Factor $\frac{4}{3}$ Approximations for Minimum 2-Connected Subgraphs

Santosh Vempala and Adrian Vetta [†]

Department of Mathematics
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

We present factor $\frac{4}{3}$ approximation algorithms for the problems of finding the minimum 2-edge connected and the minimum 2-vertex connected spanning subgraph of a given undirected graph.

[†]Email: {vempala,avetta}@math.mit.edu

1. Introduction

The task of finding small spanning subgraphs of a prescribed connectivity is a fundamental problem in network optimization; it is also a hard problem. In fact, even the problems of finding the smallest 2-edge connected spanning subgraph ($2EC$) or the smallest 2-vertex connected spanning subgraph ($2VC$) are NP-hard. This can be seen via a simple reduction from the Hamiltonian cycle problem.

It is known that both these problem are MAXSNP-hard ([4], [1]). However, constant factor approximation algorithms have been developed. For example, it is easy to find a 2-edge connected spanning subgraph that has no more than twice as many edges as the optimum; take the edges of a depth-first search tree on the graph along with the deepest back-edge from each vertex to obtain a 2-connected subgraph. In an n vertex graph this produces a solution with at most $2n-2$ edges. Since the optimum has at least n edges, this is a 2-approximation. Khuller and Vishkin [8] gave a $\frac{3}{2}$-approximation algorithm for $2EC$. Cheriyan et al [3] have recently improved upon this, with a $\frac{17}{12}$-approximation algorithm. In [8], Khuller and Vishkin also gave a $\frac{5}{3}$ approximation algorithm for $2VC$. This was improved to $\frac{3}{2}$ in [5].

This paper presents $\frac{4}{3}$-approximation algorithms for both $2EC$ and $2VC$. The ratio $\frac{4}{3}$ has a special significance; a celebrated conjecture in combinatorial optimization says that the travelling salesman problem on metrics is approximable to within $\frac{4}{3}$ via a linear programming relaxation called the *subtour relaxation*. A (previously unverified) implication of this conjecture is that $2EC$ is also approximable to within $\frac{4}{3}$.

An overview of our approach is as follows. In Section 2 we present the basic lower bound against which the performance of our algorithm will be measured. To obtain this bound we find the minimum sized spanning subgraph in which every vertex has degree at least two. This can be done in polynomial-time via a reduction to the maximum matching problem. Section 3 contains decomposition theorems which are the basis of our algorithm. These decompositions allow us to focus on graphs which do not contain certain problematic structures. In such graphs the degree two subgraphs can be modified to obtain a solution to the $2EC$ or $2VC$ without increasing their size by more than a third. The bulk of the analysis lies in showing that such a low-cost modification is possible. This involves detailed arguments for several tricky configurations; these are given in Section 4 and Section 5 for the edge connectivity and vertex connectivity cases, respectively. The running time of the algorithm is $O(n^2m)$, where m is the number of edges in the graph.

2. The Lower Bound

The idea behind the algorithms is to take a minimum sized spanning subgraph with minimum degree 2 and, at a small cost, make it 2-edge connected. We will denote by $D2$ the problem of finding a minimum sized spanning subgraph in which each vertex has degree at least two. Notice that the solution to $D2$ (which may not be connected) provides a lower bound for both $2EC$ and $2VC$.

Lemma 2.1. *The size of the optimal solution to $D2$ is a lower bound on the size of the optimal solutions to $2EC$ and $2VC$.*

Proof. Any 2-connected subgraph must have minimum degree two. □

This gives a possible method for approximating $2EC$ and $2VC$. Find an optimal solution to $D2$. Alter the solution in some way to get a 2-connected graph. The rest of the paper is devoted to specifying the "some way" and proving that the method gives a solution whose size is at most $\frac{4}{3}$ the size of the optimum solution for $D2$. Hence it is at most $\frac{4}{3}$ the size of the optimum solution for $2EC$ or $2VC$.

The conversion of a solution of $D2$ to a solution of $2EC$ or $2VC$ will be quite fast and the overall running time is dominated by the time to solve $D2$. The latter problem can be solved exactly in polynomial time. One fast way to do this is to find a maximum sized subgraph in which each vertex has degree at most two (this is the *maximum cycle-path cover* problem which asks for the maximum sized subgraph consisting of disjoint cycles and paths) and add an arc to the end vertices of each path. This will give an optimal solution to $D2$. This follows from the fact that if the minimum $D2$ solution contains $n+p$ edges then there are $2p$ edges whose removal induces a collection of p disjoint paths plus some disjoint cycles. The maximum cycle-path cover problem has a simple reduction to maximum matching so the running time is $O(m\sqrt{n})$.

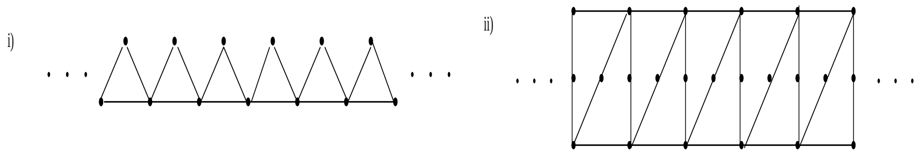

Figure 1.

How good a lower bound is our solution to $D2$ though? The following examples show that, on its own, it is not sufficient for our purposes. Consider finding a minimum 2-edge connected subgraph in the graph shown in Figure 1i). It is easy to see that the optimal solution is $\frac{3}{2}$ times larger than the optimal solution to $D2$. Similarly if we wish to find a minimum 2-vertex connected subgraph in the graph in Figure 1ii), it is easy to see that this is also $\frac{3}{2}$ times larger than the optimal solution to $D2$. We will address the implications of these examples in the next section.

3. The Decomposition Theorems

In this section we present decomposition procedures that allow us to focus upon graphs with simplified structure. The decomposition procedure for edge connectivity is significantly more involved than the one for vertex connectivity.

Edge connectivity.

Cut Vertices Dealing with the problem illustrated by the example given in Figure 1i) is not difficult. We assume that for $2EC$ our graph contains no cut vertices. This condition entails no loss of generality. An α-approximation algorithm for 2-vertex connected graphs can be used to give α-approximate solutions for general graphs. Simply apply the algorithm separately to each maximal 2-vertex connected block, then combine the solutions to each block.

We remark that in a 2-vertex connected graph, in which multiple edges are allowed, there is an optimal 2-edge connected subgraph that does not contain multiple edges. Therefore, when considering graphs with multiple edges we may focus on the underlying simple graph. We will also contract together vertices during the course of the algorithm; the previous observation shows us how to deal with the multiple edge that may result.

We may also assume that G has no adjacent degree 2 vertices. This also entails no loss of generality. Observe that both edges incident to a degree two vertex must be contained in any 2-edge connected spanning subgraph. Hence, for a path whose internal vertices are all degree 2, every edge in the path must be chosen. This allows us to transform an instance of the general case into an instance with non-adjacent degree 2 vertices. Given G, contract, to just two edges, any such path. Call the resulting graph G'. Given the trivial bijection between solutions to G and solutions to G', it follows that any α-approximate solution to G' induces an α-approximate solution to G.

Beta Structures The example of Figure 1ii) is more troublesome. To counter it we need to eliminate a certain graph structure, which we will call a *beta-structure.* A vertex u is a *beta-vertex* if the removal of two vertices v_1 and v_2 induces at least three components, one of which consists of just u. This situation is illustrated in Figure 2, with the other two components labelled C_1 and C_2. Without loss of generality, we may assume that the component C_1 is smaller (i.e. contains fewer vertices) than C_2.

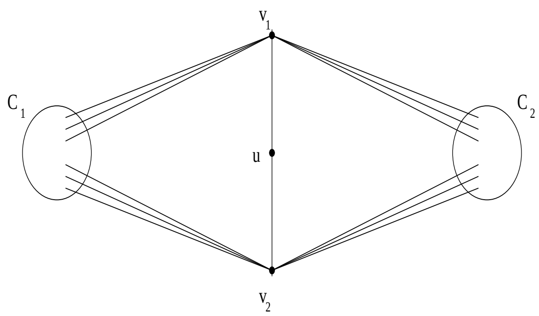

FIGURE 2. A beta-vertex.

We say that two adjacent vertices u_1 and u_2 are a *beta-pair* if there exist two nodes, v_1 and v_2, whose removal leaves at least 3 components, one of which contains just u_1 and u_2.

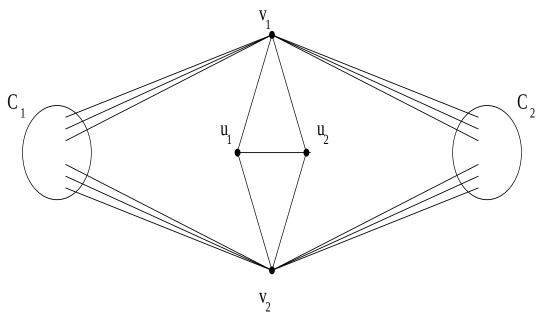

FIGURE 3. A beta-pair.

This situation, with the other two components labelled C_1 and C_2, is shown in Figure 3. Note, the figure shows all four edges $(u_1, v_1), (u_1, v_2), (u_2, v_1)$ and (u_2, v_2) present. It may be the case that, for a beta-pair, only three of these edges are present. This will not affect our argument in any way. In addition, for both beta-vertices and beta-vertices we may assume that the edge (v_1, v_2) is not present in the graph G. To see this note that there is some optimal subgraph in G that does not contain the edge (v_1, v_2); thus it suffices to find an α-approximate solution in $G - (v_1, v_2)$. We say that a graph without a beta-vertex or beta-pair is *beta-free*.

Here we will show how to recursively obtain, in polynomial time, a lower bound $l(G)$ on the optimal solution. In the following sections we will show how to augment the solution induced by the lower bound to give a feasible 2-edge connected subgraph whose size is at most $\frac{4}{3}$ that of the lower bound. The lower bound is obtain by decomposing the graph around beta-structures.

First we show that the case of beta-pairs can be reduced to that of beta-vertices. Observe that both edges incident to a beta-vertex u must be in any solution. These two edges induce a path of length two from v_1 to v_2. Similarly given a beta-pair u_1 and u_2 it is easy to show that there is some optimal solution which contains a path of length three from v_1 to v_2 using the edge (u_1, u_2). It follows that if we find a beta-pair u_1 and u_2 in G we may contract the edge (u_1, u_2) and solve our problem on a smaller graph, denoted $G/(u_1, u_2)$, that contains a beta-vertex. Note, also, that this edge contraction reduces the size of the optimal solution in the graph by exactly one; the lower bound that we present will also be reduced by exactly one. Any approximation guarantee we may obtain, via the lower bound, for the contracted graph $G/(u_1, u_2)$ will then apply to the original graph G. Thus, it suffices to assume that all our beta-structures are beta-vertices.

