International Journal of Parallel Programming, Vol. 16, No. 1, 1987

On Mapping Processes to Processors in Distributed Systems[1]

Shlomit S. Pinter[2,3] and Yaron Wolfstahl[4]

Received December 1986; Revised June 1987

This paper is concerned with the implementation of parallel programs on networks of processors. In particular, we study the use of the network augmenting approach as an implementation tool. According to this approach, the capabilities of a given network of processors can be increased by adding some auxiliary links among the processors. We prove that the minimum set of edges needed to augment a line-like network so that it can accommodate a parallel program is determined by an optimal path cover of the graph representation of the program. An *optimal path cover* of a simple graph G is a set of vertex-disjoint paths that cover all the vertices of G and has the maximum possible number of edges. We present a linear time optimal path covering algorithm for a class of sparse graphs. This algorithm is of special interest since the optimal path covering problem is NP-complete for general graphs. Our results suggest that a "cover and augment" scheme can be used for optimal implementation of parallel programs in line-like networks.

KEY WORDS: Cacti; distributed systems; graph covering; mapping; network computers.

1. INTRODUCTION

A *network computer* (NC) is a set of autonomous, loosely-coupled processors, communicating via an interconnection network to solve a mutual problem. Advances in VLSI technology have greatly widened the

[1] A preliminary version of this paper was presented at the 6th IEEE Conference on Computer Communications (INFOCOM '87).
[2] This research is supported in part by National Semiconductor (Israel), Ltd.
[3] Faculty of Electrical Engineering, Technion, Israel Institute of Technology, Haifa, Israel 32000.
[4] Faculty of Computer Science, Technion, Israel Institute of Technology, Haifa, Israel 32000.

variety of design techniques for NCs. It is now technically possible and economically feasible to construct a NC by interconnecting thousands of processors, using components available in most hardware laboratories.(1,2) Moreover, users can interconnect VLSI processors to assemble NCs tailored to their specific applications.(3)

Mapping processes to processors is one of the most fundamental activities concerning the processing of parallel programs in NCs.(4–10) A good mapping results in a good correspondence between the interaction pattern of the processes and the interconnection paths among the processors. Such a correspondence is necessary in order to minimize the communication overhead in the system, which results in performance degradation.(11) The *mapping problem* arises when the logical interconnection structure of a parallel program differs from the physical interconnection architecture of the intended NC.

A parallel program is viewed as a *program graph*: the vertices represent the participating processes, and the edges connect processes that communicate during the execution of the program. Similarly, a NC is modeled by the underlying interconnection network, where the vertices represent the processors and the edges represent the communication lines connecting the processors. In our model, each process is to be mapped to a processor in a 1-1 fashion, and each edge in the program graph is to be mapped to a path in the network. In order to minimize the communication overhead, we seek to minimize the maximal distance in the network between images of vertices that are adjacent in the program graph.

This mapping problem is NP-complete in many of its formulations.(12) Motivated by the NP-completeness of the problem, some researchers have devised suboptimal mapping algorithms.(4,6,8) Others have shown that in certain restricted cases, optimal solutions can be efficiently found(9,10) (all these works except Ref. 4 allow *clustering* of processes, that is, mapping several processes to a single processor). Another approach to the mapping problem is to use a configurable architecture that can be adjusted to match a given program.(13)

In this paper we study the use of the network augmenting approach as a tool for mapping. According to this approach, the capabilities of a given NC can be increased by adding some auxiliary edges that decrease the distances between the processors.(14,15) In particular, mapping programs to augmented NCs is likely to result in improved performance.

A fundamental problem that stems from the augmenting approach is to find the minimum number of additional edges required to ensure the existence of a "good" mapping. Let $G = (V_G, E_G)$ be a program graph; an *optimal path cover of* G is a set of vertex-disjoint paths that cover all the vertices of G and has the maximum possible number of edges. We prove that a minimum set of edges needed to augment a line-like NC so that it can accommodate a program graph G is determined by an optimal path cover of G. Our proof is constructive: given an optimal path cover of G, we rigorously describe how to optimally augment the NC. The reason for considering a line-like NC rather than, say, a cube, is best expressed using the following quotation from Kung's fundamental survey(16):

One-dimensional linear arrays represent the simplest and also the most fundamental geometry for connecting processors... For a large number of important algorithms, this simple structure is all that is needed for communication.

