# The SONET Edge-Partition Problem

by
Olivier Goldschmidt
OPNET Technologies, Inc.
2006 Delaware Street Berkeley, CA 94709-2122 ogoldschmidt@opnet.com

Dorit S. Hochbaum
Department of Industrial Engineering and Operations Research
4135 Etcheverry Hall University of California, Berkeley Berkeley, CA 94720-1777
hochbaum@ieor.berkeley.edu

Asaf Levin
Department of Statistics and Operations Research
Tel-Aviv University, Tel-Aviv 69978, Israel.
levinas@post.tau.ac.il

Eli V. Olinick
Department of Engineering Management, Information, and Systems
Southern Methodist University
PO Box 7502123
Dallas, TX 75275-0123
olinick@engr.smu.edu

**Abstract**

Motivated by a problem arising in the design of telecommunications networks using the SONET standard, we consider the problem of covering all edges of a graph using subgraphs that contain at most $k$ edges with the objective of minimizing the total number of vertices in the subgraphs. We show that the problem is $\mathcal{NP}$-hard when $k \geq 3$ and present a linear-time $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation algorithm. For even $k$ values we present an approximation scheme with a reduced ratio but with increased complexity.

**keywords:** combinatorial optimization, graph partitioning, approximation algorithms, survivable network design, equipment placement in SONET rings, telecommunications networks

**Acknowledgments**

The authors would like to thank the anonymous reviewers whose many thoughtful comments and suggestions greatly the improved this paper. This work was partially supported by the Office of Naval Research under award number N00014-96-1-0315.

# 1 Introduction

The edge-partitioning problem presented in this paper is motivated by a problem in the design of SONET telecommunication networks. In a SONET ring, customer sites are connected via a ring of fiber-optic cable, each one sending, receiving, and relaying messages through a device called an add-drop-multiplexer (ADM). A bandwidth request, or demand, is given for every pair of sites. Each SONET ring serves a collection of demand pairs. The capacity of a ring must accommodate the sum of the bandwidth requests of all the demand pairs it serves. The problem is to partition all customer demand pairs into subsets so that the traffic in each subset is bounded by a given capacity limit. A site may be assigned to more than one ring, but the traffic between two sites cannot be split between rings. It is assumed that there are no digital cross connects (DCS) to transfer the traffic from one ring to another. A site requires an ADM for each ring to which it is assigned. The objective is to minimize the total number of the ADMs.

The problem can be formulated as an optimization problem defined on an edge-weighted graph. The vertices of the graph correspond to customer sites while the weight of an edge between two vertices corresponds to the bandwidth request for the pair of sites linked by the edge. The problem is then to cover the edges of the graph with a set of edge-disjoint subgraphs such that the total weight of the edges covered by each subgraph is at most a given ring capacity. Since the subgraphs have no edges in common, we say that they *partition* the edges of the graph. The ring capacity is fixed and determined by the size of the ADMs being used. ADMs come in an industry-standard set of sizes, for example 155 or 622 megabits per second, and all rings must use the same size ADM. The objective is to minimize the total number of vertices used

in all subgraphs, which corresponds to the total number of ADMs.

Exact methods and heuristics for this problem are presented by Lee, Sherali, Han and Kim [LSHK00] and by Sutter, Vanderbeck and Wolsey [SVW98]. In this paper, we focus on the "uniform" version of the problem where all bandwidth pair requests are equal and the ring capacity is $k$ times this common demand unit. The study of the uniform case can provide important insights for the more general weighted problem. Our results can be immediately extended to a variant of the weighted case where demand splitting is allowed by formulating the problem on a multi-graph where $d$ units of demand between sites $u$ and $v$ are represented by $d$ edges between vertices $u$ and $v$. Note, however, that this transformation is polynomial only if $d$ is bounded by a polynomial in the number of links in the network.

This paper is organized as follows. In Section 2, we define the uniform version of the edge-partitioning problem formally as the *$k$-Edge-Partitioning Problem* ($k$-EP) and show that it is $\mathcal{NP}$-hard for $k \geq 3$. In Section 3, we present a linear-time $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation algorithm called the *$k$-cover algorithm*. We present a post-processing routine and approximation scheme for even values of $k$ in Section 4. A summary of results for different values of $k$ appears in the following table:

| $k$ | $k$-cover | improved ratio for even $k$ |
|---|---|---|
| 3 | $\frac{3}{2}$ | — |
| 4 | $\frac{3}{2}$ | $\frac{4}{3}+\epsilon$ |
| 5 | $\frac{5}{3}$ | — |
| 6 | 2 | $\frac{15}{8}+\epsilon$ |
| 7 | $\frac{7}{4}$ | — |
| 8 | 2 | $\frac{48}{25}+\epsilon$ |
| $k$ | $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$ | $\frac{k(1+\frac{2}{k+2})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}+\epsilon$ |

Table 1: Approximation ratios for $k$-EP as a function of $k$.

## 2 The $k$-Edge-Partitioning Problem ($k$-EP) and Bounds on the Optimal Solution

Throughout this paper, $G=(V,E)$ is a simple (i.e., without loops or multiple edges) undirected graph with vertex set $V$, $|V|=n$, and edge set $E$, $|E|=m$. We use the term *size* to refer to the number of edges in a graph. The notation $(i,j)$ refers to the undirected edge between vertices $i$ and $j$. A simple path with $\ell$ edges is called an $\ell$-path and a set of three edges sharing a common vertex is called a 3-star.

Given a graph $G=(V,E)$ and an integer $k<|E|$, define the $k$-Edge-Partitioning Problem ($k$-EP) as that of partitioning $E$ into a collection of sets $\mathcal{R}=\{R_1,R_2,\ldots\}$ such that $|R_i|\leq k$ for all $R_i\in\mathcal{R}$ and $\sum_{R_i\in\mathcal{R}}|V_i|$ is minimized, where $V_i$ is the set of vertices in the subgraph induced by the edge set $R_i$.

We say that an edge $e\in E$ is *covered* by $R_i\in\mathcal{R}$ if $e\in R_i$. Each set of edges $R_i$ induces a

subgraph of $G$. Suppose that edge $e$ is covered by set $R_i$. The *density* or *charge* for covering $e$ is the number of vertices in the subgraph induced by $R_i$ divided by $|R_i|$. Note that the sum of the edge densities over all edges in the graph is equal to the number of vertices used by $\mathcal{R}$ to cover $E$. Thus, $k$-EP may also be stated as a problem of covering $E$ with subgraphs of size $k$ or smaller so as to minimize the total, or equivalently average, charge per edge. To compute the approximation ratio for a $k$-EP solution $\mathcal{R}$, we divide the upper bound (the average charge per edge in $\mathcal{R}$), by the lower bound established in the following lemma:

**Lemma 2.1** *The average charge per edge in any $k$-EP solution is at least*

$$\frac{1}{k}\lceil\frac{1+\sqrt{8k+1}}{2}\rceil.$$

**Proof**

We need at least $p$ vertices to cover any set of $k$ edges where $p$ is the smallest integer such that $k \leq \binom{p}{2}$. In any solution, the minimum number of vertices is at least $\frac{mp}{k}$. This minimum is obtained if the edges of the graph can be partitioned into sets of $k$ edges such that each set induces a complete graph with $p$ vertices. The minimum density for any edge is $\frac{p}{k}$. Note that $k \leq \binom{p}{2} = \frac{p^2}{2} - \frac{p}{2}$ which implies the following relationship between $p$ and $k$:

$$p^2 - p \geq 2k \Rightarrow 4p^2 - 4p + 1 \geq 8k + 1 \Rightarrow (2p-1)^2 \geq 8k+1 \Rightarrow p \geq \frac{1+\sqrt{8k+1}}{2}$$