Our basic technique to deal with a beta-vertex is as follows. We will decompose the graph into two pieces around the beta-vertex and iterate the procedure. Eventually we will have a decomposition into beta-free graphs. We will then work on these pieces and show that the sub-solutions obtained can be combined to give a good solution to the whole graph. Given a beta-vertex, u, with components C_1 and C_2, let G_1 be the subgraph formed by removing C_2 and its incident edges. Let G_1' be the graph formed by contracting $V - C_1$. These situations are shown in Figure 4i) and ii), where the super-node formed by contracting $V - C_1$ is labelled v'. The graphs G_2 and G_2' are defined analogously. Let k_1, k_1', k_2 and k_2' be the sizes

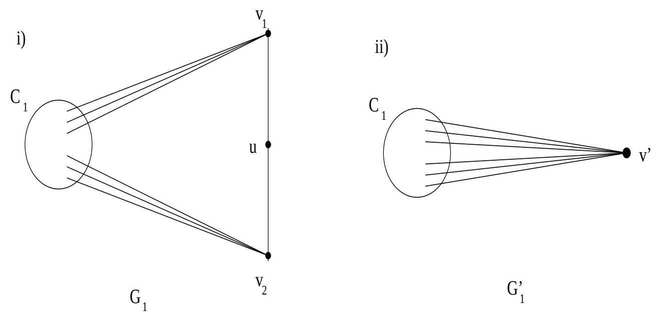

FIGURE 4. A split.

of the optimal 2-edge connected spanning subgraphs of G_1, G'_1, G_2 and G'_2, respectively. Let $\mathrm{OPT}(G)$ be the size of the optimal solution to the whole graph.

Lemma 3.1. *Given a 2-edge connected graph,* $\mathrm{OPT}(G) = \mathrm{Min}(k_1 + k'_2, k'_1 + k_2)$.

Proof. It is clear that the combination of a solution to G_1 and G'_2, or vice versa, gives a 2-edge connected subgraph for G. Conversely any solution to G can be decomposed to give a solution to G_1 and G'_2, or to give a solution to G'_1 and G_2. □

In addition, there is the following relationship between k_i and k'_i.

Lemma 3.2. *For* $i = 1, 2$, *we have* $k'_i + 2 \leq k_i \leq k'_i + 3$.

Proof. First note that to connect up u we must use both its incident edges. It follows that $k'_i + 2 \leq k_i \leq k'_i + 3$, $i = 1, 2$. This is because, given a 2-edge connected subgraph for G'_i, we are forced to use 2 extra edges in connecting up u and we may need to use an extra edge from C_i in certain instances in which the solution to G'_i had all its edges leaving C_i going to the same vertex (either v_1 or v_2). □

From Lemma 3.1 and Lemma 3.2, we obtain the following simple corollary.

Corollary 3.3. *If G has a beta-vertex inducing the components C_1 and C_2 then*

$$\begin{aligned} &(a) \qquad k'_1 + k'_2 + 2 \;\leq\; \mathrm{OPT}(G) \leq k'_1 + k'_2 + 3 \\ &(b) \qquad k_1 + k_2 - 3 \;\leq\; \mathrm{OPT}(G) \leq k_1 + k_2 - 2 \end{aligned}$$ □

(Note that if the components C_1 and C_2 were induced by a beta-pair not a beta-vertex then we would obtain, for example, the inequalities $k'_i + 3 \leq k_i \leq k'_i + 4$, $i = 1, 2$. The lower bound defined below may then be constructed with regards to both beta-pairs and beta-vertices. However, it is cleaner to use the fore-mentioned reduction of beta-pairs to beta-vertices.)

To compute optimally the values k_1, k'_1, k_2 and k'_2 is as hard as the original problem. So instead we will need to find approximate solutions. First, though, we present a lower bound on $\mathrm{OPT}(G)$ that is defined recursively. If G is beta-free then let $l(G)$ be given by the lower bound provided by $D2$. If G contains a beta-vertex then we evaluate $l(G_1)$ and $l(G'_1)$ and, based upon their relative magnitudes, we either set $l(G) = l(G_1) + l(G'_2)$ or set $l(G) = l(G'_1) + l(G_2)$. The exact procedure is:

```
LowerBound(G)
{
If G has a cut vertex v
   Then let l(G) = Σ_i l(G_i), where the G_i are blocks induced by v
If G has two adjacent degree two vertices v_1 and v_2
   Then let l(G) = l(G/(v_1,v_2))+1
If G has a beta-pair u_1,u_2
   Then let l(G) = l(G/(u_1,u_2))+1
If G has a beta-vertex u
   Then if l(G_1) ≤ l(G'_1) + 2 let l(G) = l(G_1) + l(G'_2)
   Else if l(G_1) ≥ l(G'_1) + 3 let l(G) = l(G'_1) + l(G_2)
Else let l(G) = |D2(G)|
}
```

This recursive lower bound procedure induces the following decomposition into 2-vertex connected, beta-free subgraphs.

```
Decompose(G)
{
If G has a cut vertex v
   Then output Decompose(G_i), for all blocks G_i induced by v
If G has two adjacent degree two vertices v_1 and v_2
   Then output Decompose(G/(v_1,v_2))
If G has a beta-pair u_1,u_2
   Then output Decompose(G/(u_1,u_2))
If G has a beta-vertex
   If l(G_1) ≤ l(G'_1) + 2 then output Decompose(G_1) and Decompose(G'_2)
   If l(G_1) ≥ l(G'_1) + 3 then output Decompose(G'_1) and Decompose(G_2)
Else output G
}
```

Since we have that $|C_1| \leq |C_2|$, this recursive procedure can be applied in polynomial time.

Theorem 3.4. *The decomposition procedure can be implemented in polynomial time.*

Proof. Let $f(n,m)$ be the running time for our algorithm on a graph of n vertices and m edges. The time required to search for possible decompositions in G is $O(nm)$. In addition, if G is beta-free then we require $O(m\sqrt{n})$ to find a solution to $D2$. For graphs with beta-vertices, the running time is then at most $f(n,m) = f(n_1,m_1) + f(n_1-2, m_1-2) + f(n_2,m_3) + O(nm)$, where $m_1 + m_2 = m, n_1 + n_2 = n+3$ and $n_1 \leq n_2$; that is, $f(n,m) \leq 2f(n_1,m_1) + f(n_2,m_2) + O(nm)$. For graphs with a cut vertex inducing k blocks, the running time is at most $f(n,m) = \sum_i f(n_i, m_i) + O(nm)$, where $\sum_i m_i = m$ and $\sum_i (n_i - 1) = n - 1$. Solving the resultant recurrence we get that $f(n,m) = O(n^2 m)$. $\square$

We now prove that $l(G)$ is indeed a lower bound on the size of the optimal 2-edge connected subgraph.

Theorem 3.5. *Given a 2-edge connected graph, $l(G) \leq \mathrm{OPT}(G)$.*

Proof. We proceed by induction. If G is beta-free then $l(G)$ is given by the solution to $D2$, a lower bound on the optimal solution. Suppose that G contains a beta-vertex. By construction of $l(G)$, we have two possibilities:
(i) $l(G_1) \leq l(G_1') + 2$: thus we have

$$\begin{aligned} l(G) &= l(G_1) + l(G_2') && \text{by LowerBound}(G) \\ &\leq l(G_1') + 2 + l(G_2') && \\ &\leq k_1' + 2 + k_2' && \text{by induction} \\ &\leq \mathrm{OPT}(G) && \text{by Corollary 3.3a)} \end{aligned}$$

(ii) $l(G_1) \geq l(G_1') + 3$: thus we have

$$\begin{aligned} l(G) &= l(G_1') + l(G_2) && \text{by LowerBound}(G) \\ &\leq l(G_1) - 3 + l(G_2) && \\ &\leq k_1 - 3 + k_2 && \text{by induction} \\ &\leq \mathrm{OPT}(G) && \text{by Corollary 3.3b)} \end{aligned}$$

The induction step clearly applies if we decompose around a cut vertex, contract two adjacent degree two vertices, or contract a beta-pair. $\square$

Corollary 3.6. *Given an algorithm that finds a solution with at most $\alpha|D2(G)|$ edges for any beta-free graph, we get an α-approximation algorithm for any graph.*

Proof. Consider the decomposition tree induced by the decomposition procedure. Let $H_1, H_2, \ldots H_p$ be its leaves. Then $OPT(G) \geq l(G) = |D2(H_1)| + |D2(H_2)| + \cdots + |D2(H_p)|$, whilst the number of edges used by the algorithm is at most $\alpha\,(|D2(H_1)| + |D2(H_2)| + \cdots + |D2(H_p)|)$. $\square$

Thus, it suffices to consider only beta-free graphs. It is in these graphs that we will find $\frac{4}{3}$-approximate solution based upon the subgraph induced by the $D2$ solution.

Vertex connectivity.

Beta Structures Consider the case of a beta-vertex u for the 2-vertex connectivity problem. We deal with this situation as follows. Given G, let $G' = G - u$. See Figure 5i) and ii). Given that the two edges incident to u are forced in any solution to G, we have the following simple lemma.

Lemma 3.7. *Consider the problem $2VC$ on a graph G that contains a beta-vertex u. An α-approximation for G can be obtained from an α-approximation for G' by adding the arcs incident to u.* $\square$

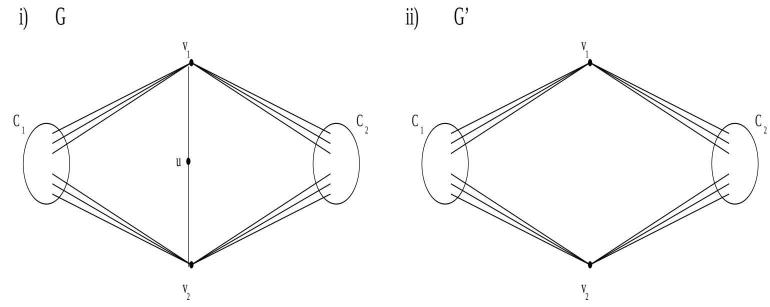

Figure 5.

So for $2VC$ our decomposition is straightforward. Remove the beta-vertex and consider the induced subproblem.

4. Edge Connectivity

For a beta-free graph, our lower bound is the same for both $2EC$ and $2VC$. Since a $2VC$ solution is also a $2EC$ solution we could simply use that for $2EC$ on beta-free graphs. Along with the $2EC$ decomposition procedure, and the guarantee that the $2VC$ solution is within $4/3$ of the $D2$ bound on beta-free graphs (see next section), we get a $4/3$ approximation for $2EC$ on general graphs. However, the $2EC$ problem seems to be considerably simpler to solve conceptually as well as in the details; so we describe a separate solution for it.

4.1. **The Algorithm.** Recall that we find an optimal solution to $D2$ by adding edges to the end vertices of each path in an optimal solution to the maximum cycle-path cover problem, i.e. a collection of disjoint cycles and paths with as many edges as possible. We begin with these path and cycle components (including single nodes if any, which represent paths or cycles of length 0). Our next step is to add a collection of edges so that these components become connected. Our basic algorithm is the following.

```
EdgeConnAlg(G)
{
Partition the D2 solution into path and 2-edge connected components.
Select a 2-edge connected component to be the root component R.
Repeat until there is only one component
    Select a target component C.
    Add edges to 2-edge connect together R and C.
    Call the resulting component the root R.
}
```

Lemma 4.1. *The algorithm EdgeConnAlg produces a feasible solution.*

Proof. First note that there must be at least one 2-edge connected component after we partition our $D2$ solution. Thus, the root component is always 2-edge connected during the course of the algorithm. ☐

So this basic algorithm produces a feasible solution. It remains to specify exactly how we choose a target component, which edges we add in each step, and also to analyse the performance of the algorithm i.e. bound the total number of edges that we add during this process.

