similar.

For the rest of this case, we assume that any back edge from $\{v_2, v_3, v_4\}$ lands within the path $v_7 \dots v_{11}$. If there is a back edge from $v_2$ to $v_7$, then we expand the cycle to $v_1v_2v_7 \dots v_xv_3v_4v_5v_6v_1$. Also, if $v_7$ has a back edge to the core, we can 2-connect $D$ using 2 ears: $v_{12}v_7v_8v_9v_{10}v_{11}v_{12}$ followed by $v_wv_2v_1v_6v_5v_4v_3v_x$. Also, if $(v_7, v_{11}) \in E$, then also we can 2-connect $D$ with 2 ears either by expanding the 6-cycle as in the previous case by replacing $(v_2, v_3)$ by $v_2v_w \dots v_xv_3$ and then connecting the remaining nodes with another ear, or by finding a first ear that starts at the core and includes all nodes of the 6-cycle and then connect the remaining nodes using a second ear.

If any of the vertices $\{v_2, v_3, v_4, v_7, v_{w-1}, v_{x-1}\}$ has a child outside $D$, then we can donate up to 3 nodes and 2-connect the remaining nodes using 2 ears. For example, if $v_3$ has a child $c$, then we donate $\{v_3, v_4, v_5\}$ to $c$ and use the cycle $v_1v_2v_w \dots v_6v_1$ as the first cycle and we can 2-connect the 8 remaining nodes in the path using at most two ears.

If three or more nodes in $v_7, \dots v_{11}$ have back edges to the core, then we can 2-connect $D$ using two ears. On the other hand, if at most two nodes of $D$ are connected to the core, and the other nodes do not have any neighbors outside $D$, then any 2-ECSS of $G$ has at least 10 edges within the subgraph induced by $D$, and we can shrink $D$ into a single vertex (and we can 2-connect $D$ using at most 11 edges).

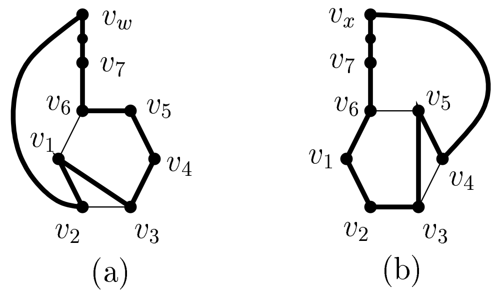

Figure 8: Lemma 4.5: Case 8

*Case 8: only $v_2$ and $v_4$ cross $v_6$.* Let the farthest back edge with which $v_2$ crosses $v_6$ land on $v_w$, and the corresponding back edge for $v_4$ land on $v_x$. If edge $(v_1, v_3)$ exists, then expand cycle to $v_1v_2v_w \dots v_6v_5v_4v_3v_1$ (see Figure 8 (a)). If edge $(v_3, v_5)$ exists, the expand the cycle to $v_1v_2v_3v_5v_4v_x \dots v_6v_1$ (see Figure 8 (b)). The case when edge $(v_1, v_5)$ exists was already considered in Case 2. If $\{v_1, v_3, v_5\}$ form an independent set, with none of the vertices connected to a node outside the hexagon, then all 6 edges of the hexagon are needed in any 2-ECSS solution, and therefore we can shrink the hexagon and we call Shrink$(H)$. The only case left is when node $v_3$ has a neighbor outside $H$. This can happen when there is a back edge from a descendant of $v_2$ that has a back edge to $v_3$. In this case, we replace $(v_2, v_3)$ by a path using the plunder operation. On the other hand, if $v_3$ has a child $c$ outside $H$, then we donate either $\{v_3, v_4, v_5\}$ or $\{v_3, v_2, v_1\}$ to $c$ depending on whether $w < x$ or not, and 2-connect the remaining nodes using two ears. ∎

## 5 Summary

In summary, our algorithm uses a 3-cycle-free maximal 2-matching to select a suitable depth-first search tree of the graph. The DFS tree is broken into a collection of paths and we 2-connect the paths one at a time. We account for edges by a charging method where we try to pay using vertex charges whenever possible and show that when we need extra, then path charges are available to cover the deficit. We introduced new operations of donation and plundering to handle some inconvenient configurations. For a graph, the algorithm either shrinks a subset of nodes $C$ (by calling SHRINK($C$)) when we are able to prove that at least $|C| - 1$ edges are necessary within $C$ in any 2-ECSS, or finds a solution that uses at most $\frac{5}{4}(n + |P|)$. A more precise statement of when we apply SHRINK($C$) is stated in Lemma 3.1.