As $p$ is a positive integer, this implies that $p \geq \lceil\frac{1+\sqrt{8k+1}}{2}\rceil$. Therefore, the charge per edge is at least $\frac{p}{k} \geq \frac{1}{k}\lceil\frac{1+\sqrt{8k+1}}{2}\rceil$. ■

Consider two extreme cases in the range of objective function values for the $k$-EP problem. In the "worst" case, each subgraph is a collection of $m$ two-vertex subgraphs (i.e., each edge

is covered by itself). Such a solution uses $2m$ vertices to cover $m$ edges. In the "best" case, all of the edges adjacent to any given vertex are covered by the same subgraph so that each vertex appears in exactly one covering subgraph and the objective function value is $n$. This can only happen if the degree of each vertex is at most $k$ and each connected component has $k$ or fewer edges. For example, suppose that $G$ is a complete graph on four vertices. When $k = 6$, the optimal $k$-EP solution is $G$ itself and the objective function value is 4. The worst possible solution consists of six two-vertex subgraphs, one for each edge, giving an objective function value of 12. In general, the objective function value must be at least $n$ and at most $2m$. Any solution will thus be at most $\frac{2m}{n}$ times the optimum, and the ratio gets worse for denser graphs. For the remainder of this paper, we assume that the given graph is connected and has at least three edges. Given a graph that is not connected, we can apply our algorithms to its connected components.

## 2.1 Complexity of $k$-EP

When $k = 1$, the trivial solution to $k$-EP is $E$ itself with $2m$ vertices used, two for each edge. When $k = 2$, the problem can be solved in $O(m)$ time using an algorithm due to Masuyama and Ibaraki [MI91]. For 3-EP, we are limited to using the following basic subgraph types:

1. Triangles. These subgraphs are the most desirable to use in a cover since they use on average only one vertex per edge (the charge of covering an edge by a triangle is one).

2. Subgraphs that are 3-*stars* or 3-paths using four vertices to cover three edges for a charge of $\frac{4}{3}$ vertices per edge.

3. 2-paths that use three vertices to cover two edges for a charge of $\frac{3}{2}$ vertices per edge.

4. Single edges that use two vertices per edge.

If we use triangles only to cover the edges, then we can meet the lower bound of $m$ on the number of vertices in the covering subgraphs. However, the question of deciding whether or not we can do this is $\mathcal{NP}$-complete [Hol81]. Thus, the problem of covering the edges of a graph with a minimum number of triangles is $\mathcal{NP}$-hard. Given any fixed three-edged tree $H_i$ (a 3-path or a 3-star), the problem of partitioning the edges of $G$ into subgraphs isomorphic to $H_i$ is $\mathcal{NP}$-complete [DT97]. As we will now prove, $k$-EP is $\mathcal{NP}$-hard for $k \geq 3$.

The decision version of $k$-EP is stated as follows:

**$k$-Edge-Partitioning Problem ($k$-EP)**
**Instance**: An undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$, and integers $k$ and $L$.
**Question**: Is there a partition of $E$ into sets $\mathcal{R} = \{R_1, R_2, \ldots\}$ such that the following conditions are met?

1. $|R_i| \leq k$ for all $R_i \in \mathcal{R}$ and
2. $\sum_{R_i \in \mathcal{R}} |V_i| \leq L$ where $G_i = (V_i, R_i)$ is the subgraph induced by $R_i$.

We show that the *Edge-Partition into Triangles* Problem (EPT), which is known to be $\mathcal{NP}$-complete [Hol81], can be reduced to $k$-EP in polynomial time. EPT is stated as follows.

**Edge-Partition into Triangles** (EPT)
**Instance**: An undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$.
**Question**: Is there a partition of $E$ into sets $\{E_1, E_2, \ldots, E_{\frac{m}{3}}\}$ such that each $E_i$ induces a subgraph of $G$ isomorphic to a complete three-vertex graph (i.e. a triangle)?

**Theorem 2.1** *The $k$-Edge-Partitioning Problem is $\mathcal{NP}$-complete.*

**Proof**

The problem is clearly in $\mathcal{NP}$, given a partition $\mathcal{R}$, the number of edges and vertices in each

$R_i \in \mathcal{R}$ can be checked in linear time. Given an instance of the EPT, construct an instance of $k$-EP on the same graph with $k = 3$ and $L = m$. We now show that this $k$-EP instance is a "yes" instance if and only if the original EPT is also a "yes" instance.

Consider a "yes" instance of the EPT. Let $R_i = E_i$ for $i = 1, \ldots, \frac{m}{3}$. Since each set $E_i$ is a triangle, $|R_i| = 3$ for $i = 1, \ldots, \frac{m}{3}$. Each set in $\mathcal{R}$ contains three vertices and there are exactly $\frac{m}{3}$ sets, so $\sum_{R_i \in \mathcal{R}} |V_i| = m$. Thus, a "yes" instance of EPT yields a "yes" instance of $k$-EP.

Suppose that the answer to the $k$-EP instance derived from the original EPT instance is "yes". Consider a partition $\mathcal{R}$ that satisfies conditions 1 and 2 above. Let $x$ and $y$ be the number of sets in $\mathcal{R}$ that consist of exactly one edge and exactly two edges, respectively. We now show that $x = y = 0$ which implies that $|R_i| = 3$ for all $R_i \in \mathcal{R}$. It follows that all the associated $G_i$ are triangles.

Since there are $x$ sets each containing a single edge, $\mathcal{R}$ requires at least $\frac{(m-x)}{3}$ sets to cover the remaining edges. This means that $\sum_{R_i \in \mathcal{R}} |V_i| \geq 2x + \frac{3(m-x)}{3} = m + x$. Thus, $x = 0$. Now, if there are $y$ sets each containing exactly two edges, then the remaining $m - 2y$ edges are covered by triangles and thus require $\frac{m-2y}{3}$ sets to be covered by $\mathcal{R}$. This means that $\sum_{R_i \in \mathcal{R}} |V_i| = 3y + m - 2y = m + y$. Thus, $y = 0$.

Since $|R_i| = 3$ for all $R_i \in \mathcal{R}$, it follows that all the associated $G_i$ are triangles. Thus, the answer to the EPT instance must also be "yes". Therefore the EPT instance has a "yes" answer if and only if the associated $k$-EP instance has a "yes" answer. ■

# 3 The $k$-Cover Algorithm

In this section, we present a linear-time $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation algorithm for $k$-EP. As we will show, the average charge per edge in the cover delivered by the algorithm is at most $1 + \frac{1}{\lceil \frac{k}{2} \rceil}$. Recall from Lemma 2.1, that the minimum possible charge per edge is $\frac{1}{k}\lceil \frac{1+\sqrt{8k+1}}{2} \rceil$. Thus, the approximation ratio for our algorithm is $1 + \frac{1}{\lceil \frac{k}{2} \rceil} \div \frac{1}{k}\lceil \frac{1+\sqrt{8k+1}}{2} \rceil = \frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$.

