APPROXIMATION ALGORITHMS FOR COVERING GRAPH BY VERTEX-DISJOINT PATHS OF MAXIMUM TOTAL WEIGHT

by

S. Moran, I. Newman and Y. Wolfstahl

Technical Report #486
January 1988
Approximation Algorithms for Covering a Graph by Vertex-Disjoint Paths of Maximum Total Weight

Shlomo Moran, Ilan Newman and Yaron Wolfshtabl

† Faculty of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel 32000

‡ Department of Computer Science
Hebrew University, Givat Ram
Jerusalem, Israel

Abstract
We consider the problem of covering a weighted graph \( G = (V, E) \) by a set of vertex-disjoint paths, such that the total weight of these paths is maximized. This problem is clearly NP-complete, since it contains the Hamiltonian path problem as a special case. Three approximation algorithms for this problem are presented. First, we develop an algorithm for covering both directed and undirected graphs. The time complexity of this algorithm is \( O(|E| \log |E|) \), and its performance-ratio is \( 1/2 \). Second, we develop an algorithm for covering undirected graphs, whose performance-ratio is \( 1/2 \). This algorithm uses a weighted matching algorithm as a subroutine, which dominates the overall complexity of our algorithm. Finally, we describe an algorithm for covering directed graphs, whose performance-ratio is \( 1/2 \). This algorithm uses a Linear Programming algorithm as a subroutine, which dominates the overall complexity of our algorithm.
1. Introduction

Let $G = (V, E)$ be a (possibly directed) graph with no self-loops and parallel edges (or anti-parallel edges), and let $W : E \rightarrow \mathbb{Z}^+$ be a weight function. A path in $G$ is either a single vertex $v \in V$ or a sequence of distinct vertices $(v_1, v_2, \ldots, v_k)$ where $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k-1$ (in a directed graph, $(v_i, v_j)$ is taken to be an edge from $v_i$ to $v_j$). A path cover (abbr. cover) of $G$ is a set of vertex-disjoint paths which cover all the vertices of $G$. The weight of a cover $S$, denoted by $W_o(S)$, is the total sum of the weights of the edges included in $S$. A cover of the maximum possible weight is an optimal cover of $G$, and its weight is denoted by $\beta(G)$. We note that when $W_o(e)=1$ for each $e \in E_o$, an optimal cover of $G$ can be equivalently defined as a cover where the number of paths is minimized.

The concept of graph covering arises in various applications, such as mapping parallel programs to parallel architectures [9] and code optimization [2]. Unfortunately, the optimal covering problem is NP-complete even for cubic 3-connected planar graphs where no face has fewer than 5 edges [6]. There are, however, several results on optimal covering of restricted classes of graphs. Boesch, Chu and McHugh have derived in [1] an efficient optimal covering algorithm for undirected trees. Their result was generalized by Pinter and Wolfstahl [9], who developed a linear optimal covering algorithm for undirected graphs where no two cycles share a vertex. Moran and Wolfstahl have developed a linear optimal covering for cacti, i.e. undirected graphs where no edge lies on more than one cycle [8]. Boesch and Gimpel [2] have reduced the problem of covering a directed acyclic graph to the matching problem. In all those works $W_o(e)=1$ for each $e \in E_o$, and the optimal covering problem is formulated in the equivalent manner described above.

Motivated by the NP-completeness of the optimal covering problem, we set out to develop approximation algorithms for optimal covering of graphs. Define a covering algorithm to be an $r$-approximation algorithm if for any graph $G = (V, E)$ and a weight function $W : E \rightarrow \mathbb{Z}^+$, the algorithm produces a cover $S$ such that $\frac{W_o(S)}{\beta(G)} \geq r$. In this case, $r$ is called the performance-ratio of the algorithm. Three $r$-approximation covering algorithms are presented, exhibiting a complexity-performance trade-off. First, we develop a $\frac{1}{2}$-approximation algorithm for covering both directed and undirected graphs, called Algorithm A. The time complexity of this algorithm is $O(|E| \log |E|)$. Second, we develop a $\frac{3}{2}$-approximation algorithm for covering undirected graphs, called Algorithm B. This algorithm uses a weighted matching algorithm as a subroutine, the complexity of which dominates the overall complexity of Algorithm B. Finally, we describe a $\frac{3}{2}$-approximation algorithm for covering directed graphs, called Algorithm C. This algorithm uses a Linear Programming algorithm as a subroutine, the complexity of which dominates the overall complexity of Algorithm C.
2. A \( 1/2 \)-Approximation Covering Algorithm