THEOREM 5.1. *Given a 2-edge-connected, undirected graph $G = (V, E)$, there is a polynomial-time algorithm that returns a 2-ECSS of $G$ that is within* $5/4$ *of an optimal 2-ECSS.*

Instead of using a 3-cycle-free 2-matching, if an arbitrary maximal 2-matching is used, then it can be shown that the performance ratio of our algorithm is at most $21/16 = 1.3125$.

## 6 Vertex connectivity

Recently, we have identified how to extend our algorithm to 2-VCSS. There are several issues to solve. First, we need to show how SHRINK($C$) is handled for vertex connectivity. Also, we shrink cycles when they get large enough during the course of the algorithm. It needs to be shown that edges can be carefully chosen such that the resulting graph is 2-vertex-connected. In fact, in certain examples, extra edges need to be added when we expand $C$ after calling SHRINK($C$), and in these cases we are able to show that *Opt* is also bigger. Also, when we select edges incident to the core, we need to ensure that both edges that 2-connect a path $D$ are not incident to the same vertex of the core (i.e., we need to get an open ear decomposition). More details of the following theorem will be provided in the full version of the paper to be made available soon.

THEOREM 6.1. *Given a 2-vertex-connected, undirected graph $G = (V, E)$, there is a polynomial-time algorithm that returns a 2-VCSS of $G$ that is within* $5/4$ *of an optimal 2-VCSS.*

## References

[1] J. Cheriyan, A. Sebő, Z. Szigeti, *Improving on the 1.5 approximation of a smallest 2-edge connected spanning subgraph*, SIAM J. Discret. Math., **14**, pp. 170-180, 2001.

[2] J. Cheriyan, S. Vempala and A. Vetta, *Approximation algorithms for minimum-cost k-connected subgraphs*, Proc. of the 34th ACM Symposium on the Theory of Computing (STOC), 2002.

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

[4] A. Czumaj and A. Lingas, *On approximability of the minimum-cost k-connected spanning subgraph problem*, Proc. 10th Annual ACM-SIAM Symposium on Discret. Alg. (SODA), pp. 281-290, 1999.

[5] J. Edmonds, *Matroid intersection*, Annals of Discret. Math., **14**, pp. 39-49, 1979.

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

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

[8] H. N. Gabow, *An ear decomposition approach to approximating the smallest 3-edge connected spanning subgraph of a multigraph*, Proc. 13th Annual ACM-SIAM Symp. on Discret. Algorithms (SODA), pp. 84-93, 2002.

[9] H. N. Gabow, *Better performance bounds for finding the smallest k-edge connected spanning subgraph of a multigraph*, Proc. 14th Annual ACM-SIAM Symp. on Discret. Algorithms (SODA), 2003.

[10] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.

[11] N. Garg, V. Santosh and A. Singla, *Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques*, Proc. 4th Annual ACM-SIAM Symp. on Discret. Algorithms (SODA), pp. 103-111, 1993.

[12] D. Hartvigsen, *Extensions of matching theory*, Ph.D. Thesis, Carnegie-Mellon University, 1984.

[13] D. Hartvigsen, *The square-free 2-factor problem in bipartite graphs*, Proc. of the 7th Integer Programming and Combinatorial Optimization Conference (IPCO), pp. 234-241, 1999.

[14] A. Itai, C.H. Papadimitriou, and J.L. Szwarcfiter, *Hamilton paths in grid graphs*, SIAM J. Comput., pp 676-686, 1982.

[15] S. Khuller and B. Raghavachari, *Improved approximation algorithms for uniform connectivity problems*, J. Algorithms, **21**, pp. 433-450, 1996.

[16] S. Khuller and U. Vishkin, *Biconnectivity approximations and graph carvings*, J. Assoc. Comput. Mach., **41**, pp. 214-235, 1994.

[17] P. Krysta and V. S. Anil Kumar, *Approximation algorithms for minimum size 2-connectivity problems*, Proc. 18th Intl. Symposium on Theoretical Aspects of Computer Science (STACS), pp. 431-442, 2001.

[18] H. Nagamochi and T. Ibaraki. *Linear time algorithms for finding sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, **7**, pp. 583-596, 1992.

[19] S. Vempala and A. Vetta, *Factor 4/3 approximations for minimum 2-connected subgraphs*, APPROX 2000, pp. 262-273, 2000.