This section is organized as follows. In Subsection 3.1, we present a linear-time algorithm called *rooted_tree_cover* that covers a rooted tree by partitioning its edges into trees each of which have between $\lceil k/2 \rceil$ and $k$ edges. Since a tree with $\lceil \frac{k}{2} \rceil$ edges has $\lceil \frac{k}{2} \rceil + 1$ vertices, the charge per edge for the resulting $k$-EP solution is $(\lceil \frac{k}{2} \rceil + 1) \div \lceil \frac{k}{2} \rceil = 1 + \frac{1}{\lceil \frac{k}{2} \rceil}$. In Subsection 3.2, we present the *$k$-cover algorithm* for covering any connected graph. This algorithm finds cycles in the graph and "opens" them. To open a cycle $C$ in a graph $G = (V, E)$, we take an arbitrary edge $(u, v) \in C$, add a new node $v'$ to $V$ and replace $(u, v)$ with a new edge $(u, v')$. This process is repeated until $G$ becomes a tree $G'$. We then apply rooted_tree_cover to $G'$. The charge per edge in $G'$ is at most $1 + \frac{1}{\lceil \frac{k}{2} \rceil}$. Since the actual charge per edge in $G$ can only be smaller, the $k$-cover algorithm is a $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation. The details of how this algorithm is implemented and a proof of its correctness are presented in Subsection 3.2.

## 3.1 Covering Trees

We now present an algorithm called *rooted_tree_cover* used to cover the edges of a rooted tree $T = (V, E)$. We use the following notation to describe the algorithm. A subgraph of a given graph $G = (V, E)$ that is a tree with a designated root vertex is called a *rooted tree*. Given a rooted tree $T = (V, E)$ with root vertex $r$, define a *leaf* as any vertex of degree one which is not

the root. Given a vertex $v$ that is not the root, the *parent* of $v$ is the first vertex on the unique path from $v$ to the root. A vertex whose parent is $v$ is said to be a *child* of $v$. We denote the children of vertex $v$ by $ch(v)$. A *descendant* of $v$ is any vertex $w$ such that $v$ is on the unique path from $w$ to the root. A *subtree* is a tree induced by a vertex $v$, the root of the subtree, and all its descendants. The set of edges of the subtree of $T$ rooted at vertex $i$ is denoted by $T_i$. A *child-tree* of $v$ is a partial subtree formed by $v$ with the subtree rooted at one of its children together with the edge connecting $v$ to that child. Thus, if $j \in ch(i)$, then the child-tree of $i$ *associated* with vertex $j$ is the tree formed by adding the edge $(i, j)$ to $T_j$. $|T|$ denotes the number of edges in tree $T$. $T \setminus T_i$ denotes the tree obtained by removing the subtree rooted at $i$, but not vertex $i$ itself, from $T$.

We now give a brief sketch of the algorithm before presenting it formally. We assume that the vertices of the tree passed to *rooted_tree_cover* have been assigned labels or indices according to a *postorder* listing of $V$ in which the children of the vertex with index $i$ all have indices in the set $\{1, 2, \ldots, i-1\}$. In this order the index of the root is $n = |T| + 1$. A postorder listing of the vertices of a tree $T$ can be obtained in $O(|T|)$-time [AHU74].

The algorithm visits the vertices in order according to the postorder listing. It starts at vertex 1 and continues until it reaches the first vertex $i$ such that $|T_i| \geq \lceil k/2 \rceil$. If $|T_i| \leq k$, then $T_i$ becomes one of the subgraphs used to cover the graph, the edges of $T_i$ are removed from the tree and the algorithm proceeds to vertex $i + 1$.

If $|T_i| > k$, then the algorithm finds a subtree $T_i'$ of $T_i$ such that $\lceil k/2 \rceil \leq |T_i'| \leq k$. $T_i'$ becomes one of the subgraphs used to cover the graph and its edges are removed from the tree. This process is repeated until $|T_i| < \lceil k/2 \rceil$ at which point the algorithm proceeds to vertex

$i+1$. An example of the algorithm is given in Appendix A. We now give a formal description of the algorithm and establish its complexity and correctness.

**Procedure rooted_tree_cover** $(T, r, k)$
**Input:** A tree $T$ rooted at $r$ and an integer $k$.
**Output:** A $k$-EP solution $\mathcal{C}$ for $T$ with an average charge per edge of at most $1 + \frac{1}{\lceil \frac{k}{2} \rceil}$.

**Begin**
$\mathcal{C} \leftarrow \emptyset$.

{ Phase 1 }
$i \leftarrow 1$
**While** $|T| > \lfloor \frac{3k}{2} \rfloor$ **Do**
  $|T_i| \leftarrow |T_i \cap T|$.
  **If** $|T_i| < \lceil k/2 \rceil$ **Then**
    $i \leftarrow i+1$
  **Else If** $\lceil k/2 \rceil \leq |T_i| \leq k$ **Then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup T_i$, $T \leftarrow T \setminus \{T_i\}$, $i \leftarrow i+1$
  **Else If** $|T_i| > k$ **Then**
    **While** $|T_i| \geq \lceil k/2 \rceil$ **Do**
      Let $p \leq i-1$ be largest so that $\sum_{j=1, j \in ch(i) \cap T}^{p} (|T_j| + 1) \leq k$
      $T' \leftarrow \cup_{j=1, j \in ch(i) \cap T}^{p} (T_j \cup \{(i,j)\})$
      Update: $\mathcal{C} \leftarrow \mathcal{C} \cup \{T'\}$, $T \leftarrow T \setminus \{T'\}$, $|T_i| \leftarrow |T_i \cap T|$
    **End**
    $i \leftarrow i+1$
**End**

{ Phase 2 }
**If** $|T| \leq k$ **Then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{T\}$
**Else**
  **Begin**
  Let $i$ be the smallest index such that $|T_i| > k$.
  Let $p \leq i-1$ be largest so that $\sum_{j=1, j \in ch(i) \cap T}^{p} (|T_j| + 1) \leq k$
  $T' \leftarrow \cup_{j=1, j \in ch(i) \cap T}^{p} (T_j \cup \{(i,j)\})$, $\mathcal{C} \leftarrow \mathcal{C} \cup \{T'\} \cup \{T \setminus T'\}$
  **End**
**Output** $\mathcal{C}$
**End**

**Lemma 3.1** *Given a graph $T = (V, E)$ with $|E| > k$, rooted_tree_cover outputs a $k$-EP solution with an average charge of at most $1 + \frac{1}{\lceil \frac{k}{2} \rceil}$.*

**Proof**

When the algorithm reaches a vertex $i$ such that $|T_i| > k$, it covers and removes child-trees of $i$ until $|T_i| < \lceil \frac{k}{2} \rceil$. Thus, the algorithm maintains the invariant condition that $|T_i| < \lceil \frac{k}{2} \rceil$ after it visits vertex $i$.

By construction, the algorithm never adds a tree to $C$ with more than $k$ edges. We now show that in Phase 1 it never adds a tree to $C$ with fewer than $\lceil \frac{k}{2} \rceil$ edges. Consider a case where $|T_i| > k$ when the algorithm reaches vertex $i$. Let $q$ be the smallest vertex index such that $q \in ch(i) \cap T$ and $q > p$. By definition, $\Sigma_{j=1, j \in ch(i) \cap T}^{p}(|T_j| + 1) + (|T_q| + 1) > k$. Thus, $\Sigma_{j=1, j \in ch(i) \cap T}^{p}(|T_j| + 1) > k - (|T_q| + 1)$. Since, $|T_q| < \lceil \frac{k}{2} \rceil$, then $|T_q| + 1 \leq \lceil \frac{k}{2} \rceil$. Thus, $|T'| = \Sigma_{j=1, j \in ch(i) \cap T}^{p}(|T_j| + 1) \geq \lceil \frac{k}{2} \rceil$. Therefore, any tree added to $C$ in Phase 1 has at least $\lceil \frac{k}{2} \rceil$ and no more than $k$ edges and it follows that the average charge per edge is at most $1 + \frac{1}{\lceil \frac{k}{2} \rceil}$.