Notable line-like NCs are the WARP(17) and the ESL.(18) The class of parallel algorithms suitable for implementation on line-like NCs includes algorithms for sorting,(19) data-structure manipulation,(16) recurrence evaluation,(20) and various graph problems.(21)

Our result suggests that a "cover and augment" scheme can be used for optimal implementation of parallel programs in line-like NCs. The feasibility of this scheme essentially depends on the existence of efficient algorithms for optimal path covering. Although this problem is NP-complete even for planar graphs (by a simple reduction from the planar Hamiltonian path problem(22)), there are classes of graphs where it can be solved efficiently. Optimal path covering algorithms for trees and directed acyclic graphs are presented in Refs. 23 and 24, respectively. These algorithms, however, are applicable only to acyclic graphs. Motivated by the fact that program graphs may definitely contain cycles, we present an efficient optimal path covering algorithm for a class of graphs that contain cycles. Specifically, we consider 1-*cacti*, defined to be simple undirected graphs where no vertex lies on more than one cycle. Being sparse, such program graphs are natural candidates for implementation on line-like NCs. Our algorithm finds an optimal path cover of a given 1-cactus, G, in time linear in the size of G.

The rest of this paper is organized as follows. In Section 2, we define some graph theoretic notions used throughout the paper. In Section 3, we establish a connection between network augmentation and covering a program graph. An optimal covering algorithm for 1-cacti is presented in Section 4. Finally, a summary is given in Section 5.

2. DEFINITIONS

Let $H=(V_H, E_H)$ and $G=(V_G, E_G)$ be two simple undirected graphs satisfying $|V_H| \geqslant |V_G|$. A set $E=\{e=(v_i, v_j) \mid v_i, v_j \in V_H, e \notin E_H\}$ is an *augmentation of* H *with respect to* G if adding the edges in E to E_H makes G isomorphic to a subgraph of H. For brevity, such an augmentation is also referred to as a (G, H)-augmentation. Figure 1 shows an augmentation of a line with respect to the star graph $K_{1,4}$. The additional edges are dashed; these edges together with the thick edges induce a graph isomorphic to $K_{1,4}$.

Let $\Gamma(G, H)$ be the set of all (G, H)-augmentations. The *augmenting number of H with respect to G* is the size of the smallest augmentation in $\Gamma(G, H)$. This number is denoted by $A(G, H)$, and is formally defined to be

$$A(G, H) = \min_{E \in \Gamma(G,H)} \{|E|\}$$

Let E be a (G, H)-augmentation. If $|E| = A(G, H)$, then E is an *optimal* (G, H)*-augmentation.*

A *path* in a graph $G = (V_G, E_G)$ is either a single vertex $v \in V_G$ or a sequence of distinct vertices $(v_1, v_2, \ldots, v_k)$ where $(v_i, v_{i+1}) \in E_G$ for $1 \leqslant i \leqslant k-1$. Given a path p, the number of vertices in p is denoted by $|p|$. A *path cover* of G is a set of vertex-disjoint paths which cover all the vertices of G. Henceforth, we use the term *cover* as an abbreviation for path cover.

Let S_G be a cover of G. We say that S_G *employs* an edge $e \in E_G$ if some path in S_G includes e. Let $n(S_G)$ denote the number of edges employed by S_G, and let $\Delta(G)$ be the set of all covers of G. The *covering number* of G, denoted by $C(G)$, is defined to be

$$C(G) = \max_{S \in \Delta(G)} \{n(S)\}$$

A cover S_G satisfying $n(S_G) = C(G)$ is an *optimal cover* of G, that is, a cover employing the maximum possible number of edges.

3. AUGMENTING A LINE-LIKE NETWORK

In this section we establish a connection between network augmentation and covering a program graph. A *line* is a simple undirected graph

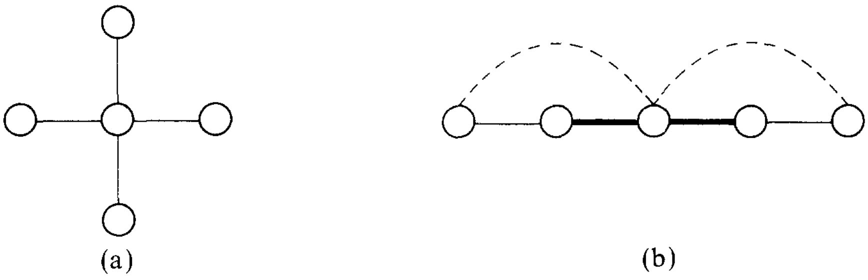

Fig. 1. Augmenting a line with respect to $K_{1,4}$; a. $K_{1,4}$; b. The augmented line.