A tree of components. In order to choose a target component we first grow a tree, T, on the graph induced by the components, rooted at the root component. The set of edges we add will induce a spanning tree on the graph induced by the components i.e. on the graph whose vertices are the components. Call the vertices of the tree "nodes" to distinguish them from the vertices of our original graph. We grow the tree as follows. Edges will be chosen in a depth first search manner; however, since each node represents a component in the graph we also need a priority structure (on the vertices within a node) which specifies the order in which edges from each node are examined. So suppose we enter the node along a tree edge (u, v). We have three cases.

(1) The node represents a cycle: first search for edges incident to vertices adjacent to v in the cycle; next consider the vertices at distance 2 from v along the cycle, etc. Finally consider v itself.
(2) The node represents a path: give priority to the "left" endpoint (where one of the path's two endpoints is arbitrarily chosen to be the left endpoint), with declining priority as we move rightwards.
(3) The node is 2-edge connected: in this case, where the node is not a cycle, search the vertices in any order.

Building the tree in a depth first search manner gives the nice property that every edge in the graph is between a descendant node and one of its ancestor nodes (except for edges between vertices in the same node). We will call these edges "back edges" signifying that they go back towards the root. Given the tree, our target component will be the first node added to T after the root component. (In subsequent phases we may select the next target component by either growing a tree from the enlarged root or simply by considering the tree obtained by contracting together the old root and the target component). Next we describe how we deal with the target component and provide approximation guarantees concerning the performance of the algorithm.

4.2. **Performance Analysis.** Given our current subgraph, the components within a tree structure, how do we create a good 2-edge connected subgraph? In a given phase, any edges that we add will be *charged* to current target component. Our proof will follow from the fact that no component is charged more than one third its own weight. First some terminology. Each component has one tree edge going up towards the root node. This edge we will call an *upper tree edge* w.r.t. the component. The component may have several tree edges going down towards the leaf nodes. These will be called *lower tree edges* w.r.t. the component.

We examine in turn the situation in which the target component is a path, 3-cycle, 4-cycle, 5-cycle and a cycle with at least 6 edges. When the target component P is a path or a 3-cycle, then we also ensure that the endpoints of two edges connecting P to the current root component R have disjoint endpoints in R. We also ensure that one of the final edges chosen has the same endpoint in R as the original tree edge connecting R to P. This technical condition will be quite useful.

Lemma 4.2. *For path nodes the charge is at most $\lceil \frac{k}{3} \rceil + 2$, where k is the number of edges in the path.*

Proof. Consider a path, P, of $k+1$ vertices $v_0, v_1, \ldots, v_k$. Let the upper tree edge e^* of the path be incident to the vertex v^*. Let the subtree rooted a path node v_i be called T_i. This situation is shown in Figure 6. Now we wish to obtain a 2-edge connected component that contains both the root node R and the path node P by adding as few as edges as possible. We will break this problem into two paths. Firstly we will 2-edge connect up the vertices on the path. Secondly we will 2-edge connect up the path and the root R.

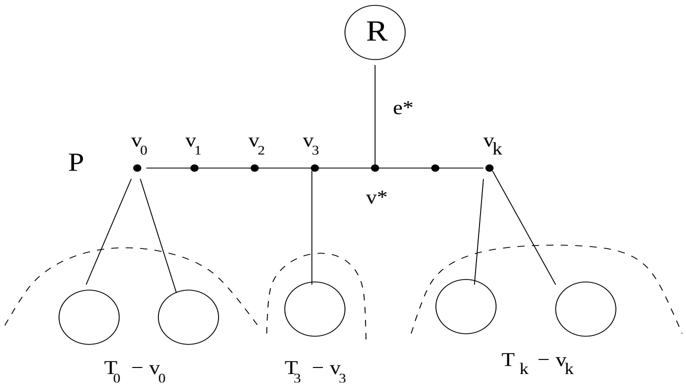

FIGURE 6. Subtrees.

To begin we contract the root node into v^*. We make two observations, concerning edges between vertices in P, that follow from the DFS priority scheme used to choose tree edges. First, there are no edges that correspond to edges incident to a vertex in $T_i - v_i$ and another in $T_j - v_j$, $i \neq j$. Such edges can not exist as they would have been chosen instead of the tree edge e_j, where we assume that e_i was chosen before e_j. Second, consider an edge in P corresponding to an edge between a vertex in $T_i - v_i$ and a vertex in P. This edge is between v_i and some vertex v_j to the right of v_i in the path P, (possibly $v_i = v_j$), otherwise it would have been selected before e_i in the DFS procedure.

We now show that the cost of 2-connecting up P is at most $\lceil \frac{k}{3} \rceil$. To show this, we break up the path into blocks of at least 3 edges such that each block is charged only one extra edge in the connection process. Start with the leftmost vertex in the path. We add rightward edges (from vertices on the path or their subtrees) wherever a path edge threatens to be a bridge. On adding an edge it may be possible to remove some current edge. If not, then all of the vertices to the left of the current bridge, not contained in any other block, form a new block. The blocks formed have the property that, except for the blocks containing a path endpoint, they contain at least three edges. To see this, let (u_0, u_1) be the current bridge, with the last block closed at u_0. Let u_2 and u_3 be the two vertices to the right of u_1. First assume that neither of u_1 and u_2 is the root component. Choose the edge, e, from the last block reaching furthest to the right. Now e is not incident to u_1, otherwise u_1 is a cut vertex. If e is incident to u_2 then there is a rightward edge e' out of T_1, otherwise u_2 is a cut vertex. We can choose e' for free by dropping the edge (u_1, u_2), see Figure 7. If e goes to u_3 or beyond then we are already done. Since the block containing the right endpoint does not contribute an extra edge, we have added an extra $\lceil \frac{k}{3} \rceil$ edges.

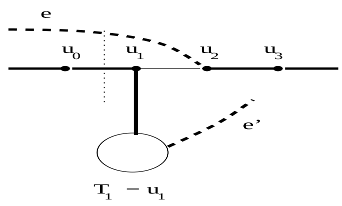

FIGURE 7. Paths.

Now to 2-edge connect up the path P and the root R we need two edges between P and R. Thus, in addition, to the tree edge e^* we add another edge f. Moreover, since the graph is 2-vertex connected, we may choose f to be incident to a different vertex in R than e^* (this property will simplify our subsequent analyses). So to deal with P we add, at most, an extra $\lceil \frac{k}{3} \rceil + 2$ edges.

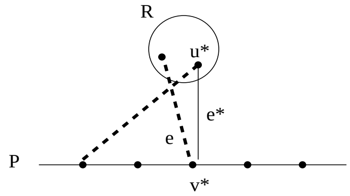

FIGURE 8.

We have one caveat: there is one block in which u_1 or u_2 may be the contracted vertex $v^* \cup R$. In this case the arguments with regard to cut vertices may not apply. As a result, we may be forced to add an extra edge in internally connecting the path. However, in such a circumstance, it is easy to ensure that the additional edge is incident to the root $R \setminus \{u^*\}$ (possibly by switching e^* for another edge e that has the same endpoint in R - see Figure 8 for an example). Thus, although we use an extra edge while internally 2-edge connecting P, we save one edge in connecting together P and R. □

Recall, that associated with P we have two free edges, (i.e. $k+2$ edges in total), so these extra two edges are accounted for. Further, $\lceil \frac{k}{3} \rceil \leq \frac{1}{3}(k+2)$ so the associated blow-up factor for a path is at most $\frac{4}{3}$.

Note that an added edge e (or e') could be from $T_i - v_i$ for some v_i. That is, the edge emanates from a descendant component C_t of P. Let $C_1, C_2, \ldots, C_{t-1}$ be the other components in the path from v_i to C_t in the DFS tree. Then each component $C_1, C_2, \ldots, C_t$ is also 2-edge connected up to the root for the cost of just one arc (the upper tree arc of that component). If a C_i is already a 2-edge connected component then the charge to that component is at most one third the size of the component. If the C_i is a path, then we have already added two edges 2-connecting the path to the root component for a charge of only one edge (the tree edge) to the path.

Lemma 4.3. *The charge to a* 3*-cycle is at most one edge.*

Proof. We will label a back edge from the current node going up the DFS tree by b. A back edge from below that is incident to the node will be labelled f. The deepest back edge from the descendants of the node will be labelled d. In all of our diagrams we use the following scheme. Solid edges were already selected at the point of examination of the component; dashed edges were not. Bold edges are selected by the algorithm after the examination; plain edges are not.

(I) Lower tree edges are all adjacent at the same vertex as the upper tree edge. Assume this vertex is v_1. Note that neither v_2 nor v_3 have incident edges going down the tree, otherwise the DFS algorithm would have chosen them as tree edges. Now v_1 is not a cut vertex, so there must be a back edge b from v_2 or v_3. Suppose it is incident to v_3; the other case is similar. We may add b and remove the edge (v_1, v_3) to 2-connect upwards the node. This is shown in Figure 9.I)1). Again, it will be useful, if when dealing with 3-cycles, the tree edge e and back edge b are incident to different vertices, say u_1 and u_2 in R. If we can not choose b in such a manner, then there must be an edge e' emanating from a subtree rooted at v_1 that is incident to different vertex than b in R. In such a case we may remove e and (v_1, v_3) and add e'. See Figure 9.I)2).

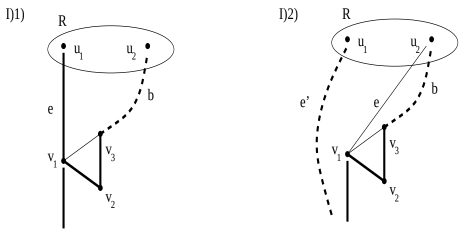

FIGURE 9. 3 cycles.

(II) The upper tree edge is incident to a vertex which has an adjacent vertex that is incident to a lower tree edge.

(1) There is a back edge b from v_2 or v_3. We may assume it is from v_3. The other case is similar. Add b and remove the edge (v_1, v_3). If the endpoints of b and e are disjoint then we are done. See Figure 10.II)1)a). Otherwise, there must be an edge e' emanating from a subtree rooted at v_1 that is incident to different vertex than b in R. In such a case we may remove e and (v_2, v_3) and add e'. See Figure 10.I)1)b).
(2) Otherwise, since v_1 is not a cut vertex, there is an edge d, emanating from a subtree rooted at v_2, going beyond v_1. Moreover, d can be chosen so that it does not share an endpoint with e in R. Add d and remove (v_1, v_2). See Figure 10.II)2). □

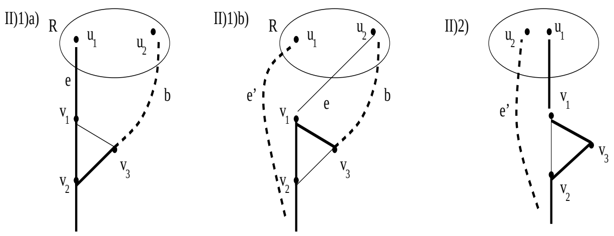

FIGURE 10. More 3 cycles.

Observe that it may be the case that the 3-cycle is a leaf component. Clearly this does not effect the argument above. For example any edge emanating from a subtree rooted at v_i, say, must in fact emanate from v_i.

Lemma 4.4. *The charge to a* 4*-cycle is at most* $\frac{4}{3}$ *edges.*

Proof.
(I) Lower tree edges are all adjacent at the same vertex as the upper tree edge.

(1) There is a back edge b from v_2 or v_4. We may assume it is from v_4; the other case is similar. Add b and remove the edge (v_1, v_4). This is shown in Figure 11.I)1).
(2) Otherwise, since v_1 is not a cut vertex, there is a back edge b incident to v_3. In addition, v_4 is not a beta-vertex. Therefore, as there is no edge from v_4 going down the tree (otherwise it would be a tree edge) there must be an edge between v_2 and v_4. Add b and (v_2, v_4). Remove (v_1, v_4) and (v_2, v_3). See Figure 11.I)2).