Since $|T| > \lfloor \frac{3k}{2} \rfloor$ at the start of Phase 1 and the algorithm covers at most $k$ edges in each iteration of the **For** loop, and $|T| \leq \lfloor \frac{3k}{2} \rfloor$ at the start of Phase 2, it follows that $|T| \geq \lceil \frac{k}{2} \rceil$ at the start of Phase 2. Thus, if $|T| \leq k$ then every tree in $C$ has at least $\lceil \frac{k}{2} \rceil$ edges and the average charge per edge is at most $1 + \frac{1}{\lceil \frac{k}{2} \rceil}$.

If $|T| > k$ at the start of Phase 2, then it is possible that $|T \setminus T'| < \lceil \frac{k}{2} \rceil$ (see Appendix A for an example). However, when $|T| > k$ then the algorithm covers the last $|T|$ edges of the input tree with $|T| + 2$ vertices. Thus the average charge per edge for the edges covered in Phase 2 is at most $\frac{|T|+2}{|T|} \leq \frac{k+3}{k+1} \leq 1 + \frac{1}{\lceil \frac{k}{2} \rceil}$. ■

**Lemma 3.2** *The complexity of rooted_tree_cover is $O(|E|)$, for an input tree $T = (V, E)$.*

**Proof**

For a given vertex $i$, Phase 1 of the algorithm inspects each edge $(i, j)$, where $j \in ch(i)$, at most twice: once when it calculates $|T_i|$ and possibly one more time when it finds the largest

$p$ such that $\sum_{j=1,j\in ch(i)\cap T}^{p}(|T_j|+1) \leq k$. Since it inspects each edge of the tree at most twice, the complexity of Phase 1 is $O(|E|)$. Similarly, the complexity of Phase 2 is also $O(|E|)$. Thus, the complexity of the algorithm is $O(|E|)$. ■

### 3.2 Covering Connected Graphs

The main steps of our $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation algorithm for an arbitrary connected graph $G = (V, E)$ may be summarized as follows:

**Procedure** $k$-**cover** $(G, k)$
**Input:** A connected graph $G = (V, E)$ and an integer $k$
**Output:** A $k$-EP solution $\mathcal{C}$ for $G$ with an average charge per edge of at most $1 + \frac{2}{k}$

**Step 1)** Choose a root vertex $r \in V$ and find a rooted spanning tree $T = (V, E_T)$ of $G$.
**Step 2)** Apply the algorithm *open_cycles* (see Subsection 3.2.1) to construct a tree $G' = (V', E')$.
**Step 3)** Cover $G'$ using *rooted_tree_cover*.
**Step 4)** Convert the cover of $G'$ to a cover $G$ with the algorithm *close_cover* (see Subsection 3.2.2).

Our algorithm begins by finding a rooted spanning tree $T = (V, E_T)$ of $G$. It then selects an edge $(u, v)$ in $E \setminus E_T$ and opens the cycle that would be created in $T$ by adding $(u, v)$ to $E_T$. This process is repeated for each edge in $E \setminus E_T$. Figure 1 illustrates this idea; the edges in bold-face form a spanning tree of the original graph rooted at vertex 1.

Step 3 of the algorithm applies rooted_tree_cover to produce a $k$-EP solution for $G'$. This solution is then "closed" using the algorithm *close_cover* to obtain a $k$-EP solution to original graph $G$. Figure 2 illustrates how a $k$-EP solution for the open graph in Figure 1 is "closed" to obtain a solution for the original graph.

After describing the details of open_cycles in Subsection 3.2.1 and close_cover in Subsection 3.2.2, we prove that the decomposition-based $k$-EP algorithm is a linear-time $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation algorithm for $k$-EP in Subsection 3.3.

### 3.2.1 Opening the Cycles of a Connected Graph

In addition to the notation introduced in the previous subsection, we need to define one more term before we present our algorithm for opening the cycles of a graph. Let $G = (V, E)$ and $G' = (V', E')$ be graphs such that $V \subseteq V'$ and $|E'| = |E|$. A function $g : V' \to V$ is called a *closing function on* $V'$ if the function $f : E' \to E$ defined as $f((u, v)) = (g(u), g(v))$ is one-to-one and onto. The open_cycles algorithm opens a given graph $G = (V, E)$ into a tree $G' = (V', E')$ and creates a closing function $g$ that is used in the last step of the $k$-cover algorithm to map the cover of $E'$ back to a partition of the original edge set $E$.

**Procedure open_cycles** $(G, T, r)$
**Input:** A graph $G = (V, E)$, a spanning tree $T = (V, E_T)$ of $G$ rooted at $r$
**Output:** A tree $G' = (V', E')$ rooted at $r$ where $V \subseteq V'$, $|E'| = |E|$,
a closing function $g : V' \to V$ on $V'$.
**Begin**
$V' \leftarrow V, E' \leftarrow E_T$, $i \leftarrow |V|$
**For All** $v \in V$
    $g(v) \leftarrow v$
**For All** $(u, v) \in E \setminus E_T$
    **Begin**
    $i \leftarrow i + 1$
    $V' \leftarrow V' \cup \{i\}$
    $E' \leftarrow \{(u, i)\}$
    $ch(u) \leftarrow ch(u) \cup \{i\}$, $ch(i) \leftarrow \emptyset$
    $g(i) \leftarrow v$
    **End**
**Output** $G'$, $g$
**End**

**Lemma 3.3** *The graph $G' = (V', E')$ produced by open_cycles is a rooted tree.*

**Proof**

The algorithm begins by setting $G' = T$. Thus, $G'$ is a rooted tree before the **For** loop begins

which establishes the base case for an inductive argument to show that the **For** loop maintains the invariant condition that $G'$ is a rooted tree.

Suppose that $G' = (V', E')$ is a rooted tree after iteration $i - 1$. Without loss of generality, let $(u, v)$ be the next edge of $E \setminus E_T$ processed in the **For** loop. Since $G'$ is a rooted tree and $u \in V \subseteq V'$, there is a unique path between the root vertex $r$ and $u$. Thus, there is a unique path between $r$ and the new vertex $i$ when $i$ is added to $V'$ and $(u, i)$ is added to $E'$. Since $(u, i)$ is the only edge incident to $i$, there is still a unique path between $r$ and every vertex in $V' \setminus \{i\}$. Thus, the graph $\hat{G} = (V' \cup \{i\}, E' \cup \{(u, i)\})$ is a rooted tree. ■

**Lemma 3.4** *When the algorithm terminates,* $|E'| = |E|$.

**Proof**

This is true by construction. The **For** loop adds exactly one edge to $E'$ for each of $E \setminus E_T$ and $E' = E_T$ prior to the loop. ■

**Lemma 3.5** *The function* $g : V' \to V$ *produced by the algorithm is a closing function on* $V'$.

**Proof**

Let $\hat{E} = E \setminus E_T$, $\hat{E}' = E' \setminus E_T$. Consider an edge $(u, v) \in \hat{E}$. When the **For** loop processes $(u, v)$, it adds a vertex $i$ to $V'$, sets $g(v) = i$ and adds $(u, i)$ to $E'$. Clearly, $f((u, i)) = (u, v)$. Since the algorithm never adds another edge to $E'$ that is incident to $i$, $(u, i)$ is the only edge in $\hat{E}'$ such that $f((u, i)) = (u, v)$. Since the **For** loops adds exactly one edge to $E'$ for each edge in $\hat{E}$, it follows that the function $f((u, v)) = (g(u), g(v))$ is one-to-one and onto for the domain $\hat{E}'$ and range $\hat{E}$.

