Approximation Algorithms for Covering a Graph by Vertex-Disjoint Paths of Maximum Total Weight

Shlomo Moran
Faculty of Computer Science, Technion–Israel Institute of Technology, Haifa, Israel 32000

Ilan Newman
Department of Computer Science, Hebrew University, Givat Ram, Jerusalem, Israel

Yaron Wolfstahl
Faculty of Computer Science, Technion–Israel Institute of Technology, Haifa, Israel 32000

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, exhibiting a complexity-performance trade-off. First, we develop an algorithm for covering undirected graphs. The time complexity of this algorithm is $O(|E|\log|E|)$, and its performance-ratio is $\frac{1}{2}$. Second, we present an algorithm for covering undirected graphs, whose performance-ratio is $\frac{2}{3}$. This algorithm uses a maximum weight matching algorithm as a subroutine, which dominates the overall complexity of our algorithm. Finally, we develop an algorithm for covering directed graphs, whose performance-ratio is $\frac{2}{3}$. This algorithm uses a maximum weight bipartite matching algorithm as a subroutine, which dominates the overall complexity of the 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 $W_G: E \rightarrow Z^{+}$* 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* (abbrev. *cover*) of G is a set of vertex-disjoint paths that cover all the vertices of G. The *weight* of a cover S, denoted by $W_G(S)$, is the total sum of the weights of the edges

*Z^{+} is the set of positive integers.

NETWORKS, Vol. 20 (1990) 55–64

CCC 0028-3045/90/010055-010$04.00

included in S. A cover of the maximum possible weight is an *optimal cover* of G, and its weight is denoted by $\beta(G)$.

The concept of graph covering arises in various applications, such as mapping parallel programs to parallel architectures [10] and code optimization [3]. Unfortunately, the optimal covering problem is NP-complete even for cubic 3-connected planar graphs where no face has fewer than five edges [8]. There are, however, several results on optimal covering of restricted classes of graphs. Boesch et al. have derived in [2] an efficient optimal covering algorithm for undirected trees. Their result was generalized by Pinter and Wolfstahl [10], 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 [9]. Bodlaender has developed a polynomial time algorithm [4] for optimal covering of classes of graphs with bounded treewidth (see [1] for a related result). Boesch and Gimpel [3] have reduced the problem of covering a directed acyclic graph to the matching problem. All the above works consider nonweighted graphs only, in which case an optimal cover can be equivalently defined as a cover with the minimum number of paths.

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 $W_G: E \rightarrow Z^+$, the algorithm produces a cover S such that $[W_G(S)]/[\beta(G)] \geq r$. In this case, r is called the *performance-ratio* of the algorithm. Three r-approximation covering algorithms are presented. First, we develop a $\frac{1}{2}$-approximation algorithm for covering undirected graphs, called Algorithm **A**. The time complexity of this algorithm is $O(|E| \log |E|)$. Second, we present a $\frac{2}{3}$-approximation algorithm for covering undirected graphs, called Algorithm **B**. This algorithm uses a maximum weight matching algorithm as a subroutine, the complexity of which dominates the overall complexity of Algorithm **B**. Finally, we develop a $\frac{3}{4}$-approximation algorithm for covering directed graphs, called Algorithm **C**. This algorithm uses a maximum weight bipartite matching algorithm as a subroutine, the complexity of which dominates the overall complexity of Algorithm **C**.

2. A $\frac{1}{2}$-APPROXIMATION COVERING ALGORITHM

Our first approximation algorithm, named algorithm **A**, is presented below. The algorithm is intended for covering undirected graphs. Informally, Algorithm **A** operates as follows: It initially constructs a cover constituting of all the vertices of G and no edges. This initial cover, where each vertex is a path by itself, has a weight of zero. The algorithm then proceeds to increase the weight of the cover by adding edges between endpoints of existing paths so as to create longer paths. The edges used to create longer paths from existing ones are chosen in descending order of weight. 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 endpoint of a path in the cover.) When the weight of the cover cannot be increased any more, the algorithm terminates.