Our first approximation algorithm, named algorithm A, is presented below. The algorithm, as described here, is intended for covering undirected graphs, but it can be easily modified to apply to directed graphs. Informally, algorithm A constructs the cover by repeatedly choosing the edge of the maximum weight over all edges that were not previously chosen or ruled out. Once an edge \( e=(u,v) \) is chosen for the cover, the edges that, by the choice of \( e \), cannot be chosen later, are ruled out. (These are the redundant edges, i.e. edges incident to vertices \( x \in \{u,v\} \) where \( x \) is not an end-point of a path in the cover).

**Algorithm A**

**Input:** An undirected graph \( G=(V,E) \) and a weight function \( W_G:E \rightarrow \mathbb{Z}^+ \).

**Output:** \( P_A \), a cover of \( G' \).

**Method:**

Initialize: \( E' \leftarrow \emptyset \). For each \( v \in V \), \( p(v) \leftarrow v \), \( o(v) \leftarrow v \).

(* If \( v \in V \) is an end-point of a path in the cover, then \( p(v) \) is the path covering \( v \), and \( o(v) \) is the other end-point of this path. *)

Sort the edges of \( E \) in descending order of weight.

Loop: while \( E \neq \emptyset \)

- Choose an edge \( e=(u,v) \in E \) such that \( W_G(e) \) is maximum.
- \( E \leftarrow E - \{e\} \).

If \( p(u) \neq p(v) \), then

- \( E' \leftarrow E' \cup \{e\} \). (* Add \( e \) to the cover *)
- \( M \leftarrow \{(x,y) \in E \mid x \in \{u,v\}, o(x) \neq x\} \). (* Redundant edges ... *)
- \( E \leftarrow E - M \). (* ... are ruled out. *)
- \( p(o(u)) \leftarrow p(o(v)) \). (* merging of paths *)
- \( o(o(u)) \leftarrow o(v) \).

Output: \( P_A \leftarrow \{ p \mid p \) is a maximal connected component in \( G'=(V,E') \} \)

**Theorem 1:** Algorithm A is a \( 1/2 \)-approximation covering algorithm. Moreover, the algorithm can be implemented in \( O(|E| \cdot \log |E|) \) time.

**Proof:** It is easy to see that A produces a cover of \( G \): The extraction of the edges of \( M \) from \( E \) in each iteration ensures that no vertex of \( G'=(V,E') \) is of degree exceeding 2. The test \( p(u) \neq p(v) \) ensures that \( G' \) contains no cycles. Thus, \( P_A \) is a cover of \( G \).
The following definitions are used to prove the claimed performance-ratio. Extend the notion of weight, such that for \( M \subseteq E \), the weight of \( M \) is defined to be \( W_0(M) = \sum_{e \in M} W_0(e) \). An edge \( e \in E' \) is called an \( E' \)-edge. Denote the number of \( E' \)-edges incident to a vertex \( v \in V \) by \( d_A(v) \). Given a path \( p \in P_A \), let \( e(p) \) denote the set of \( E' \)-edges included in \( p \), and let \( E_p = \{ (u,v) \mid e \in e(p), d_A(u)=1 \} \).

Let \( OPT(G) \) be an optimal cover of \( G \). Observe that \( OPT(G) \) can be viewed as the union of three sets, namely, \( E_1, E_2 \) and \( E_3 \), where

1. \( E_1 = \{ (u,v) \mid e \in E' \cap OPT(G) \} \).
2. \( E_2 = \{ (u,v) \mid e \in OPT(G) \setminus E', d_A(u)=d_A(v)=1 \} \).
3. \( E_3 = \{ (u,v) \mid e \in OPT(G) \setminus E', \max(d_A(u), d_A(v))=2 \} \).

Let \( e=(u,v) \) be an edge of \( E_2 \). Note that \( u \) and \( v \) are covered by the same path \( p \in P_A \) where \( |e(p)| > 1 \), for otherwise the algorithm would have included \( e \) in \( P_A \). Also, observe that \( W_0(e) \leq \min\{W_0(e')\} \). For otherwise, e.g. if \( W_0(e) > W_0(e_1) \) where \( e_1 \in E_p \), the algorithm would have chosen \( e \) for \( P_A \), instead of \( e_1 \). Hereafter, if \( e=(u,v) \in E_2 \) where \( u \) and \( v \) are covered by \( p \in P_A \), let \( S(e)=E_p \). Clearly, for each edge \( e \in E_2 \) where \( S(e)=\{e_1,e_2\} \), \( W_0(e) \leq \frac{1}{2}(W_0(e_1)+W_0(e_2)) \).