Since $g(v) = v$ for all $v \in V$, $f : E' \rightarrow E$ is one-to-one and onto for the domain $E_T$ and range $E_T$. It follows that $f$ is one-to-one and onto for the domain $E' = E_T \cup \hat{E}'$ and range $E = E_T \cup \hat{E}$. ■

**Lemma 3.6** *The complexity of open_cycles is $O(|E|)$ for an input graph $G = (V, E)$.*

**Proof**

There are $|E|$ iterations of the **For** loop and each step of loop can be done in constant time. Therefore, the complexity of the algorithm is $O(|E|)$. ■

### 3.2.2 Closing a $k$-EP Solution

We now present the procedure close_cover for the last step of the $k$-cover algorithm. This procedure takes the original graph $G$, the tree $G'$ and closing function $g$ produced by the open_cycles algorithm and the cover $\mathcal{C}'$ of $G'$ produced by rooted_tree_cover and outputs a cover $\mathcal{C}$ of $G$. As we will show in Lemma 3.9, the charge per edge in $\mathcal{C}$ is less than or equal to the charge per edge in $\mathcal{C}'$.

**Procedure close_cover** $(G', G, \mathcal{C}' = \{C'_1, C'_2, \ldots, C'_\ell\}, g)$
**Input:** Graphs $G' = (V', E')$ and $G = (V, E)$ with $|E'| = |E|$, a cover $\mathcal{C}'$ of $E'$ and a closing function $g : V' \rightarrow V$ on $V'$
**Output:** A cover $\mathcal{C}$ of $E$

**Begin**
$\mathcal{C} \leftarrow \emptyset$
**For** $i = 1$ to $|\mathcal{C}'|$
    $C_i \leftarrow \emptyset$
    **For All** $(u, v) \in C'_i$
        $C_i \leftarrow C_i \cup \{(g(u), g(v))\}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_i\}$
**Output** $\mathcal{C}$
**End**

**Lemma 3.7** *$\mathcal{C}$ is a partition of $E$.*

**Proof**

Since $\mathcal{C}'$ is a cover of $E'$, each edge $(u, v) \in E'$ is covered by exactly one $C_i \in \mathcal{C}'$. Since $g$ is a closing function on $V'$, each edge $(u, v) \in E'$ corresponds to a distinct edge $(g(u), g(v)) \in E$. Since $\mathcal{C}$ covers $E$, then it is a partition of $E$. ■

**Lemma 3.8** *If $\mathcal{C}'$ is a $k$-EP solution for $G'$, then $\mathcal{C}$ is a $k$-EP solution for $G$.*

**Proof**

Consider $C'_j \in \mathcal{C}'$ and the corresponding set $C_j \in \mathcal{C}$ constructed by the algorithm. Since $g$ is a closing function on $V'$, $|C_j| = |C'_j|$. Thus, if $|C'_j| \leq k$ for all $C'_j \in \mathcal{C}'$, then $|C_j| \leq k$ for all $C_j \in \mathcal{C}$. ■

**Lemma 3.9** *The average charge per edge for $\mathcal{C}$ is less than or equal to the average charge per edge of $\mathcal{C}'$.*

**Proof**

Consider an edge set $C'_j \in \mathcal{C}'$. Since $g$ is a closing function on $V'$, the corresponding edge set $C_j$ in $\mathcal{C}$ has the same size as $C'_j$. Let $V'_j = \{v_1, v_2, \ldots, v_n\}$ be the vertices in the subgraph of $G'$ induced by $C'_j$. Likewise, let $V_j$ be the set of vertices in the subgraph of $G$ induced by the edge set $C_j$. Since it is possible $g(x) = g(y)$ for $x, y \in V'_j$, $|\cup_{i=1}^{n} \{g(v_i)\}| \leq |V'_j|$. Thus $\frac{|V_j|}{|C_j|} \leq \frac{|V'_j|}{|C'_j|}$. This implies that the average charge per edge for $\mathcal{C}$ is less than or equal to the average charge per edge of $\mathcal{C}'$. ■

**Lemma 3.10** *The complexity of close_cover is $O(|E|)$.*

**Proof**

Each step of the inner **For** loop can be done in constant time and the inner loop executes exactly once per edge in $E'$. Since $|E'| = |E|$, the complexity of the inner **For** loop is $O(|E|)$. The other steps of the outer **For** loop (e.g. $i \leftarrow i + 1$), can each be done in constant time and are executed at most $|C'|$ times each. Since the $|C'| \leq |E|$, the execution of these steps is $O(|E|)$ and the overall complexity of the algorithm is $O(|E|)$. ■

## 3.3 Approximation Ratio and Complexity of the $k$-Cover Algorithm

In this Subsection, we show that the $k$-cover algorithm gives a solution which uses at most $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$ times the minimum number of vertices for $k$-EP. We also show that the complexity of the algorithm is linear in the size of the graph. In order to apply the results from the previous subsections, we first restate the $k$-cover algorithm as follows:

**Procedure** $k$**-cover** $(G, k)$
**Input:** A connected graph $G = (V, E)$ and an integer $k$
**Output:** A $k$-EP solution $\mathcal{C}$ for $G$ with an average charge per edge of at most $1 + \frac{2}{k}$

**Begin**
**Step 1)** Choose a root vertex $r \in V$ and find a rooted spanning tree $T = (V, E_T)$ of $G$.
**Step 2)** Let $G' = (V', E')$ and $g$ be the graph and closing function
output by open_cycles$(G, T, r)$.
**Step 3)** Let $\mathcal{C}'$ be the cover of $G'$ output by rooted_tree_cover$(G', r, k)$.
**Step 4)** Let $\mathcal{C}$ be the cover of $G$ output by close_cover$(G', G, \mathcal{C}', g)$.
**Return** $\mathcal{C}$
**End**

### 3.3.1 Approximation Ratio

**Theorem 3.1** *For a given graph $G = (V, E)$ and an integer $k \geq 3$, $k$-cover is a $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation algorithm for $k$-EP.*

**Proof**

From Lemma 3.1, edges covered by rooted_tree_cover are covered at an average charge of at most $1+\frac{1}{\lceil \frac{k}{2} \rceil}$. Thus, the algorithm covers $E'$ at an average charge at most $1+\frac{1}{\lceil \frac{k}{2} \rceil}$. From Lemmas 3.7 and 3.8, $\mathcal{C}$ is a $k$-EP solution for the original graph $G$. The average charge per edge for $\mathcal{C}$ is at most $1+\frac{1}{\lceil \frac{k}{2} \rceil}$ by Lemma 3.9. From Lemma 2.1, the optimal charge for any cover is at least $\frac{1}{k}\lceil \frac{1+\sqrt{8k+1}}{2} \rceil$. Therefore, the algorithm uses at most $(1+\frac{1}{\lceil \frac{k}{2} \rceil}) \div \frac{1}{k}\lceil \frac{1+\sqrt{8k+1}}{2} \rceil = \frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$ times the minimum number of vertices and is thus a $\frac{k(1+\frac{1}{\lceil \frac{k}{2} \rceil})}{\lceil \frac{1+\sqrt{8k+1}}{2} \rceil}$-approximation algorithm for $k$-EP. ■