Algorithm A.

Input: An undirected graph $G = (V, E)$ and a weight function $W_G: E \to Z+$.

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 endpoint of a path in the cover, then $p(v)$ is the path covering v, and $o(v)$ is the other endpoint of this path.*)

Sort the edges of E in descending order of weight.

Loop: while $E \neq \emptyset$

do

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**

do (*u, v are endpoint of different paths*)

$E' \leftarrow E' \cup \{e\}$. (*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*)
$p(o(u)) \leftarrow p(v)$. (*In the new path just created, update $p(\,)$...*)
$t \leftarrow o(u)$; $o(o(u)) \leftarrow o(v)$; $o(o(v)) \leftarrow t$. (*... and $o(\,)$.*)

od

od

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

Theorem 1. Algorithm **A** is a $\frac{1}{2}$-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 $W_G(M) = \sum_{e \in M} W_G(e)$. 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 $\epsilon(p)$ denote the set of E'-edges included in p, and let $E_p = \{e = (u, v) | e \in \epsilon(p), d_A(u) = 1\}$.

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

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 $|\epsilon(p)| > 1$, for otherwise, the algorithm would have included e in P_A. Also, observe that $W_G(e) \leq \min_{e' \in E_p} \{W_G(e')\}$. For otherwise, e.g., if $W_G(e) > W_G(e_1)$, 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 $S(e) = \{e_1, e_2\}$, $W_G(e) \leq \frac{1}{2}(W_G(e_1) + W_G(e_2))$.

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 $S_1 = \{e_1, e_2\}$ is the set of E'-edges incident to u, then $W_G(e) \leq \min\{W_G(e_1), W_G(e_2)\}$. For otherwise, e.g., if $W_G(e) > W_G(e_1)$, 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 $S_2 = \{e_3, e_4\}$ is the set of E'-edges incident to v, then $W_G(e) \leq \min_{e' \in S}\{W_G(e')\}$ for some $S \in \{S_1, S_2\}$. 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 $S(e) = \{e_1, e_2\}$, $W_G(e) \leq \frac{1}{2}(W_G(e_1) + W_G(e_2))$.

Using the said above, we find that

$$W_G(OPT(G)) = W_G(E_1) + W_G(E_2) + W_G(E_3) \leq W_1 + W_2 + W_3,$$

where

$$W_1 = \sum_{e \in E_1} W_G(e), \qquad W_2 = \sum_{\substack{e \in E_2 \\ S(e) = \{e_1, e_2\}}} \tfrac{1}{2}(W_G(e_1) + W_G(e_2)),$$

and

$$W_3 = \sum_{\substack{e \in E_3 \\ S(e) = \{e_1, e_2\}}} \tfrac{1}{2}(W_G(e_1) + W_G(e_2)).$$

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 $\hat{e}(p) - E_p$.

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

Assume next that e is an edge of E_p.

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

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

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

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 $V[1 \ldots n]$, 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 $O(|E| \cdot \log|E|)$ 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 $O(|E|)$. 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_G is constant, resulting in an $O(|E|)$ algorithm. Furthermore, it can be shown that in graphs where $W_G(e) = k$ for each $e \in E$, the algorithm produces a cover P_A that satisfies $\beta(G) \leq 2 \cdot W_G(P_A) - |P'_A| \cdot k$, where P'_A is the number of paths that are not isolated vertices. This bound is tight, that is, there are graphs where the execution of the algorithm attains equality. The proof is omitted.

3. A $\frac{2}{3}$-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 [5], where the linear programming approach is taken. Another algorithm for the maximum weight degree-constrained subgraph problem, which uses matching techniques, is given in [11].

Algorithm B

Input: An undirected graph $G = (V, E)$ and a weight function $W_G: E \to Z^+$.

Output: P_B, a cover of G.