Let \( e=(u,v) \) be an edge of \( E_3 \), where, w.l.o.g., \( d_A(u)=2 \). If \( d_A(v)=1 \) and \( S=\{e_1,e_2\} \) is the set of \( E' \)-edges incident to \( u \), then \( W_0(e) \leq \min\{W_0(e_1), W_0(e_2)\} \). For otherwise, e.g. if \( W_0(e) > W_0(e_1) \), the algorithm would have chosen \( e \) for \( P_A \), instead of \( e_1 \). Using a similar argument, one can verify that if \( d_A(v)=2 \) and \( S=\{e_3,e_4\} \) is the set of \( E' \)-edges incident to \( v \), then \( W_0(e) \leq \min\{W_0(e')\} \) for some \( e \in S \). Hereafter, if \( e \in E_3 \) and \( S_1, S_2 \) are defined as above, then the set \( S \) satisfying the latter inequality is denoted by \( S(e) \) (if \( S_1 \) and \( S_2 \) both satisfy the inequality, let \( S(e)=S_1 \) ). Clearly, for each edge \( e \in E_3 \) where \( S(e)=\{e_1,e_2\} \), \( W_0(e) \leq \frac{1}{2}(W_0(e_1)+W_0(e_2)) \).

Using the said above, we find that

\[ W_0(OPT(G)) = W_0(E_1) + W_0(E_2) + W_0(E_3) \leq W_1 + W_2 + W_3 \]

where

\[ W_1 = \sum_{e \in E_1} W_0(e), \quad W_2 = \sum_{e \in E_2} \frac{1}{2}(W_0(e_1)+W_0(e_2)), \quad \text{and} \quad W_3 = \sum_{e \in E_3} \frac{1}{2}(W_0(e_1)+W_0(e_2)) \]

Let us now estimate the contribution of each edge \( e \in E' \) to \( W = W_1 + W_2 + W_3 \). Let \( p \) be a path of \( P_A \), and let \( e \) be an edge on \( p \).
Assume first that $e$ is an edge of $e(p) - E_p$.

- If $e \in \text{OPT}(G)$, then $W_o(e)$ appears once in $W_1$. Also, there are at most two edges $e' \in E_3$ such that $e \in S(e')$, so $W_o(e)$ appears at most twice in $W_3$, where in each such appearance it is multiplied by $\frac{1}{2}$. Note that $W_o(e)$ does not appear in $W_2$. Hence, $e$ contributes at most $2 \cdot W_o(e)$ to $W$.

- If $e \notin \text{OPT}(G)$, then $W_o(e)$ appears at most four times in $W_2$, where, in each such appearance, it is multiplied by $\frac{1}{2}$. Note that $W_o(e)$ does not appear in $W_1$ and $W_3$. Hence, $e$ contributes at most $2 \cdot W_o(e)$ to $W$.

Assume next that $e$ is an edge of $E_p$.

- If $e \in \text{OPT}(G)$, then $W_o(e)$ appears once in $W_1$. Also, there may be at most one edge $e' \in E_2$ such that $e'$ is incident to $e$, so $W_o(e)$ appears at most once in $W_2$, where it is multiplied by $\frac{1}{2}$. Furthermore, there is at most one edge $e' \in E_3$ such that $e \in S(e')$, so $W_o(e)$ appears at most once in $W_3$, where it is multiplied by $\frac{1}{2}$. Hence, $e$ contributes at most $2 \cdot W_o(e)$ to $W$.

- If $e \notin \text{OPT}(G)$, then there may be a single edge $e' \in E_2$ such that $e'$ is incident to $e$, so $W_o(e)$ appears at most once in $W_2$, where it is multiplied by $\frac{1}{2}$. Also, there are at most two edges $e' \in E_3$ such that $e \in S(e')$, so $W_o(e)$ appears at most once in $W_3$, where in each appearance it is multiplied by $\frac{1}{2}$. Note that $W_o(e)$ does not appear in $W_1$. Hence, $e$ contributes at most $\frac{3}{2} \cdot W_o(e)$ to $W$.