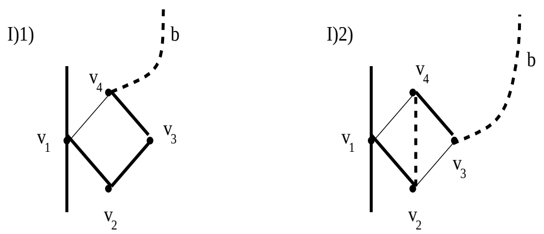

FIGURE 11. 4 cycles.

(II) The upper tree edge is incident to a vertex which has an adjacent vertex that is incident to a lower tree edge.

(1) There is an edge d (emanating from a subtree rooted at v_2) going beyond v_1. Add d and remove (v_1, v_2). See Figure 12.II)1).

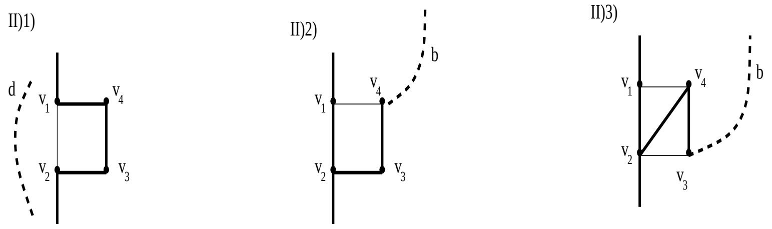

FIGURE 12. More 4 cycles.

(2) There is a back edge b from v_2 or v_4. We may assume it is from v_4; the other case is similar. Add b and remove the edge (v_1, v_4). This is shown in Figure 12.II)2).
(3) Otherwise, since v_1 is not a cut vertex, there is a back edge b incident to v_3 (or, similarly, an edge d emanating from a subtree rooted at v_3). In addition, v_4 is not a beta-vertex. Therefore, either

there is an edge between v_2 and v_4 or there must be an edge f from v_4 going down the tree. In the former case, see Figure 12.II)3), add b and (v_2, v_4). Remove (v_1, v_4) and (v_2, v_3). In the latter case we have two possibilities.

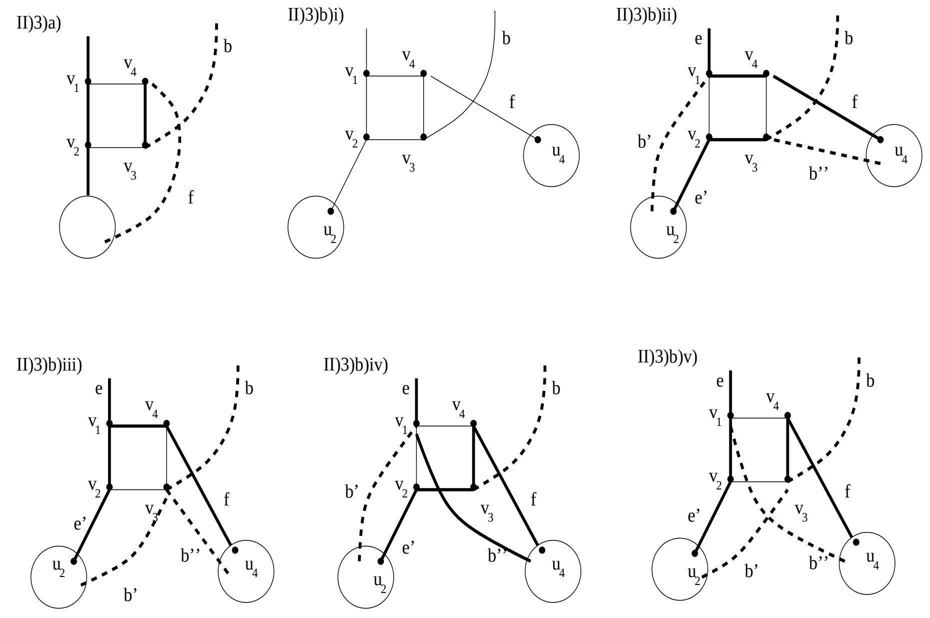

FIGURE 13. Even more 4 cycles.

(a) First suppose that the lower tree edge and f are incident to the same subtree below the 4-cycle. Then add b and f, remove also (v_1, v_4) and (v_2, v_3). See Figure 13.II)3)a).

(b) Otherwise the lower tree edge $e' = (v_2, u_2)$ and the edge $f = (v_4, u_4)$ originate in subtrees T_2 and T_4 rooted at v_2 and v_4, respectively. See Figure 13.II)3)b)i). If the top component of T_2 is a 3-cycle or a path, then we 2-connect it upwards first, and let b' be the edge chosen in addition to e' going above T_2. We can assume that b' is not incident to v_2. If the top component of T_2 is some other type of component, we simply pick any edge b' that does not share an endpoint with e' on the 4-cycle. Now b' must be incident to either v_1 or v_3. Similarly we can find a back edge b'' from T_4 that is not incident to v_4, but is incident to either v_1 or v_3. The case where b' is incident to v_1 and b'' is incident to v_3 is shown in Figure 13.II)3)b)ii). (the other three cases are similar and are shown in Figure 13.II)3)b)iii)-v)). We select b, b', b'', e, e', f and remove (v_1, v_2) and (v_3, v_4). Now lets analyse the performance of this method. First assume that the top components of T_2 and T_4 are 2-connected components with at least 4 edges each. We have used 6 edges to connect us up to the root and removed 2 edges. The total charge is 4 which is at most a third of the 12 or more edges of the three components. If the top component of T_2 or T_4 (say T_2) is a path, then the cost of e' and b' is covered by the path. Similarly, if one of these components is a 3-cycle, say for T_2, then while adding b' we also removed some edge from the 3-cycle, so the cost of b' and e' is covered by the 3-cycle. This makes the average charge lower than $\frac{4}{3}$. [We remark that the average charge is even lower if, say b' emanated from a component below a top component of T_2. The worst-case is when such back arcs emanate from top components].

(III) The upper tree edge is incident to a vertex which has an opposite vertex that is incident to a lower tree edge, but adjacent vertices are not incident to a lower tree edge.

(1) There is a back edge b from v_2 or v_4. We may assume it is from v_4. Add b and remove the edge (v_1, v_4).
(2) Otherwise, note that v_4 is not a beta-vertex. Therefore, as there is no edge from v_4 going down the tree (otherwise it would be a tree edge) there must be an edge between v_2 and v_4. In addition, since v_1 is not a cut vertex, there is either an edge d emanating from the subtree rooted at v_3 going beyond v_1. Add (v_2, v_4) and d, remove (v_1, v_4) and (v_2, v_3). See Figure 14.III)2). □

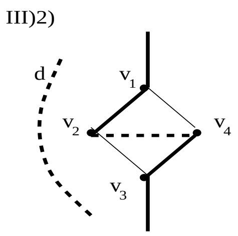

FIGURE 14. Yet another 4 cycle.

Lemma 4.5. *The charge to a 5-cycle is at most $\frac{5}{3}$ edges.*

Proof.
(I) Lower tree edges are all adjacent at the same vertex as the upper tree edge.

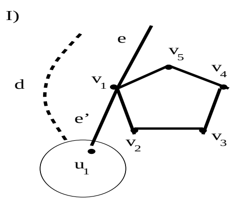

FIGURE 15. A 5 cycle

We 2-connect upwards the top component of T_1, the subtree rooted at v_1. If it is a path or a 3-cycle we proceed as in Lemmas 4.2 and 4.3. If it is a 2-connected component with 4 or more edges, we simply pick any edge d incident to R (there is such an edge as v_1 is not a cut vertex). The 3 edge charge is covered by 9 or more edges, see Figure 15.I). If the top component of T_1 is a path then e' and d can be charged to the path. If the top component is a 3-cycle, then we can choose d and remove an edge from the 3-cycle. In both these cases, the edge d is for free for the current 5-cycle.

(II) The upper tree edge is incident to a vertex which has an adjacent vertex that is incident to a lower tree edge.

(1) There is an edge d from a subtree rooted at v_2 or v_5 going beyond v_1. We may assume it is from the subtree rooted at v_2; the other case is similar. Add d and remove (v_1, v_2). See Figure 16.1).

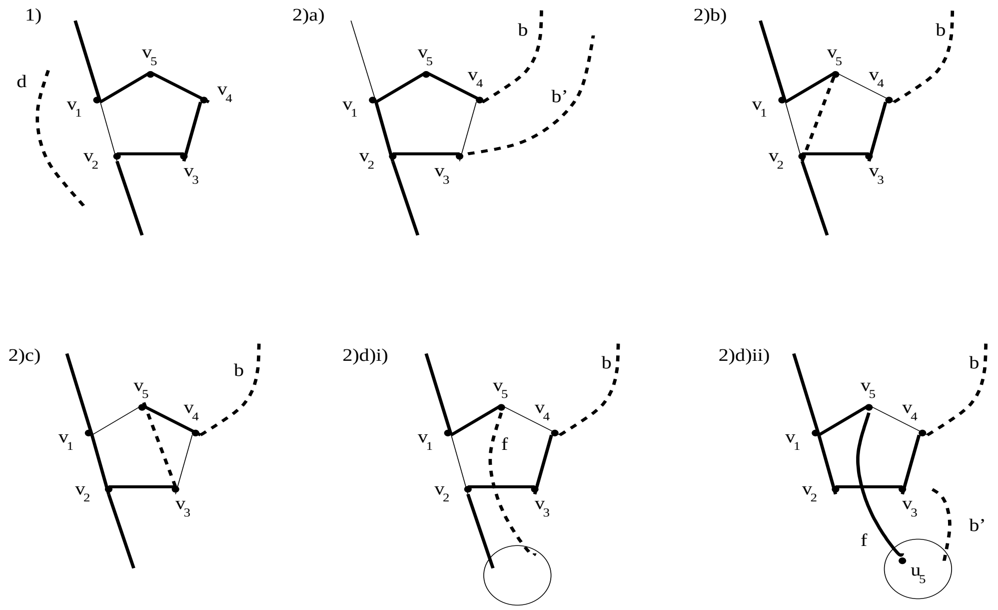

FIGURE 16. Some More 5 cycles.

(2) There is a back edge b incident to v_4 (or, similarly, an edge d emanating from a subtree rooted at v_4).
   (a) There is also a back edge b' from v_3 (or, similarly, an edge d' emanating from a subtree rooted at v_3). Add b and b', remove (v_3, v_4) and the upper tree edge incident to v_1. See Figure 16.2)a). Note that we have removed the upper tree edge; this is not a problem since our goal is to 2-edge connect the 5-cycle to the root component.
   (b) There is an edge (v_2, v_5). Add b and (v_2, v_5). Remove (v_1, v_2) and (v_4, v_5). See Figure 16.2)b).
   (c) There is an edge (v_3, v_5). Add b and (v_3, v_5). Remove (v_1, v_5) and (v_3, v_4). See Figure 16.2)c).
   (d) Since v_5 is not a beta-vertex, there must be a down edge f from v_5.
      (i) If f is incident to a component in a subtree rooted at v_2, add b and f, remove (v_1, v_2) and (v_4, v_5). See Figure 16.2)d)i).
      (ii) So $f = (v_5, u_5)$ is a lower tree edge. Again, by 2-connecting up the top component of T_5, the subtree rooted at v_5, we can assume that there is a back edge b' from T_5 that is not incident to either v_5 or u_5. We add b and b' and remove (v_4, v_5). See Figure 16.2)d)ii). The cost of the additional edges is charged to the components in an identical fashion to that of part (I).