Method:

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

Theorem 2. Algorithm **B** is a $\frac{2}{3}$-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 $W_G(X) \geq W_G(OPT(G))$, 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 $W_G(c)$ 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 $W_G(e) \leq (W_G(c)/3)$. It follows that

$$W_G(Y) \leq \sum_{c \in C} \frac{W_G(c)}{3} \leq \frac{W_G(X)}{3}.$$

Hence,

$$W_G(P_B) = W_G(E') = W_G(X) - W_G(Y) \geq \frac{2W_G(X)}{3} \geq \frac{2W_G(OPT(G))}{3} = \frac{2\beta(G)}{3}.$$

The Theorem follows. ∎

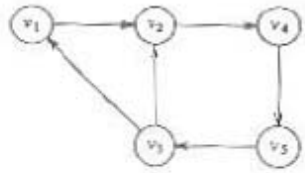
The complexity of Step 1 of Algorithm **B** is that of the available algorithm for finding a maximum weight degree-constrained subgraph of $G = (V, E)$ (here the degree bound is 2). The latter problem was reduced in [11] to the maximum weight matching proglem on a graph $G_1 = (V_1, E_1)$, where, in terms of G, both $|V_1|$ and $|E_1|$ are $O(|E|)$. The maximum weight matching problem, in turn, can be solved on a graph $G_1 = (V_1, E_1)$ in $O(|V_1| \cdot t(|E_1|, |V_1|) + |V_1| \log |V_1| f)$ time, where $t(m, n) = O(m \log\log\log_b n)$, $b = \max\{m/n, 2\}$ [6]. Hence, in terms of the original graph G, Step 1 can be executed in $O(|E| \cdot t(|E|, |E|) + O(|E| \log |E|) = O(|E|^2 \log\log\log_b |E|) = O(|E|^2 \log \log\log_b |V|)$ 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 $\frac{2}{3}$-APPROXIMATION ALGORITHM FOR COVERING DIRECTED GRAPHS

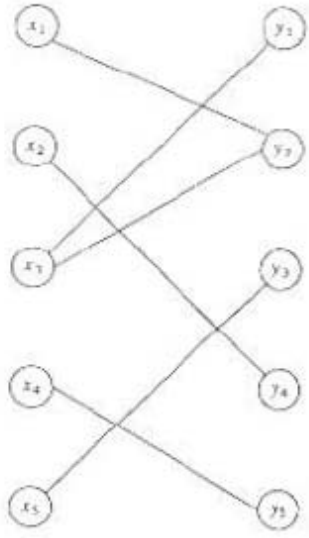
In this section, we develop a $\frac{2}{3}$-approximation algorithm for covering directed graphs, called Algorithm **C**. The following definitions are required:

Definitions. Let $G = (V, E)$ be a directed graph where $V = \{v_1, v_2, \ldots, v_n\}$, and let $W_G: E \to Z^+$ be a weight function. Then, $G_B = (X, Y, E_B)$ and $W_B: E_B \to Z^+$ are an undirected bipartite graph and a weight function, respectively, defined as follows (see Fig. 1):

(1) $X = \{x_i \mid v_i \in V\}$, $Y = \{y_i \mid v_i \in V\}$,
(2) $E_B = \{(x_i, y_j) \mid (v_i, v_j) \in E\}$, and
(3) $W_B((x_i, y_j)) = W_G((v_i, v_j))$ for each $(v_i, v_j) \in E$.



a. The graph $G=(V, E)$...



b. ... and the corresponding graph G_B.

FIG. 1.

Given a directed graph $G=(V, E)$, a *fair subgraph* of G is a spanning subgraph $F=(V, E_F)$, where each connected component is either a directed path or a directed cycle. Note that any matching M in G_B defines a fair subgraph of G, $F(M)=(V, E_F)$, and vice versa, in the following way: $(x_i, y_j) \in M \Leftrightarrow (v_i, v_j) \in E_F$. Moreover, M and E_F are of equal weight, that is, $W_G(E_F)=W_B(M)$. It follows that a maximum weight fair subgraph of G can be derived from a maximum weight matching in G_B.