To summarize the said above, an edge $e$ on a path $p \in P_A$ contributes at most $2 \cdot W_o(e)$ to $W$. It follows that

$$\beta(G) = W_o(\text{OPT}(G)) = W_o(E_1) + W_o(E_2) + W_o(E_3) \leq W \leq \sum_{p \in P_A} \sum_{e \in E(p)} 2 \cdot W_o(e) = 2 \cdot W_o(E') = 2 \cdot W_o(P_A).$$

We now turn to the complexity of the algorithm. Assume $V = \{v_1, v_2, \ldots, v_n\}$. The following data structures are used. The set $E$ is represented by a doubly-linked list. The set $E'$ is represented by a list.

The set $V$ is represented by a table $V[1..n]$, where for each $v_i \in V$ the entry $V[i]$ contains the values of $p(v_i), o(v_i)$, and a pointer to an incidence list of $v_i$. This incidence list, denoted by $L(v_i)$, contains the vertices adjacent to $v_i$ in $G$. Each vertex $v_j$ in $L(v_i)$ is associated with a pointer to the edge $(v_i, v_j)$ in $E$, called an $E$-pointer. Clearly, the initialize step is linear in $|E|$, and the sort step can be implemented in $O(|E| \log |E|)$ time. Let $e = (v_i, v_j) \in E$ be an edge chosen at the head of the loop. Since the list $E$ is doubly-linked, the extraction of $e$ from $E$ is done in $O(1)$ time. The edges rendered redundant by the choice of $e$ are found, and immediately deleted, by tracing the $E$-pointers in the incidence lists of $v_i$ and $v_j$. Thus, the extraction of $M$ from $E$ requires $O(|M|)$ time, so the execution time of the loop is $O(|E|)$. 

Technion - Computer Science Department - Technical Report CS0486 - 1988
The output step is also linear in $|E|$. It follows that the run time of the algorithm is $O(|E| \cdot \log |E|)$ time. ■

We note that the sorting step can be omitted when $W_0$ is constant, resulting in an $O(|E|)$ algorithm. Furthermore, it can be shown that in graphs where $W_0(e) = k$ for each $e \in E$, the algorithm produces a cover $P_A$ that satisfies $\beta(G) \leq 2 \cdot W_0(P_A) - |P_A| \cdot k$. This bound is tight, that is, there are graphs where the execution of the algorithm attains equality. The proof is omitted.

3. A ½-Approximation Algorithm for Covering Undirected Graphs

In this section we describe and analyze a second approximation algorithm for optimal covering of undirected graphs, named Algorithm B. In doing so we rely on an algorithm for finding a maximum weight degree-constrained subgraph of a given graph. A polynomial algorithm for the latter problem can be derived from [3], where the linear programming approach is taken. Another algorithm for the maximum weight degree-constrained subgraph problem, which uses matching techniques, is given in [10].

Algorithm B

Input: An undirected graph $G = (V, E)$ and a weight function $W_0 : E \rightarrow \mathbb{Z}^+$. 
Output: $P_B$, a cover of $G$.

Method:

1. Obtain a graph $G_x = (V, X)$, $X \subseteq E$, where $\deg_{G_x}(v) \leq 2$ for each $v \in V$ and $W_0(X)$ is maximized. (* $G_x$ consists of isolated paths and cycles. *)
2. Let $C$ be the set of cycles in $G_x$.
3. $Y \leftarrow \{ e \mid e \text{ is an edge of minimum weight on some } c \in C \}$
4. $E' \leftarrow X - Y$
5. Output: $P_B \leftarrow \{ p \mid p \text{ is a maximal connected component in } G' = (V, E') \}$

Theorem 2: Algorithm B is a ½-approximation covering algorithm.

Proof: It is immediate that $P_B$ is a cover of $G$, so we begin by proving the claimed performance-ratio.

Let $OPT(G)$ be an optimal cover of $G$. Observe that $W_0(X) \geq W_0(OPT(G))$, since the weight of $X$ is maximized over all subgraphs of $G$ where the degree of no vertex exceeds 2.