(3) There is a back edge b incident to v_3. Consider the lower tree edge $e' = (v_2, u_2)$. Since v_2 and u_2 are not cut vertices there is an edge b' from the subtree T_2 that is not incident to v_2 or u_2, that is b' is incident to a one vertex in $T_2 - \{u_2\}$ and one of the vertices v_1, v_3, v_4, v_5. It is then easy

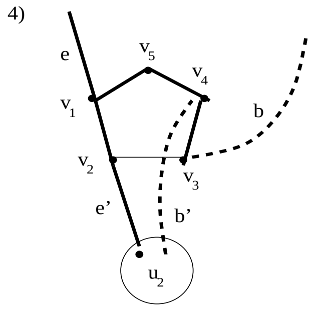

FIGURE 17. Even More 5 cycles.

to 2-edge connect up our two components to the root component. The case where b' is incident to v_4 (the other cases are similar) is shown in Figure 17.4), we use e, b, e', b' and remove (v_1, v_2). Again, the cost of the additional edges is charged to the components in an identical fashion to that of part (I).

(III) The upper tree edge is incident to a vertex which has an opposite vertex that is incident to a lower tree edge, but adjacent vertices are not incident to a lower tree edge.

(1) There is a back edge b from v_2 or v_5 (or their subtrees). We may assume it is from v_5; the other case is similar. Add b and remove the edge (v_1, v_5).

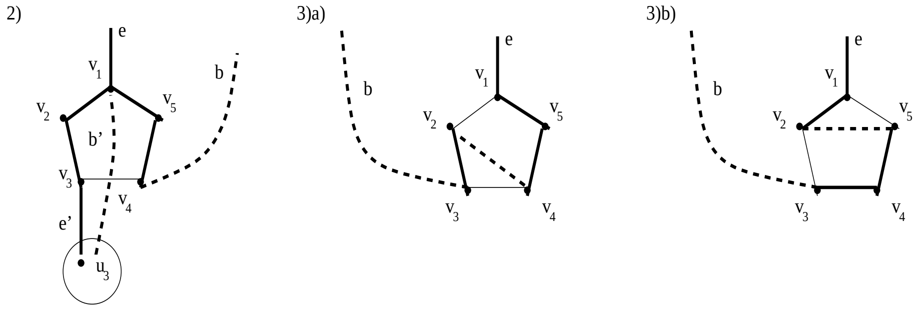

FIGURE 18. Yet more 5 cycles.

(2) There is a back edge b from v_4. Consider the lower tree edge $e' = (v_3, u_3)$. As in case (I), by 2-edge connecting upwards the top component of T_3, the subtree rooted at v_3, we get a back edge b' from T_3 that is not incident to v_3; it follows that b' is incident to v_1 or v_4 (by the DFS priority scheme it cannot be incident to v_2 or v_5). For the case is which b' is incident to v_1 (the other case is similar) we add b and b' and remove (v_3, v_4). This is shown in Figure 18.2). Again, the cost of the additional edges is charged to the components in an identical fashion to that of part (I).

(3) There is a back edge b from v_3. We may assume that there is no lower tree edge incident to v_4, otherwise we are in a symmetric situation to the previous case. Thus, since v_2 is not a beta-vertex there must be an edge (v_2, v_4) or (v_2, v_5). In both cases the charge to the 5-cycle is just one edge, see Figures 18.3)a) and 18.3)b).

Lemma 4.6. *For nodes representing* 2*-edge connected components the charge is at most* 2.

Proof. If we examine such a component then we may add any back edge from the node, in addition to the upper tree edge. This gives a charge of two to the component. □

Notice that if the component contains at least 6 edges then the associated charge is at most one third the components weight. Our main result follows.

Theorem 4.7. *The algorithm is a $\frac{4}{3}$-approximation algorithm for $2EC$.* □

5. Vertex Connectivity

5.1. **The tree of components.** As before, we will build a tree whose nodes are the paths and cycles given by the solution to the maximum cycle-path cover problem. This ensures our initial components are either paths or 2-vertex connected components. Note that we again have two "free" edges associated with each path.

The DFS tree T on the components will use a slightly different priority scheme:

(1) The node represents a cycle: first search for edges incident to vertices adjacent to v in the cycle; next consider the vertices at distance 2 from v along the cycle, etc. Finally consider v itself.
(2) The node represents a path: give priority to one endpoint (the "left" endpoint), with declining priority as we move rightwards. We add one technical condition to reduce the case analysis. Consider two adjacent vertices on the path, v_i and v_{i+1}. If giving v_i priority over v_{i+1} produces a DFS tree in which v_{i+1} has no components rooted at v_{i+1}, whereas giving v_{i+1} priority over v_i produces a DFS tree in which v_{i+1} has a component rooted at v_{i+1} then we give v_{i+1} priority over v_i. Further, this change in priorities will be made only if we have not already switched the priorities of v_{i-1} and v_i.

5.2. **Algorithm and Analysis.** Given the tree it will be simpler to apply a "bottom-up" approach in connecting together our components, rather than the "top-down approach" used in the edge connectivity case. We will begin by examining the leaf components and will require that the following property is maintained at each step.

Property 1. After dealing with component C we require that at least two edges cross from T_C to $G - T_C$ (here T_C is the subtree in T rooted at a component C). In addition these two edges must be vertex disjoint.

The maintenance of Property 1 ensures that our final solution will be 2-vertex connected. To see this note that if we look at our solution in reverse, i.e. top-down, then we are just 2-vertex connecting into the root each component in turn. Moreover, we may assume that the root component is 2-vertex connected. It remains therefore to bound the total number of arcs we add during the algorithm. As before, we charge additional edges to the component we are examining at the time.

Lemma 5.1. *For path nodes the charge is at most $\lceil \frac{k}{3} \rceil + 2$, where k is the number of edges in the path.*

Proof. Our approach is as follows. Starting at the leftmost vertex in the path P we add rightward "edges" (u_i, u_j), where u_i is before u_j on the path. We remark that these "edges" (u_i, u_j) may, in fact, correspond to paths created either by adding one edge to u_j from a component with a back edge incident to u_i, or by adding one edge from u_i to a component that has a back edge incident u_j. We attempt to split the path into blocks of at least three edges with the property that we add only one edge per block.

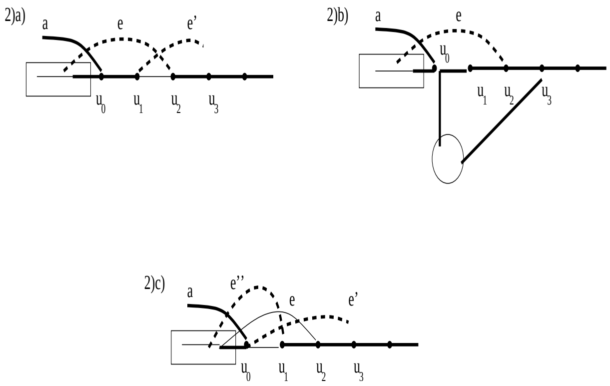

FIGURE 19.

To begin with we apply some preprocessing. Let e^* be the upper tree edge of P and let e^* be incident to u^* on P. For the purpose of analysis we treat $e^* \cup (G - T_P)$ as a single vertex. Next, suppose the last rightward edge added, a, is incident to u_0. We close the previous block at the vertex in the path before u_0. Let u_{-1} be the vertex preceding u_0 on the path. If there is an edge f from u_{-1} to a component rooted at u_0 then we will add f and remove (u_{-1}, u_0) (we may perform such a change as will ensure that whenever we close a block a vertex u we must be using the edge preceding u on the path). We will now show that the next block consists of at least three edges. Take the edge, e, from the previous block that goes furthest rightwards.

(1) If e reaches u_3 then we are done.
(2) Suppose e reaches u_2. We may assume that there is no component C below P whose upper tree edge and back edge are incident to adjacent vertices in $\{u_0, u_1, u_2\}$; if there is then we may save an edge by removing the corresponding edge from P. Now since, u_2 is not a cut vertex we have the following possibilities:
    (a) There is an e' from u_1 going beyond u_2. We get this edge for free by removing (u_1, u_2). See Figure 19.2)a).
    (b) There is a component rooted at u_0 or u_1 whose back edge goes beyond u_2. We simply add e. See Figure 19.2)b).
    (c) Otherwise, there is an e' originating at u_0 going beyond u_2. Since u_1 is not a beta-vertex, it must then be the case that there is an edge e'' from the previous block entering u_1. We add e' and e'' and remove (u_0, u_1). This is illustrated in Figure 192)c).
(3) Suppose e reaches u_1.
    (a) If there is an edge e' from u_0 going beyond u_2 we add e and e' and remove (u_0, u_1). See Figure 20.3)a).
    (b) Otherwise there is an edge e' from u_0 to u_2 and an edge e'' from u_1 going beyond u_2. We add e, e' and e'' and remove (u_0, u_1) and (u_1, u_2). See Figure 20.3)b).

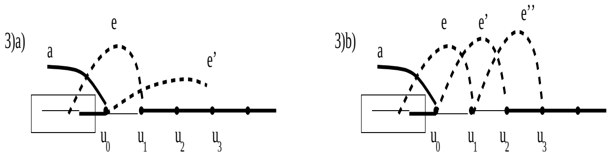

FIGURE 20.

Since the block containing the right endpoint does not contribute an extra edge, we have added an extra $\lceil \frac{k}{3} \rceil$ edges. In order to ensure that Property 1 holds we must use two edges. One of these will be either the upper tree edge e for P or an edge e' that is incident to the same vertex v above P that e is; such a choice will then not affect any arguments regarding the positioning of lower tree edges for components above P. So, in total, we add at most $\lceil \frac{k}{3} \rceil + 2$ edges. Recall, that associated with P we have two free edges, (i.e. $k + 2$ edges in total), so these extra two edges are accounted for. Note that $\lceil \frac{k}{3} \rceil \leq \frac{1}{3}(k + 2)$ so the associated blow-up factor for each path is $\frac{4}{3}$.

We have one caveat. What if u_1, u_2 or u_3 was the contracted vertex $v^* \cup G - T_P$? In this case the arguments with regard to cut vertices may not apply and we must add an extra edge f. However, we may chose f such that it is one of the additional edges required by Property 1; thus, we save one edge in connecting the path upwards. $\square$

Lemma 5.2. *For cycle nodes of size* 3*, the charge is one.*

Proof. We will use the following notation. Call the two edges chosen from the subtree below the 3-cycle a and a'. Let them originate at the vertices z_1 and z_2. We remark that during the course of our algorithm, as we work our way up the graph the upper tree edge e for a component may be removed and replaced by an edge e'. Note that e is also a lower tree edge for some component C. Our arguments regarding the positioning of lower tree edges with respect to C are not affected by the removal of e, however, as will will insist that in such cases e' is incident to the same vertex in C as e was. Thus, since the graph below C is 2-vertex connected when we consider C we may treat e' as a lower tree edge of C. Applying this observation, we may assume that a is the lower tree edge for the 3-cycle.