We are now able to describe Algorithm C. Given a directed graph $G=(V, E)$ and a weight function $W_G: E_G \rightarrow Z^+$, the algorithm first constructs the graph G_B and the corresponding weight function, W_B. Then, the algorithm

obtains a maximum weight matching of G_B, denoted by M. Using the above observations, M corresponds to a fair subgraph of G, namely, $F(M)$, that has the maximum possible weight. The algorithm constructs $F(M)$ and deletes the edge of the minimum weight from each cycle in it, obtaining, as in Algorithm **B**, a cover whose weight is at least $\frac{2}{3}\beta(G)$.

We have thus reduced the problem of approximating a directed optimal cover of $G=(V, E)$ to the problem of finding a maximum weight bipartite matching (MWBM) of $G_B=(X, Y, E_B)$. Note that $O(|X \cup Y|)=O(|V|)$, $O(|E_B|)=O(|E|)$, and the transformation of G to G_B can be done in linear time. Hence, the overall complexity of **C** is that of the available MWBM algorithm on G_B. (The currently fastest MWBM algorithm for a graph $G=(V, E)$ runs in $O(|V| \cdot (|E|+|V| \log |V|))$ time [12].)

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, $F(M)$ is acyclic). Hence, Algorithm **C** generalizes the optimal covering algorithm for DAGS of [3].

5. SUMMARY

We have presented a variety of approximation algorithms for the optimal covering problem. Left open, is the problem of finding a polynomial-time approximation *scheme* [7] for the problem. (Note that a *fully* polynomial-time approximation scheme [7] 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 $r > \frac{2}{3}$.

ACKNOWLEDGMENTS

The authors wish to thank Avi Wigderson for simplifying Algorithm **C** and the anonymous referees for helpful remarks.

Note added in proof: E. Korach pointed out that Algorithm **A** can be applied to directed graphs, yielding performance ratio of $\frac{1}{2}$.

References

[1] S. Arnborg, J. Lagergren, and D. Seese, Problems easy for tree-decomposable graphs. *Proceedings of the 15th International Colloquium on Automata, Languages and Programming, July 1988, Tampere*, Springer Verlag, Germany, 1988, LNCS 317, 38–51.

[2] F. T. Boesch, S. Chen, and J. A. M. McHugh, On covering the points of a graph with point disjoint paths. *Proceedings of the 1972 Capital Conference on Graph Theory and Combinatorics*, Springer-Verlag, New York (1974), 201–212.

[3] F. T. Boesch and J. F. Gimpel, Covering the points of a digraph with point-disjoint paths and its application to code optimization. *J. ACM* **24**(2) (1977), 192–198.

[4] H. L. Bodlaender, Dynamic programming on graphs with bounded treewidth. *Proceedings of the 15th International Colloquium on Automata, Languages and Programming, July 1988, Tampere*, Springer Verlag, Germany, 1988, LNCS 317, 105–118.
[5] J. Edmonds and E. L. Johnson, Matching: A well solved class of integer linear programs. *Combinatorial Structures and Their Applications*, Gordon and Breach, New York (1969), 89–92.
[6] H. N. Gabow, Z. Galil and T. H. Spencer, Efficient implementation of graph algorithms using contraction. *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, New York, 1984*, 347–357.
[7] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco (1979), 136–137.
[8] M. Garey, D. Johnson, and R. E. Tarjan, The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.* **5** (1976) 707–714.
[9] S. Moran and Y. Wolfstahl, Optimal covering of cacti by vertex-disjoint paths. *Theoretical Computer Science*, to appear.
[10] S. Pinter and Y. Wolfstahl, On mapping processes to processors. *Int. J. Parallel Program.* **16**(1) (1987) 1–15.
[11] Y. Shiloach, Another look at the degree-constrained subgraph problem. *Inf. Process. Lett.* **12**(2) (1981) 89–92.
[12] Z. Galil, Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Sur.* **18**.(1) (1986) 23–38.

Received February 1988
Accepted May 1989