Consider the sets $C$ and $Y$ defined in steps 2 and 3, respectively, of the algorithm. For each cycle $c \in C$, let $W_0(c)$ denote the sum of the weights of the edges on $c$. Each cycle $c \in C$ contains at least three
edges, so there is at least one edge, \( e \), on \( c \) with \( W_o(e) \leq \frac{W_o(c)}{3} \). It follows that

\[
W_o(Y) \leq \sum_{e \in G} \frac{W_o(c)}{3} \leq \frac{W_o(X)}{3}.
\]

Hence,

\[
W_o(P_b) = W_o(E') = W_o(X) - W_o(Y) \geq \frac{2W_o(X)}{3} \geq \frac{2W_o(OPT(G))}{3} = \frac{2\beta(G)}{3}.
\]

Theorem follows. ■

The complexity of step 1 of Algorithm B is that of the available algorithm for finding a maximum weight degree-constrained subgraph of \( G=(V,E) \) (here the degree bound is 2). The latter problem was reduced in [10] to the maximal weighted matching problem on a graph \( G_1=(V_1,E_1) \) where, in terms of \( G \), both \( |V_1| \) and \( |E_1| \) are \( O(|E|) \). The maximal weighted matching problem, in turn, can be solved on a graph \( G_1=(V_1,E_1) \) in \( O((|V_1|+|E_1|)+|V_1|\log|V_1|) \) time, where \( t(m,n)=O(m\log\log\log n) \), \( b=\max\{\frac{m}{n},2\} \) [4]. Hence, in terms of the original graph \( G \), step 1 can be executed in \( O(|E|\cdot t(|E|,|E|))+O(|E|\log|E|)\cdot O(|E|\cdot t(|E|,|E|))\cdot O(|E|^2\log\log n \cdot |E|)\) time. Steps 2-5 of Algorithm B can clearly be executed in \( O(|E|) \) time, so step 1 dominates the complexity of the algorithm.

4. A \( 3/2 \)-Approximation Algorithm for Covering Directed Graphs

In this section we describe a \( 3/2 \)-approximation algorithm for covering directed graphs, called Algorithm C. The following definitions are required.

Definitions: Let \( G=(V,E) \) be a directed graph where \( V=\{v_1,v_2,\ldots,v_n\} \), and let \( W_o: E_o \rightarrow Z^+ \) be a weight function. Then \( G_L=(V_L,E_L) \) and \( W_L:E_L \rightarrow Z^+ \) are a directed graph and a weight function, respectively, defined as follows:

1. \( V_L=\{x_i,y_i : v_i \in V\} \cup \{t\} \), and

\[
E_L = E_1 \cup E_2, E_1 = \{(x_i,y_j) : (v_i,v_j) \in E \}, E_2 = \{(y_i,x_i) : (t,y_i),(x_i,t) \in V \}. \]

That is, each \( v_i \in V \) is replaced in \( G_L \) by two vertices \( y_i \) and \( x_i \). Each edge \( (v_i,v_j) \in E \) is replaced in \( G_L \) by an edge \( (x_i,y_j) \). Also, for each \( v_i \in V \), \( G_L \) contains the edges \( (t,y_i) \) and \( (x_i,t) \) (see Fig 1).

2. \( W_L((x_i,y_j))=W_o(v_i,v_j) \) for each \( (v_i,v_j) \in E \), and \( W_L(e)=0 \) for each \( e \in E_2 \).

Insert Figure 1 here
Let $H = (V_H, E_H)$ be a spanning subgraph of $G_L$ where $V_H = V_L$, $E_H \subseteq E_L$, and $\deg_H^*(v) = \deg_H^*(v)$ for each $v \in V_H - \{t\}$ ($\deg_H^*(v)$ and $\deg_H^*(v)$ denote the in-degree and the out-degree, respectively, of $v$ in $H$.) Hereafter, such a subgraph is called a fair subgraph of $G_L$. It is easy to see that for each $v \in V_H - \{t\}$, $\deg_H^*(v) \leq 1$ and $\deg_H^*(v) \leq 1$. Hence, $H$ is a union of isolated vertices and directed cycles that share no vertex except, perhaps, $t$. Observe that $x_i$ is an isolated vertex in $H$ iff $y_i$ is an isolated vertex in $H$. In this case, $(x_i)$ is defined to be the path in $G$ which corresponds to the vertices $x_i$ and $y_i$. Let $p$ be a cycle in $H$. If $p = (y_i, x_i, y_{i+1}, x_{i+1}, \ldots, y_j, x_j, y_i)$ then the cycle in $G$ which corresponds to $p$ is defined to be $(v_i, v_{i+1}, \ldots, v_j, v_i)$. Similarly, if $p = (y_i, x_i, y_{i+1}, x_{i+1}, \ldots, y_j, x_j, t, y_i)$, then the path in $G$ which corresponds to $p$ is defined to be $(v_i, v_{i+1}, \ldots, v_j)$. Using this correspondence, $H$ uniquely defines a spanning subgraph of $G$, which is a union of directed vertex-disjoint paths and cycles. Let $G(H) = (V, E')$ be the subgraph of $G$ which $H$ defines. It is easily verified that $W_G(E') = W_L(E_H)$. Conversely, let $G'(V, E')$ be a spanning subgraph of $G$ where $\deg_G(v) \leq 2$ for each $v \in V$. Then $G'$ uniquely defines a spanning fair subgraph of $G_L$, say $H = (V, E_H)$, where $W_G(E') = W_L(E_H)$.