$H=(V_H, E_H)$ where $V_H=\{v_1, v_2,..., v_{|V_H|}\}$ and $E_H=\{(v_i, v_{i+1}) \mid 1 \leqslant i<|V_H|\}$. The next theorem shows that the augmenting number of a line, H, with respect to a given program graph, G, is related to the covering number of G. The proof is constructive: given an optimal cover of G, we rigorously describe the construction of an optimal (G, H)-augmentation, and vice versa.

Theorem 1. Let $H=(V_H, E_H)$ be a line and let $G=(V_G, E_G)$ be a simple undirected graph, where $|V_G| \leqslant |V_H|$. Then $A(G, H)=|E_G|-C(G)$.

Proof. We assume in the proof that $|V_G|=|V_H|$. The case where $|V_G|<|V_H|$ is similar. Let $V_H=\{v_1, v_2,..., v_{|V_H|}\}$, and let $E_H=\{(v_i, v_{i+1}) \mid 1 \leqslant i<|V_H|\}$. Let $S_G=\{s^1,..., s^k\}$ be an optimal cover of G. For each path $s^i \in S_G$, denote the vertices of s^i by $s^i{}_1,..., s^i{}_l$ where $l=|s^i|$. Without loss of generality, we assume that $s^i{}_1,..., s^i{}_l$ is the order by which these vertices appear along s^i.

Define a function

$$f: V_G \rightarrow V_H \quad \text{by} \quad f(s^i{}_j)=v_m$$

where $1 \leqslant i \leqslant k$, $1 \leqslant j \leqslant |s^i|$ and

$$m=j+\sum_{0<n<i} |s^n|$$

Let $f^{-1}: V_H \rightarrow V_G$ be the corresponding inverse function. Next, augment H with the set of edges $\{(v_i, v_j) \mid v_i, v_j \in V_H, (f^{-1}(v_i), f^{-1}(v_j)) \in E_G, (v_i, v_j) \notin E_H\}$. Observe that exactly $|E_G|-n(S_G)$ edges are added, each of which corresponds to an edge not employed by the cover. Moreover, the images of all vertices adjacent in G are now adjacent in the augmented line, so the addition of edges is in fact a (G, H)-augmentation. It follows that $A(G, H) \leqslant |E_G|-n(S_G)=|E_G|-C(G)$.

We now establish the reverse inequality (hence equality). Let E be an optimal (G, H)-augmentation. By definition of a (G, H)-augmentation, there exists an adjacency-preserving 1-1 function mapping the vertices of G to the vertices of the augmented line. Let $\rho: V_G \rightarrow V_H$ be such a function. Consider the 1-1 mapping $\rho': E_G \rightarrow E_H \cup E$, defined to be $\rho'((v_i, v_j))=(\rho(v_i), \rho(v_j))$ for all $(v_i, v_j) \in E_G$. By the optimality of E, each edge in E is the image, under ρ', of an edge in E_G. Thus, there exists a set $S \subseteq E_H$ where $|S|=|E_G|-|E|$ and each edge in S is the image, under ρ', of an edge in E_G. Observe that S contains all the edges of a path cover of G. It follows that $C(G) \geqslant |S|=|E_G|-|E|=|E_G|-A(G, H)$, so $A(G, H) \geqslant |E_G|-C(G)$. ∎

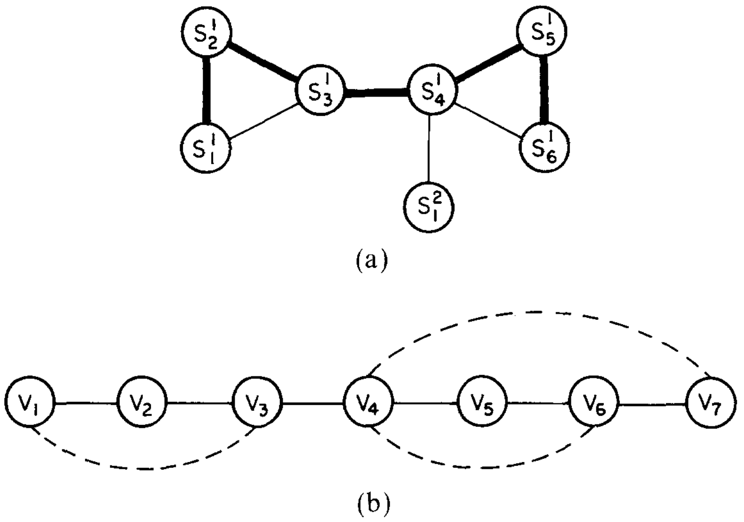

Fig. 2. An augmentation and an optimal cover; a. The graph G; b. The augmented line.