Let the upper tree edge from the 3-cycle be $e = (v_1, y_1)$. In the course of these proofs we will take y_2 to be a generic vertex in $T - (T_C \cup y_1)$. We will assume that any back edge leaving T_C, that is not incident to y_1, is incident to the same vertex y_2. This is the worst case scenario. The proofs are simpler if this does not happen to be the case. See Figure 21.

(I) Lower tree edges are all adjacent at the same vertex, v_1, as the upper tree edge. Notice that a' is not incident to the 3-cycle. It can not be incident to v_1 as a is. It is not incident to v_2 or v_3 as otherwise it would have been chosen as a lower tree edge not a.

(1) Edge a' is incident to a vertex $y_2 \neq y_1$. Since v_1 is not a cut vertex there is a back edge e' from either v_2 or v_3. These cases are symmetric, the latter is shown in Figure 21.1). We may add e' and remove (v_1, v_3) for a net charge of one to the cycle.

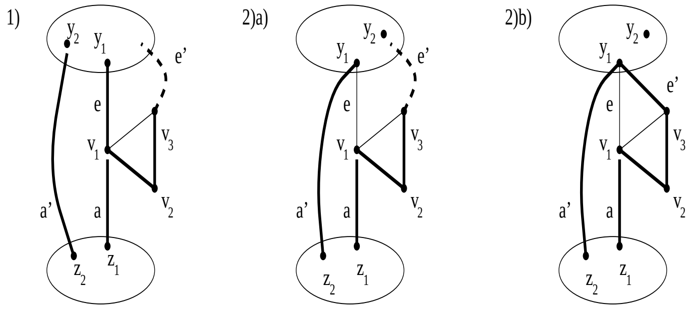

FIGURE 21.

(2) Edge a' is incident to y_1. Again there must be a back edge e' from v_2 or v_3. If e' is incident to y_2, add e' and remove e and (v_1, v_3) for zero net charge to the cycle. See Figure 21.2)a). If e' is incident to y_1, add e' and remove e and (v_1, v_3). We still have one spare edge available. Choose any back edge that is incident to some $y_2 \neq y_1$. See Figure 21.2)b).

(II) The upper tree edge e is incident to v_1 and the lower tree edge a is incident to an adjacent vertex v_2.

(1) If a' is incident to y_2 then we are already done. In fact we may also remove the edge (v_1, v_2). See Figure 22.1)a). If a' is incident to y_1 then remove (v_1, v_2). We then have one extra edge to play with. This we add to connect the 3-cycle upwards to some vertex y_2. This is shown in Figure 22.1)b).

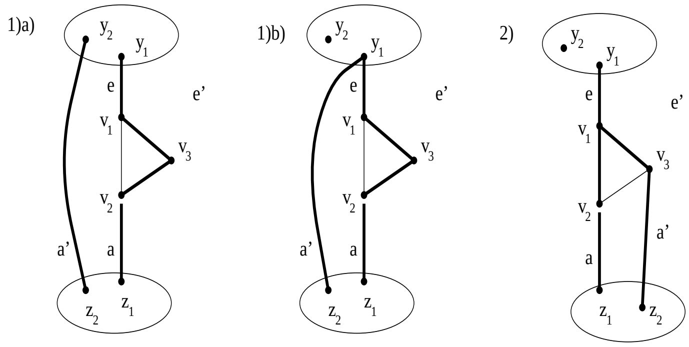

FIGURE 22.

(2) Otherwise a' is incident to v_1 or v_3. The cases are similar and we deal with the latter. We may remove (v_2, v_3) and so have one extra edge available. See Figure 22.2). ☐

Lemma 5.3. *For cycle nodes of size* 4 *the charge is one.*

Proof.

(I) All lower tree edges are incident to v_1.

(1) The back arc a' goes beyond the 4-cycle to $y_2 \neq y_1$. Note that v_1 is not a cut vertex so there is a back arc e' from one of v_2, v_3 or v_4.

(a) The back edge e' is incident to y_1. Add e' and remove e. The case where e' is incident to v_3 is shown in Figure 23.1)a). The charge to the 4-cycle is just one.

(b) The back edge e' goes to y_2 from either v_2 or v_4. Suppose it is from v_4. Add e' and remove (v_1, v_4). See Figure 23.1)b). The other case is similar.

(c) The back edge e' goes from v_3 to y_2. Since v_2 and v_4 are not beta-vertices there must be an edge (v_2, v_4). Add e' and (v_2, v_4). Remove (v_1, v_4) and (v_2, v_3). See Figure 23.1)c).

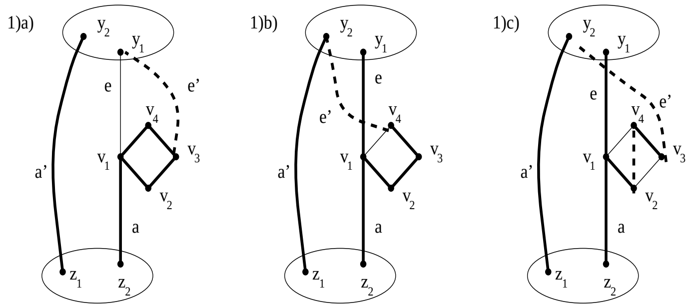

FIGURE 23.

(2) The back arc a' goes beyond the 4-cycle to y_1. This is dealt with in a similar fashion to case 1. There must be a back edge e' from v_2, v_3 or v_4.

(a) The back edge e' is incident to y_2. Add e' and remove e.

(b) The back edge e' is from either v_2 or v_4 to y_1. Then add e' and remove e as well as (v_1, v_2). We have an extra edge to play with. Add any edge from below that is not incident to y_1.

(c) The back edge is from v_3 to y_1. Since v_2 and v_4 are not beta-vertices, there must be an edge (v_2, v_4). Add e', (v_2, v_4) and remove $e, (v_1, v_4)$ and (v_2, v_3). Add any other edge from below not incident to y_1.

(II) There is a lower tree edge a incident to v_3, but none incident to v_2 or v_4.

(1) The back arc a' goes beyond the 4-cycle to $y_2 \neq y_1$. Then we are already 2-vertex connected beyond the 4-cycle. See Figure 24.1).

(2) The back arc a' goes beyond the 4-cycle to y_1.

(a) There is a back edge e' from v_2 or v_4 to y_2. We show the latter case in Figure 24.2)a). Add e' and remove e and (v_3, v_4). This leads to zero net charge to the 4-cycle.

(b) There is a back edge e' from v_2 or v_4 to y_1. We show the latter case in Figure 24.2)b). Add e' and remove e and (v_3, v_4). We still have one edge to play with. We use this edge to connect us up to y_2.

(3) The back edge a' is incident to v_1.

(a) There is a back edge e' from v_2 or v_4 to y_2. Both cases are similar. For the latter case add e' and remove (v_1, v_4). See Figure 24.3)a).

(b) There is a back edge e' from v_2 or v_4 to y_1. For the latter case add e', remove e and (v_1, v_4). We then still have one edge to play with. We pick any suitable back edge that connects us up to y_2. See Figure 24.3)b).

(c) Otherwise since v_2 and v_4 are not beta-vertices there must be an edge (v_2, v_4). We remove (v_1, v_4) and (v_2, v_3), and add (v_2, v_4) and any suitable back edge that connects us up to y_2.

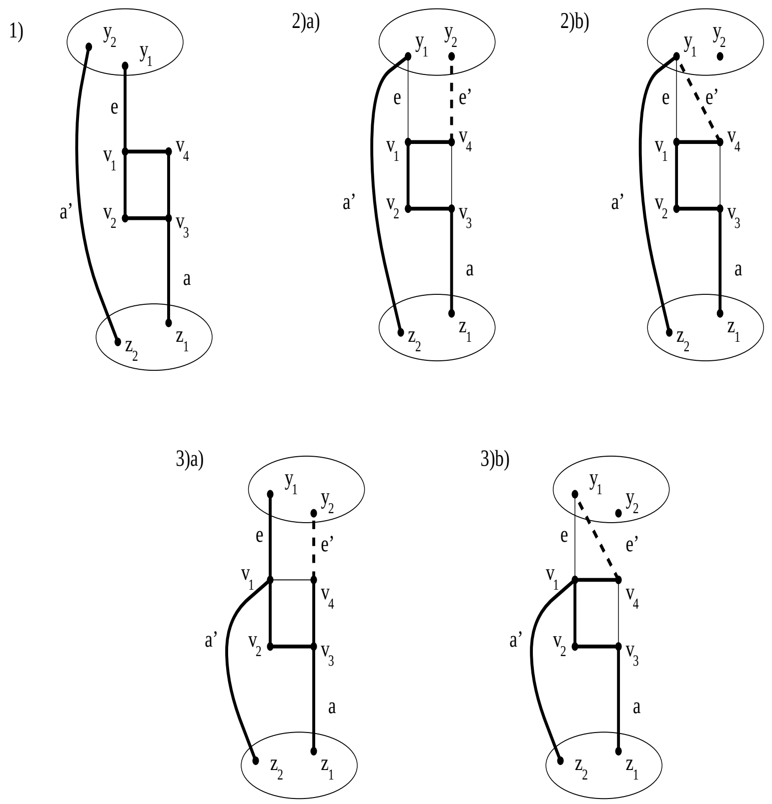

FIGURE 24.

(III) There is a lower tree edge a incident to v_2 or v_4. These cases are symmetric so we will deal with the former case.

(1) The back arc a' goes beyond the 4-cycle to $y_2 \neq y_1$. Then we are already 2-vertex connected beyond the 4-cycle. See Figure 25.1). In fact, we can also remove the edge (v_1, v_2).
(2) The back arc a' goes beyond the 4-cycle to y_1. Remove (v_1, v_2). See Figure 25.2). We may then add any appropriate back edge to connect us up to y_2.
(3) The back arc a' is incident to v_1 or v_3. Here we may remove one edge from the 4-cycle and then choose any appropriate back edge to y_2. The case where a' is incident to v_1 is shown in Figure 25.3).
(4) The back arc a' is incident to v_4. Consider how we may reach y_2.
  (a) Let e' be a back edge into y_2. The cases where e' is from below the 4-cycle or where it originates from v_2, v_3 or v_4 are easily dealt with. See Figures 25.4)a)i)-iv).
  (b) Then let e' be incident to v_1. Suppose there is a back edge b from v_2 to y_1. Then add b and e', remove e and (v_1, v_2). See Figure 25.4)b). The situation in which there is a back edge from v_4 to y_1 is similar.