We are now able to describe Algorithm C. Given a directed graph $G = (V, E)$ and a weight function $W_G : E_G \rightarrow \mathbb{Z}^+$, the algorithm first constructs the graph $G_L$ and the corresponding weight function, $W_L$. Then, in a manner described below, the algorithm obtains a maximum weight fair subgraph (MWFS) of $G_L$, denoted by $H$. Next, $G(H)$ is derived from $H$. Using the above observations, $G(H)$ is a union of directed vertex-disjoint paths and cycles of $G$, having the maximum possible weight. Algorithm C terminates by deleting the edge of the minimum weight from each cycle in $G(H)$. Thus, as in Algorithm B, a cover whose weight is at least $\frac{3}{2}\beta(G)$ is obtained.

To complete the description of Algorithm C, we must show how a MWFS of $G_L = (V_L, E_L)$ is obtained. Let $E_L = \{e_1, e_2, \ldots, e_m\}$. Observe that the problem of finding a MWFS of $G_L$ can be formulated as the following Integer Linear Programming (ILP) problem:

maximize $W_L^* \vec{x}$ under the constraints:

1. for each $z \in V_L$, $\sum_{e \in E_L, e = (z,u)} X(e) - \sum_{e \in E_L, e = (u,z)} X(e) = 0$,

2. for each $e \in E_L$, $0 \leq X(e) \leq 1$, and

3. for each $e \in E_L$, $X(e)$ is an integer,

where $W_L^* = (W_L(e_1), W_L(e_2), \ldots, W_L(e_m))$ is the vector of weights and $\vec{x} = (X(e_1), X(e_2), \ldots, X(e_m))$ is a vector of the same dimension.
However, this ILP problem can be casted as a Linear Programming (LP) problem since the above constraints can be written as

(1) \[ A \cdot X = 0, \] and

(2) \[ \text{for each } e \in E_L, \ 0 \leq X(e) \leq 1. \]

where \( A \) is the incidence matrix of \( G_L \), that is: \( A(i,j) = 1 \) if the edge \( e_j \) is directed into the vertex \( v_i \), \( A(i,j) = -1 \) if it is oppositely directed, and \( A(i,j) = 0 \) otherwise. The integrality constraint was omitted since the matrix \( A \) is Totally Unimodular [7], which implies that all the basic solutions of the above linear program are integral.

Having reduced the task of finding a MWFS of \( G_L \) to the LP problem, we note that the complexity of the rest of Algorithm C is \( O(|E_L|) \). Thus, the task of finding a MWFS of \( G_L \) dominates the complexity of Algorithm C. It follows that the overall complexity of C is that of the available LP algorithm. The currently fastest LP algorithm runs in \( O((m+n)n^2 + (m+n)n^2 + nL) \) time, where \( m \) is the number of equations, \( n \) is the number of variables, and \( L \) is a bound on the number of bits in the input [11]. Observe that in our linear program, \( m = 2|V_L| \) and \( n = |E_L| \).

An interesting feature of Algorithm C is the following: if \( G \) is a directed acyclic graph (DAG), then algorithm C produces an optimal cover of \( G \) (since, in this case, \( G(H) \) is acyclic). Hence, Algorithm C generalizes the optimal covering algorithm for DAGs of [2].

5. Summary

We have presented a variety of \( r \)-approximation algorithms for the optimal covering problem. Left open, is the problem of finding a polynomial-time approximation scheme [5] for the problem. (Note that a fully polynomial-time approximation scheme [5] for the problem cannot exist unless \( P=NP \), since a polynomial-time algorithm for the Hamiltonian path problem can be derived from such a scheme.) It may also be interesting to device \( r \)-approximation covering algorithms where \( r > \frac{3}{2} \).

References


Figure 1

a. A vertex $v_i \in V$...

b. ... and the corresponding subgraph in $G_L$. 