### 3.3.2 Complexity

**Theorem 3.2** *For a given graph $G = (V, E)$ the complexity of the $k$-cover algorithm is $O(|E|)$.*

**Proof**

Step 1 of the algorithm, finding a rooted spanning tree, be done $O(|E|)$ time [HT73]. From Lemma 3.6, the complexity of Step 2 is also $O(|E|)$. From Lemmas 3.4 and 3.2, the complexity of Step 3 is $O(|E|)$. The complexity of Step 4 is $O(|E|)$ by Lemmas 3.4 and 3.10. Thus, the complexity of the entire algorithm is $O(|E|)$. ■

## 4 Approximation Scheme for Even Values of $k$

In this section, we present an approximation scheme for $k$-EP for even values of $k$. This approximation scheme is based on using a post-processing procedure to improve the solution produced by rooted_tree_cover. The resulting cover $\mathcal{C}$ will contain at most one tree with fewer than $\frac{k}{2}+1$ edges; the rest of the trees in $\mathcal{C}$ will have at least $\frac{k}{2}+1$ and at most $k$ edges.

## 4.1 Post-Processing Procedure

Let $T = (V, E)$ be a rooted tree with root vertex $r$ and let $\mathcal{C}$ be the partition formed by rooted_tree_cover. $\mathcal{C}$ induces a tree of subtrees $T(\mathcal{C})$ as follows. Every subtree $C_i \in \mathcal{C}$ is represented by a vertex in $T(\mathcal{C})$. There is at most one subtree in $\mathcal{C}$ that contains fewer than $\frac{k}{2}$ edges. If such a subtree exists we root $T(\mathcal{C})$ at this subtree; otherwise, we can pick an arbitrary subtree as the root. In this way we assure that in the first step of the post-processing routine all the subtrees (except possibly the root subtree) have at least $\frac{k}{2}$ edges. Since $\mathcal{C}$ is $k$-EP solution, none of the subtrees will have more than $k$ edges. If two subtrees $C_i$ and $C_j$ in $\mathcal{C}$ have a common vertex $v$ such that one of them contains an edge from the unique path in $T$ from $v$ to $r$, then the corresponding vertices in $T(\mathcal{C})$ are connected by an edge (see Appendix B for an example of this construction).

In addition to the terms defined in Subsection 3.1, we say that vertices $i$ and $j$ in $T(\mathcal{C})$ are *brothers* if they have the same parent vertex and the corresponding subtrees, $C_i$ and $C_j$, in $\mathcal{C}$ have a common vertex. For convenience, we also say that subtrees $C_i$ and $C_j$ are brothers. Furthermore, if $p$ is the parent of vertex $i$ in $T(\mathcal{C})$, we say that subtree $C_p$ is the parent of subtree $C_i$. Observe that two children of the same parent subtree need not be brothers. If subtrees $C_i$ and $C_j$ are both children of subtree $C_p$, they are brothers only if $C_i \cup C_j$ induces a connected subtree of $T$.

As with rooted_tree_cover, our post-processing routine uses a post order traversal to visit the vertices of $T(\mathcal{C})$. In following discussion, let $C_i$ be the subtree corresponding to the vertex $i$ of $T(\mathcal{C})$ currently being visited and let $C_p$ be its parent. Our procedure maintains the invariant condition that all the descendants of the current subtree have at least $\frac{k}{2} + 1$ (and at most $k$)

edges. The routine works as follows. If $|C_i| \geq \frac{k}{2} + 1$, then we move to the next vertex in $T(\mathcal{C})$ as indicated by the post order. Otherwise, $|C_i| \leq \frac{k}{2}$ edges and there are three cases to consider:

1. If $C_i$ has a brother $C_j$ with $\frac{k}{2}$ edges, then we replace this pair of brothers by their union to create a subtree with at most $k$ edges. That is, we let $C_j = C_i \cup C_j$ and remove $C_i$ from $\mathcal{C}$.

2. Otherwise, if the parent $C_p$ of $C_i$ has at most $\frac{k}{2}$ edges we replace the parent and this child by their union to create a subtree with at most $k$ edges. That is, we let $C_p = C_i \cup C_j$ and remove $C_i$ from $\mathcal{C}$.

3. Otherwise, we repeat the following process until $\frac{k}{2} + 1 \leq |C_i| \leq k$:

   Let $u \in V$ be the vertex that subtree $C_i$ shares with its parent $C_p$. If $u$ is a leaf in $C_p$, then we move the adjacent edge $(u, v) \in C_p$ from the parent to the child. That is, $C_p \leftarrow C_p \setminus \{(u, v)\}$ and $C_i \leftarrow C_i \cup \{(u, v)\}$.

   If $u$ is not a leaf in the parent subtree, then it must have child-trees that are subtrees of $C_p$. Since $|C_p| \leq k$, at least one of the child-trees of $u$, say $T_u \subset C_p$, has at most $\frac{k}{2}$ edges. We move the edge set $T_u$ from $C_p$ to $C_i$. Since $|C_i| \leq \frac{k}{2}$ and $|T_u| \leq \frac{k}{2}$, the resulting subtree has at most $k$ edges.

After completing the operations described above for the appropriate case for the current subtree, we move to the next subtree indicated by the post order. We will terminate at the root of $T(\mathcal{C})$ which has no parent. An example of the routine is given in Appendix B. At the end of the post-processing routine, the root subtree may have fewer than $\frac{k}{2}$ edges. By construction,

however, the rest of the subtrees in the cover will have at least $\frac{k}{2}+1$ edges. As a result, we may state the following lemma:

**Lemma 4.1** *Given a tree $T=(V,E)$ and an even integer $k$, there is a partition of $E$ into subtrees $\mathcal{T}=\{T_1,T_2,\ldots,T_r\}$ such that $|T_i|\geq\frac{k}{2}+1$ for $T_i\in\mathcal{T}\setminus\{T_r\}$ and $|T_i|\leq k$ for all $T_i\in\mathcal{T}$.*

Note that after the post-processing procedure at most one subtree has fewer than $\frac{k}{2}+1$ edges. Let $E_r$ denote the edges of this special subtree. The remaining $|E|-|E_r|$ edges are covered by subtrees of size $\frac{k}{2}+1$ or greater. Thus, the edges in $E\setminus E_r$ are covered at a charge of at most $1+\frac{2}{k+2}$. Since the maximum charge per edge in $E_r$ is 2, the average charge per edge after post processing is at most

$$2\frac{|E_r|}{|E|}+(1+\frac{2}{k+2})\frac{|E|-|E_r|}{|E|}.$$

Now, suppose that $|E|>Mk$ for some large number $M$. Since $|E_r|\leq\frac{k}{2}$ and $\frac{|E|-|E_r|}{|E|}\leq 1$, the average charge per edge is at most $2\frac{|E_r|}{|E|}+(1+\frac{2}{k+2})\frac{|E|-|E_r|}{|E|}\leq\frac{1}{M}+(1+\frac{2}{k+2})$. By Lemma 2.1, the approximation ratio in this case ($|E|>Mk$) is at most

$$\frac{1}{M\lceil\frac{1+\sqrt{8k+1}}{2}\rceil}+\frac{k(1+\frac{2}{k+2})}{\lceil\frac{1+\sqrt{8k+1}}{2}\rceil}.$$