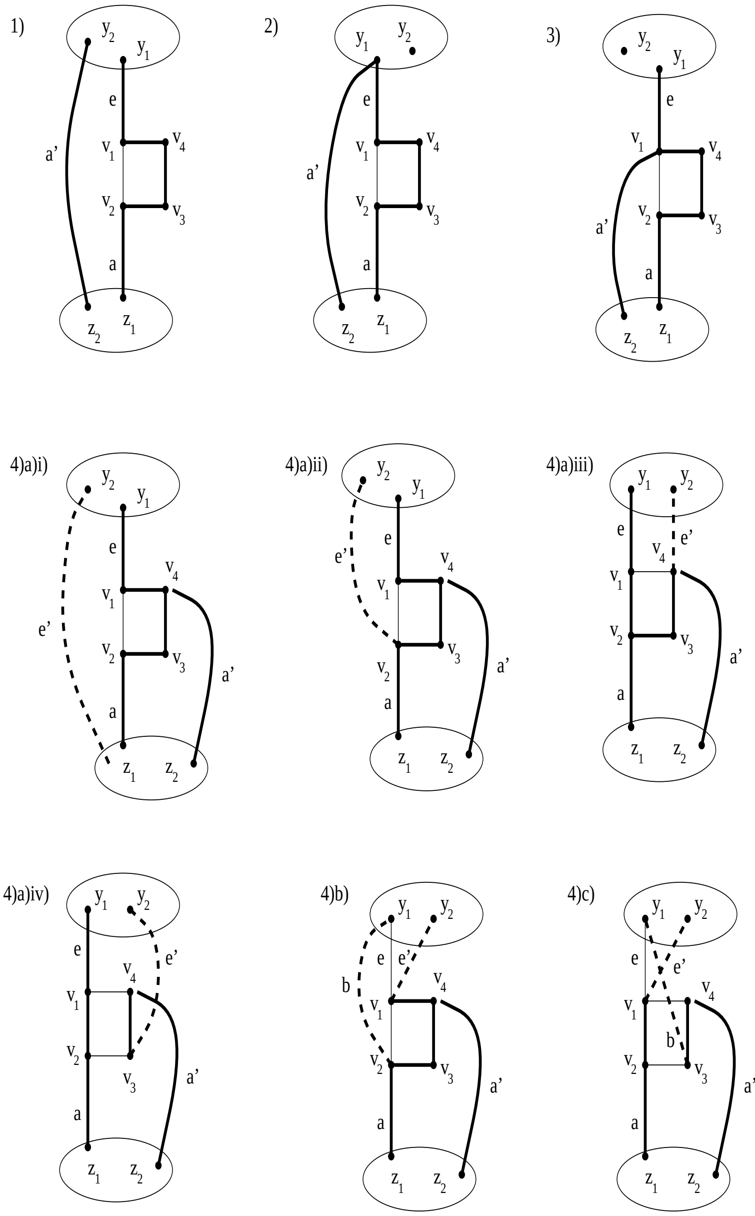

FIGURE 25.

(c) If there is a back edge b from v_3 or the subtree rooted at v_3 to y_1, then add b and e'. Remove e, (v_1, v_4) and (v_2, v_3). See Figure 25.4)c).

(d) Otherwise there must be a back edge b into y_1 originating from below the 4-cycle. Here we add b and e' and remove e and (v_1, v_2). □

We say that an edge (u, v) is *redundant* if there is an optimal solution in G that does not use the edge (u, v). Note that if we can find in polynomial time a redundant arc then we can solve the problem on the simplified graph $G - (u, v)$. To deal with the case of 5-cycles we now need two following lemmas.

Lemma 5.4. *Let G be a beta-free graph that contains a 5-cycle $C = \{v_1, \ldots, v_5\}$. Suppose v_2 has degree 2 and that v_5 has no neighbours outside C. Then G the edge (v_3, v_4) is redundant.*

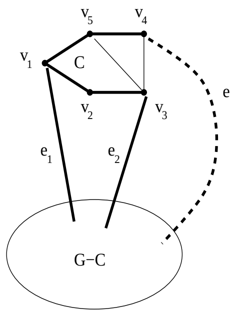

Figure 26.

Proof. Let S be a minimum 2-vertex connected subgraph. Suppose that S contains (v_3, v_4). Since v_2 has degree 2, S also contains the edges (v_1, v_2) and (v_2, v_3). Observe that v_5 has only has neighbours v_1, v_4 and (possibly) v_3. Therefore, there must be an optimal solution S' that also contains all the edges of C. To see this, note that if (v_1, v_5) or (v_4, v_5) is not present in some optimal solution then feasibility holds if we switch that edge for (v_3, v_5). Now S' must contain two disjoint edges, e_1 and e_2, from C to $G - C$. If e_1 and e_2 are incident to v_3 and v_4 then $S' - (v_3, v_4)$ is also 2-vertex connected, a contradiction. Suppose e_1 is incident to v_1 and e_2 is incident to v_3. Since G contains no beta vertices, there must be an edge e from v_4 to $G - C$. Now $S' - (v_3, v_4) \cup e$ is also 2-vertex connected. This is shown in Figure 26. Instead, suppose e_1 is incident to v_1 and e_2 is incident to v_4. If then is no edge between v_3 and v_5 then this case is symmetric to the previous case. Otherwise $S' - (v_3, v_4) \cup (v_3, v_5)$ is also 2-vertex connected and the result follows. $\square$

Lemma 5.5. *Let G be a beta-free graph that contains two vertices v_1 and v_2 whose removal disconnects the graph. Suppose there is a path $\{(v_1, v_4), (v_4, v_3), (v_3, v_2)\}$, where v_3 is a vertex of degree 2. Then (v_1, v_4) is a redundant edge.*

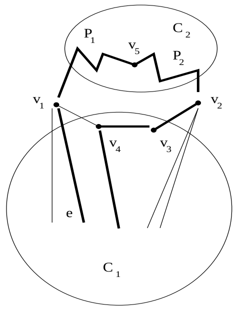

Figure 27.

Proof. Let the path $P = \{(v_1, v_3), (v_4, v_3), (v_3, v_2)\}$ be contained in a the component C_1 induced by the removal of v_1 and v_2. Take a vertex v_5 in a different induced component C_2. Now in any 2-vertex connected subgraph there must be two paths P_1 and P_2 from v_5 to v_1 and v_2, respectively, that are vertex disjoint and contain only vertices in C_2. This situation is illustrated in Figure 27.

We may assume that v_4 is not of degree two otherwise we would have considered the graph formed by the contraction of the edge (v_4, v_3). In addition, there are edges from v_1 to $C_1 - P$, otherwise v_3 is a beta-vertex. Moreover, there are edges from v_2 to $C_1 - P$, otherwise the removal of v_1 and v_4 separates the graph and, hence, (v_1, v_4) is a redundant edge. Now let S be a minimum 2-vertex connected subgraph that contains (v_1, v_4). We have two possibilities. Either $S - (v_1, v_4) \cup e$ 2-vertex connected, where e is an edge from v_1 to $C_1 - P$, or $S - (v_1, v_4) \cup f$ is 2-vertex connected, where f is an edge from v_4 to $C_1 - P$. Thus (v_1, v_4) is a redundant edge. $\square$

Lemma 5.6. *For cycle nodes of size* 5 *the charge is one.*

Proof.

(I) All lower tree edges are incident to v_1. Note that v_1 is not a cut vertex.

(1) The back edge a' is incident to y_2.
  (a) There is a back edge e' into y_1 from v_2, v_3, v_4 or v_5. We add e' and drop e. An example is shown in Figure 28.1)a).
  (b) There is a back edge e' into y_2 from v_2 or v_5. We illustrate the latter case in Figure 28.1)b). Add e' and remove (v_1, v_5).
  (c) There is a back edge e' into y_2 from v_3. The case from v_4 is symmetric. The situation in which there is an edge (v_2, v_5) is shown in Figure 28.1)c). The situation in which there is an edge (v_2, v_4) is similar. Otherwise v_2 is of degree 2 and v_5 has only neighbours within the 5-cycle. Hence we may apply Lemma 5.4 and consider the graph $G - (v_3, v_4)$.

(2) The back edge a' is incident to y_1. This case is dealt with in a similar manner to the previous case.

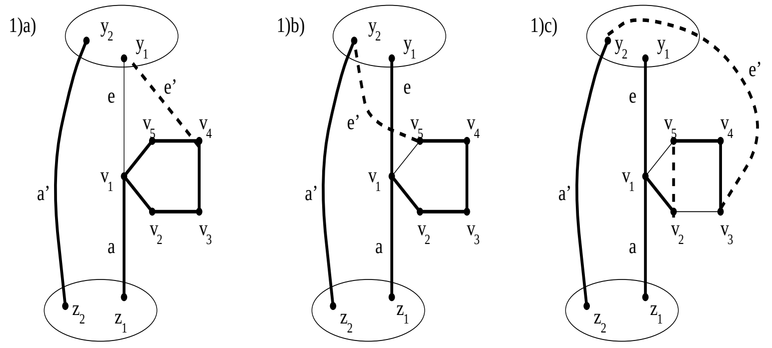

FIGURE 28.

(II) There is a lower tree edge a incident to v_3 or v_4, but none incident to v_2 or v_5. These cases are symmetric so we will deal with the former case.

(1) The back edge a' is incident to y_2. Then we are already done. See Figure 29a).
(2) The back edge a' is incident to y_1.
  (a) There is a back edge e' into y_2 from any of the vertices in the 5-cycle except v_3. Add e' and remove e. An example is given in Figure 29.2)a).

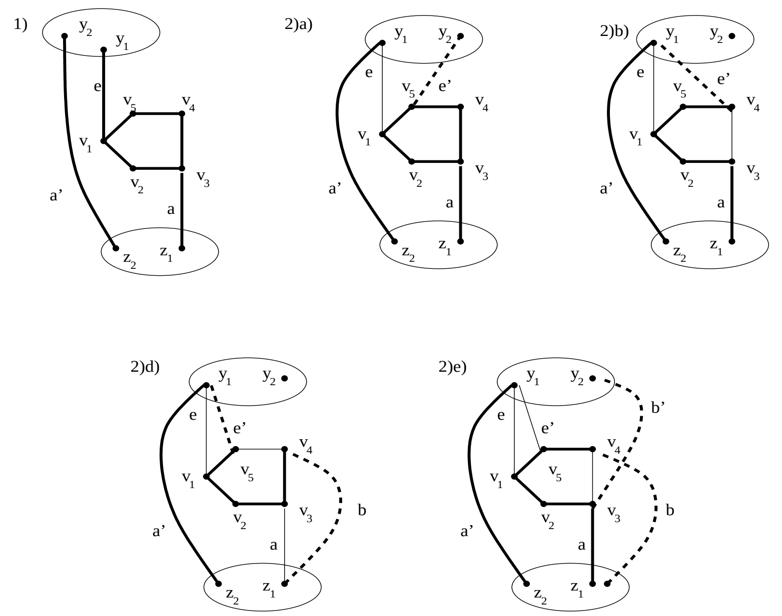

FIGURE 29.

(b) There is a back edge into y_1 from either v_2 or v_4. Both cases are similar. The latter is shown in Figure 29.2)b). Add e' and remove e and (v_3, v_4). We then add an appropriate edge incident to y_2.

(c) The situations in which there are edges (v_2, v_5) or (v_2, v_4) are easy to deal with.

(d) So v_2 has degree 2. It follows that there must be an edge $e' = (v_5, y_1)$, otherwise we apply Lemma 5.4 and consider $G - (v_3, v_4)$. In addition there must be a back edge b into v_4, otherwise we may apply Lemma 5.4 and consider $G - (v_1, v_5)$. If b is incident to z_1 then we may add b and e' and remove a and (v_4, v_5) as shown in Figure 29.2)d). Hence we have one free edge left with which to connect us up to y_2.

(e) Finally consider the situation in which b is not incident to z_1. The back edge incident to y_2 must come from v_3 or below the 5-cycle. Both cases are similar. The former case is shown in Figure 29.2)e). Add b and b'. Remove e and (v_3, v_4).

(3) The back edge a' is incident to v_4. This case follows from the observation that we may remove the edge (v_3, v_4).

(4) The back edge a' is incident to v_1.

(a) There is a back edge e' to y_2 from either v_2 or v_5. We show the later case in Figure 29.4)a).

(b) There is a back edge e' to y_1 from v_2. Add e', remove e and (v_2, v_3). Then add an appropriate back edge to y_2. See Figure 29.4)b).

(c) The situations in which there are edges (v_2, v_5) or (v_2, v_4) are easy to deal with.

(d) If there is no back edge e' to y_1 from v_5, then we apply Lemma 5.4 and consider $G - (v_3, v_4)$.

(e) There is a back edge e' to y_1 from v_5.

(i) There is a back edge b from v_4 to y_2 then we proceed as in Figure 30.4)e)i).

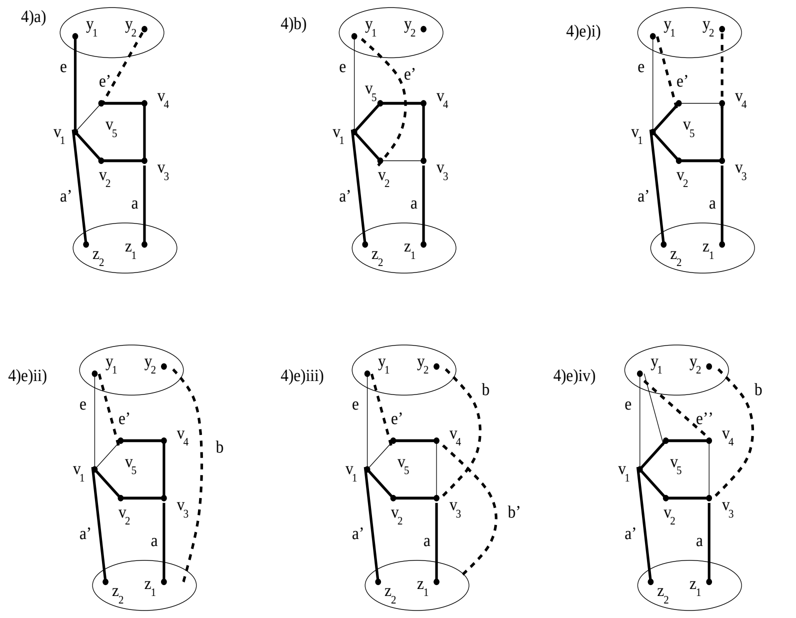

FIGURE 30.

(ii) There is a back edge b to y_2 from v_1, v_4 or from below the 5-cycle. These cases are similar. We illustrate the last of these cases in Figure 30.4)e)ii).

(iii) It must then be the case that the only back edge b into y_2 comes from v_3. If there is an edge b' into v_4 from below the 5-cycle then we add b, b' and e' and remove e, (v_1, v_5) and (v_3, v_4). See Figure 30.4)e)iii).

(iv) If there is an edge e'' from v_4 to y_1 then we add e'' and b and remove e and (v_3, v_4). See Figure 30.4)e)iv).

(v) Otherwise we apply Lemma 5.4 and consider $G - (v_1, v_5)$.

(III) There is a lower tree edge a incident to v_2 or v_5. These cases are symmetric so we will deal with the former case.

(1) The back edge a' is incident to y_2. Then we are already done. See Figure 31.1). We may also remove the edge (v_1, v_2).

(2) The back edge a' is incident to y_1. Remove the edge (v_1, v_2). We have one spare edge which we use to connect up to y_2. See Figure 31.2).

(3) The back edge a' is incident to v_1 or v_3. In either case we may remove one edge from the cycle which gives us a spare edge to play with.

(4) The back edge a' is incident to v_5. By consideration of the back edges incident to y_2 we get to remove either (v_1, v_2) or (v_1, v_5).

(5) The back edge a' is incident to v_4.

(a) The cases in which there is a back edge e' to y_2 from v_2, v_3 or v_5 are all easy to deal with.

(b) If there is a back edge e' from below the 5-cycle to y_2 we then may add e' and remove (v_1, v_2).

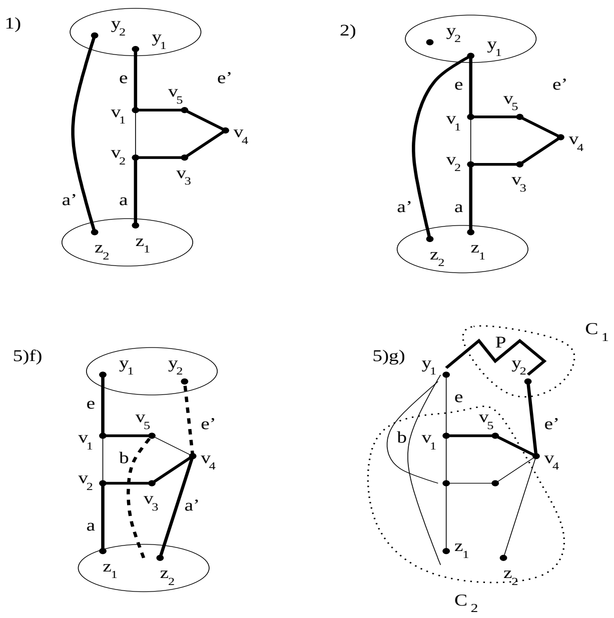

FIGURE 31.

(c) Suppose there is an edge $e' = (v_1, y_2)$ along with a back edge into y_1 from any of v_2, v_3 or v_5. These are dealt with in a similar manner to those in case 5a). Similarly if there is a back edge into y_1 from below the 5-cycle.

(d) Hence the only vertex incident to y_2 is v_4. Again the situations where there is a back edge into y_1 from v_3 or v_5 are easy to deal with.

(e) We use the usual methods to deal with the edges $(v_1, v_3), (v_3, v_5)$ or (v_2, v_5), if they are present.

(f) Suppose there is a back edge b into v_5. Then add b and e', remove (v_1, v_2) and (v_4, v_5). This is shown in Figure 31.5)f)).

(g) Since v_5 is not a beta-vertex, there must be a back edge b into y_1 from v_2 or below the 5-cycle. Observe also that the removal of y_1 and v_4 disconnects the graph into two components C_1 and C_2. See 31.5)g). In addition, note that v_5 is of degree 2. It follows, by 5.5, that (y_1, v_1) is a redundant edge and, hence, we may solve on $G - (y_1, v_1)$.

Theorem 5.7. *The algorithm is a $\frac{4}{3}$-approximation algorithm for $2VC$.* □

6. Comparison with the subtour relaxation

Consider the following linear programming relaxation for $2EC$.

$$\min \sum_{e \in E} x_e$$

(1) $$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V,\ S \neq \emptyset$$

(2) $$x_e \geq 0 \qquad \forall e \in E$$

It may be noted that the LP relaxation is similar to the familiar *subtour relaxation* for the Travelling Salesman Problem (TSP). The subtour relaxation imposes the additional constraints that $\sum_{e\in\delta(\{v\})} x_e = 2,\ \forall v \in V$. For metric costs, though, there is an optimal solution to the LP relaxation which is also optimal for the subtour relaxation. It has been conjectured that for a graph with metric costs, OPT(TSP) $\leq \frac{4}{3}$ OPT(LP). One implication of this conjecture is that for a 2-edge connected graph, OPT($2EC$) $\leq \frac{4}{3}$ OPT(LP). Carr and Ravi [2] provide a proof of the implication for the special case of half-integral solutions of the relaxation. Unfortunately, the implication (in its entirety) does not follow simply from our result. This is due to the fact that our lower bound, based upon integral degree 2 subgraphs, may be stronger in some situations than the bound offered by the linear program. The graph in Figure 32a) is an example. Figure 32b) shows the half integral optimal solution given by the LP. Here, edges given weight 1 are solid whilst edges of weight $\frac{1}{2}$ are dashed. This has total value 9, whereas the minimal integral degree 2 subgraph, Figure 32c), has value 10. A weaker implication of the TSP conjecture is that $2EC$ is approximable to within $\frac{4}{3}$, and our result does indeed verify this implication.

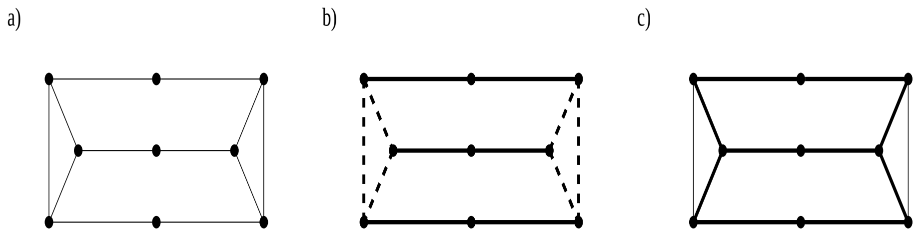

FIGURE 32. Subtour vs integral degree 2 subgraph.

7. CONCLUSION

How far can we go with the approach described here? For $2EC$, there are two bottlenecks: (i) paths and (ii) cycles of short length. When the graph has no beta-structures, but has a Hamiltonian path, there is an example where the minimum 2-edge connected subgraph is of size $\frac{5}{4}n$. One idea to deal with small cycles is as follows. It is easy to see that in graph with no cut vertices there is an optimal solution to $2EC$ that contains no 3-cycles (*triangles*). Therefore, we may use a triangle-free solution to $D2$ as a lower bound; such a solution may be obtained in polynomial time [7]. It is not possible to eliminate 5-cycles since finding the minimum size $D2$ solution whose smallest cycle has length at least six is NP-hard. It is not clear whether the idea could be used to eliminate 4-cycles (*squares*). The complexity of finding a minimum sized $D2$ solution that does not contain triangles or squares (4-cycles) is unresolved. In addition, it may not be the case that the optimal solution in a graph containing no cut vertices or beta-vertices is square-free; see the example in Figure 33.

Acknowledgment. The authors would like to thank Mette Eskesen for comments and suggestions. We are also grateful to an anonymous referee for pointing out an error in a previous version of this paper, and for many comments and suggestions that greatly improved the paper.

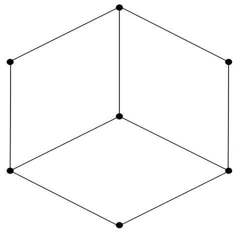

FIGURE 33.

REFERENCES

[1] A. Czumaj and A. Lingas, "On approximability of the minimum-cost k-connected spanning subgraph problem", *Proc* 10^{th} *Annual ACM-SIAM SODA*, pp281-290, 1999.

[2] R. Carr and R. Ravi, A new bound for the 2-edge connected subgraph problem, *Proceedings of the* 6*th Annual International IPCO*, 1998.

[3] J. Cheriyan, A. Sebö and Z. Sigeti, "Improving on the 1.5-approximation of a smallest 2-edge connected spanning subgraph", *SIAM Journal of Discrete Math.*, **14(2)**, pp170-180, 2001.

[4] C. Fernandes, "A better approximation ratio for the minimum k-edge-connected spanning subgraph problem", *Proc* 8^{th} *Annual ACM-SIAM SODA*, pp629-638, 1997.

[5] N. Garg, A. Singla, S. Vempala, "Improved approximations for biconnected subgraphs via better lower bounding techniques", *Proc* 6^{th} *Annual SODA*, 1993.

[6] M. Grötschel, L. Lovász and A. Schrijver, *Geometric algorithms and combinatorial optimization*, Springer-Verlag, New York, 1988.

[7] D. Hartvigsen, *Extensions of Matching Theory*, Ph.D. thesis, GSIA, Carnegie Mellon University, 1984.

[8] S. Khuller and U. Vishkin, "Biconnectivity approximations and graph carvings", *Journal of the ACM*, **41**, pp214-235 (1994).