We assume the graph $G=(V,E)$ has a relatively large number of edges with respect to $k$ (otherwise we can find an optimal solution by enumerating all possibilities to cover $E$ by subsets with at most $k$ edges). Thus, our approximation scheme works as follows. Given $\epsilon>0$, we find the smallest $M$ such that $\frac{1}{M\lceil\frac{1+\sqrt{8k+1}}{2}\rceil}<\epsilon$. If $|E|\leq Mk$, then we find an optimal solution by enumerating all the possible ways to partition $E$ into subsets of at most $k$ edges. Otherwise,

if $|E| > Mk$, we apply the post-processing routine to the cover produced by rooted_tree_cover and the result is a solution with an approximation ratio of

$$\frac{k(1+\frac{2}{k+2})}{\lceil\frac{1+\sqrt{8k+1}}{2}\rceil}+\epsilon.$$

The complexity of the enumeration is at most $O(2^{\binom{Mk}{k}})$ (by substituting for $M$, we can express this complexity in terms of $\epsilon$ and $k$). The complexity of rooted_tree_cover followed by the post-processing routine is clearly polynomial in $|E|$. Thus, the post-processing routine will result in an approximation scheme for even $k$ values such that for any $\epsilon > 0$ there is an approximation algorithm providing a ratio of $\frac{k(1+\frac{2}{k+2})}{\lceil\frac{1+\sqrt{8k+1}}{2}\rceil}+\epsilon$ with complexity $O(f(\epsilon, k, |E|))$ where $f$ is polynomial in $|E|$ and exponential in $\epsilon$ and $k$.

# A Example of rooted_tree_cover

As described in Subsection 3.1, rooted_tree_cover would cover the edges of the tree shown in Figure 3 with three subtrees: the 4-edge set of boldface edges, the 3-edge set of edges indicated by dashed lines and the 5-edge subtree comprised of the other edges. Since $k = 7$, the algorithm does not cover any edges until it reaches vertex 5. The 4-edge subtree would be covered first. The edges of $T_5$ would then be removed from the tree as shown in Figure 4.

After the edges of $T_5$ are removed from the tree, there are eight edges left in the tree. Since $k = 7 < 8 \leq \lceil\frac{3k}{2}\rceil = 11.5$, the algorithm proceeds to the **Else** statement in Phase 2 and we get $p = 13$. The sizes of the child-trees of vertex 13 are $|T_5| = |T_6| = 0$, $|T_9| = 2$ and $|T_{12}| = 2$. So, 9 is the largest index $p$ less than 13 such that $\sum_{j=1, j\in ch(13)\cap T}^{p}(|T_j|+1) \leq 7$. Thus, the algorithm covers the remaining edges with $T' = \{(13,5),(13,6),(13,9),(9,7),(9,8)\}$

and $T \setminus T' = \{(13,12),(12,11),(12,10)\}$. Even though the last subtree covered has $3 < k/2$ edges, the average charge per edge is $15/12 < 9/7 = 1 + \frac{1}{\lceil \frac{k}{2} \rceil}$.

# B Example of the Post-Processing Routine

Figure 5 shows a tree rooted at node 16. Consider the following solution returned by rooted_tree_cover when $k = 6$: $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$ where $C_1 = \{(1,4),(2,4),(3,4)\}$, $C_2 = \{(5,8),(6,8),(7,8)\}$, $C_3 = \{(4,9),(8,9),(9,16),(10,16)\}$, and the five remaining edges are covered by $C_4$. This solution uses 19 vertices to cover 15 edges.

We begin the post-processing routine by visiting subtree $C_1$ and moving the edge $(4,9)$ from the parent subtree $C_3$ to $C_1$. Next, we visit subtree $C_2$ and move edge $(8,9)$ from $C_3$ to $C_2$. Thus, subtree $C_3$ only has two edges when it is visited by the post-processing routine. Notice that $C_3$ intersects its parent subtree $C_4$ at node 16 which is not a leaf node in $C_4$. The two child-trees of node 16 in $C_4$ are $T_1 = \{(11,13),(12,13),(13,16)\}$ and $T_2 = \{(14,15),(15,16)\}$. Since neither of these subtrees have more than three edges ($\frac{k}{2}$), we are free to merge either one with $C_3$. If we move the edge set $T_1$ from $C_4$ to $C_3$, the improved solution is as follows: $C_1 = \{(1,4),(2,4),(3,4),(4,9)\}$, $C_2 = \{(5,8),(6,8),(7,8),(8,9)\}$, $C_3 = \{(9,16),(10,16),(11,13),(12,13),(13,16)\}$, and $C_4 = \{(14,15),(15,16)\}$ which uses 18 nodes to cover 15 edges.

# References


[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[DT97] D. Dor and M. Tarsi. Graph decomposition is NP-complete: A complete proof of Holyer's conjecture. *SIAM Journal on Computing*, 26(4):1166–1187, August 1997.

[Hol81] I. Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, pages 713–717, November 1981.

[HT73] J. Hopcroft and R. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, June 1973.

[LSHK00] Y. Lee, H. Sherali, J. Han, and S. Kim. A branch-and-cut algorithm for solving an intra-ring synchronous optical network design problem. *Networks*, 35(3):223–232, 2000.

[MI91] S. Masuyama and T. Ibaraki. Chain packing in graphs. *Algorithmica*, 6(6):826–839, 1991.

[SVW98] A. Sutter, F. Vanderbeck, and L. Wolsey. Optimal placement of add/drop multiplexers: Heuristic and exact algorithms. *Operations Research*, 46(5):719–728, September/October 1998.

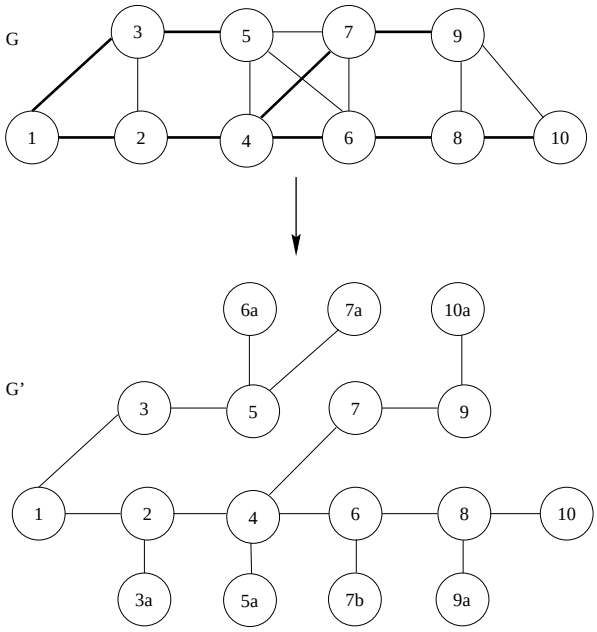


Figure 1: Opening the cycles of a connected graph.

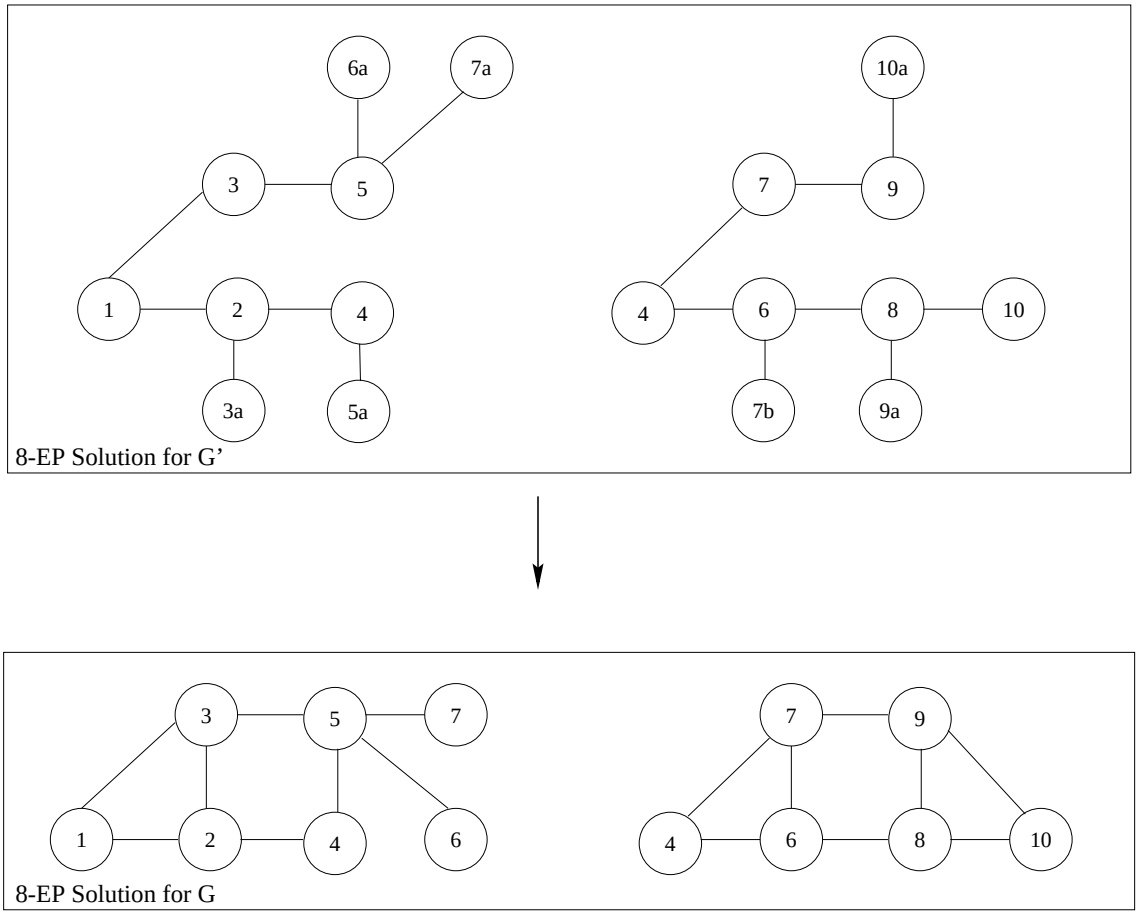


Figure 2: Closing a $k$-EP solution.

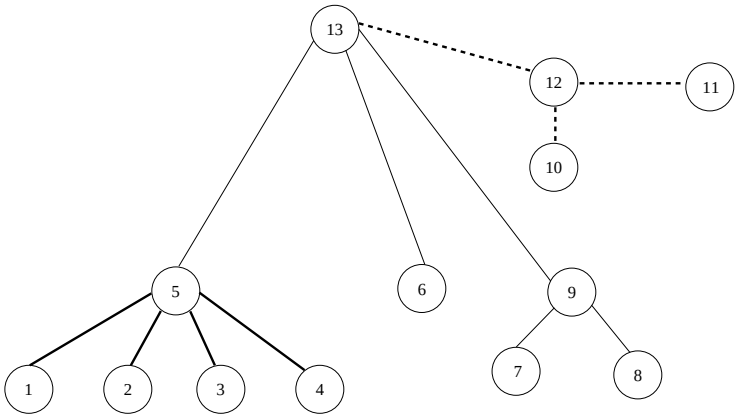


Figure 3: Example of rooted_tree_cover with $k = 7$.

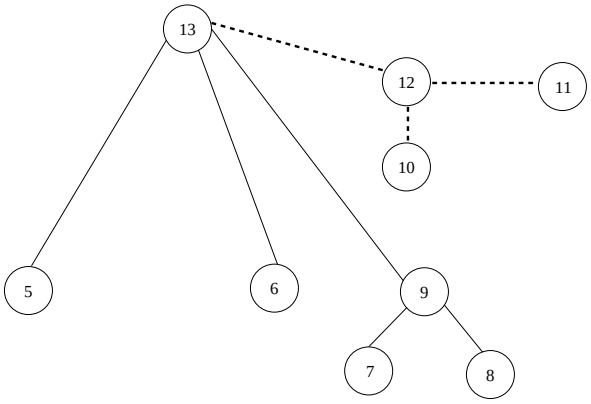


Figure 4: Example tree after the edges of $T_5$ are removed.

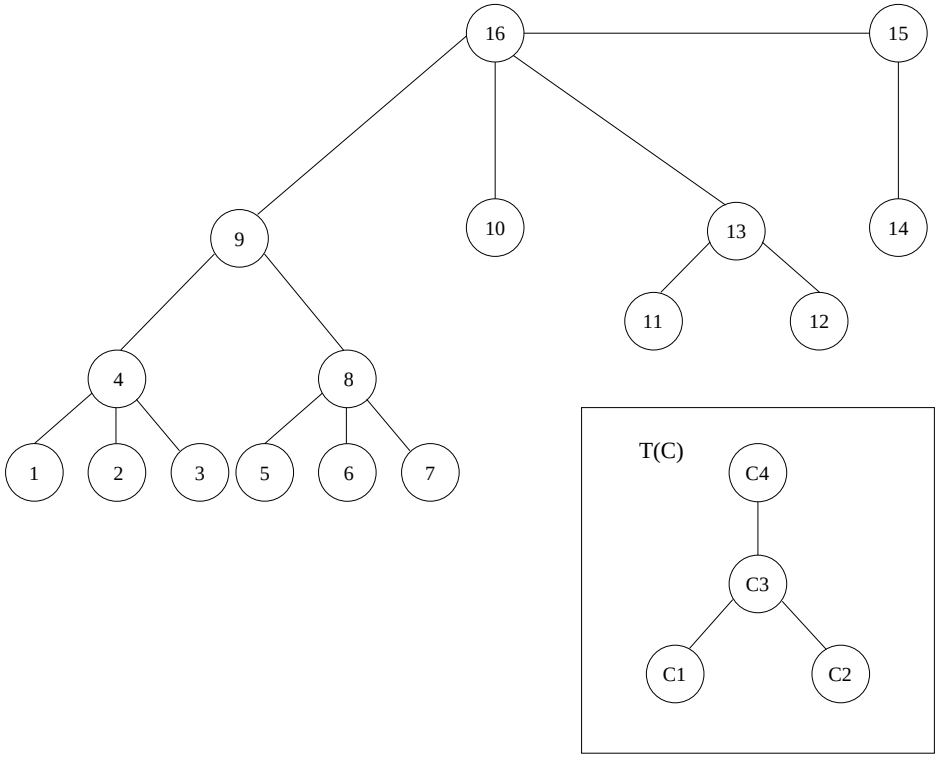


Figure 5: Post-Processing Example.