Figure 2 exemplifies the first part of the proof. Consider the graph G, shown in Fig. 2.a. The thick edges in G are employed by an optimal cover of G, namely, $\{(s^1{}_1, s^1{}_2, s^1{}_3, s^1{}_4, s^1{}_5, s^1{}_6), (s^2{}_1)\}$. The augmented line, where $f(s^1{}_j) = v_j$ for $1 \leqslant j \leqslant 6$ and $f(s^2{}_1) = v_7$, is shown in Fig. 2.b.

4. AN OPTIMAL COVERING ALGORITHM

We have shown in the previous section that once an optimal cover of a graph, G, is known, we can efficiently find an optimal augmentation of a line with respect to G. Thus, it is useful to have efficient optimal covering algorithms for classes of program graphs that are candidates for implementation on line networks. Unfortunately, the optimal cover problem is NP-complete even for planar graphs (by a reduction from the planar Hamiltonian path problem[22]). However, we now develop an efficient optimal covering algorithm for 1-cacti, that is, simple graphs where no vertex lies on more than one cycle.

The idea behind the algorithm is to repeatedly delete edges from the input 1-cactus without affecting its covering number. The resulting graph eventually reduces to a set of isolated paths, which constitutes an optimal cover of the input 1-cactus. The exact characterization of the edges that can be deleted from the 1-cactus without affecting its covering number is given in the following lemmas. This characterization is complete, in the sense that as long as the 1-cactus, G, has not reduced to a set of isolated paths, at least one lemma can be used to delete some edge(s) from G. The algorithm specifies the order by which these lemmas are to be applied such that the cover is found in linear time. We note that each of these lemmas is, in fact, applicable to any graph satisfying the requirements of the lemma. In particular, the lemmas may apply to cacti—graphs where no edge lies on more than one cycle—but they do not provide a complete framework for covering such graphs, as they do for 1-cacti.

Observe that an optimal cover can be equivalently defined as a cover where the total number of paths is minimum. This follows from the fact that the number of edges employed by a cover plus the number of paths in that cover is equal to the number of vertices in the graph. Thus, maximizing the number of edges employed by a cover is equivalent to minimizing the number of paths in that cover. The number of paths in a cover is called the *size* of the cover, and the size of an optimal cover of G is denoted by $\pi(G)$. The observation is heavily used in the algorithm: Given a 1-cactus, G, the algorithm finds a minimum size cover of G which, equivalently, is an optimal cover.

Definition. Let $G=(V_G, E_G)$ be a graph. A *trail starting at* $v_1 \in V_G$ is a path $p=(v_1, v_2, \dots, v_k)$ containing two or more vertices, where the degree of v_i is two for $1<i<k$ and the degree of v_k is one.

Lemma 1. Let $G=(V_G, E_G)$ be a graph. Let $v_1 \in V_G$ be a vertex of degree three or more which is the start-point of a trail $(v_1, v_2, \dots, v_k)$ and is adjacent to a vertex $w \neq v_2$ of degree one or two. Then $G'=(V_G, E'_G)$ where $E'_G = E_G - \{(v_1, x) \mid (v_1, x) \in E_G, x \notin \{v_2, w\}\}$ satisfies $\pi(G)=\pi(G')$.

Proof. Clearly $\pi(G) \leqslant \pi(G')$. To prove the reverse inequality (hence equality), we show that every optimal cover of G defines an equal-size cover of G'.

Let S_G be an optimal cover of G. If no edge in $E=\{(v_1, x) \mid (v_1, x) \in E_G, x \notin \{v_2, w\}\}$ is employed by S_G we are done since S_G is also a cover of G'. Otherwise, suppose that m edges in E are employed by S_G; observe that $m \in \{1, 2\}$. In this case, it is easily verified that there exists an equal-size cover of G which employs m edges from the set $\{(v_1, v_2), (v_1, w)\}$ but no edge from E. This latter cover is also a cover of G'. ▮

Definitions. A *crown* is a graph consisting of a single cycle which satisfies

1. At least one vertex on the cycle is of degree two.
2. Each vertex on the cycle is either of degree two or of degree three, being the start-point of a trail.

Given a crown C, the cycle underlying C is denoted by C°, and the degree of each vertex v in C is denoted by $deg_c(v)$. Let C be a crown that is a proper subgraph of a graph G. We say that C is an *end-cycle* of G (denoted $C \Subset G$) if there exists a vertex v, called the *separator*, on C° such that $\deg_c(v) = 2$ and v separates C from G. If each vertex u on C° satisfies $deg_c(u) = 2$, then C is an *end-cycle of order* 2. If each vertex u on C° (except for the separator) satisfies $deg_c(u) = 3$, then C is an *end-cycle of order* 3. Figure 3 shows a graph with two end-cycles; the left-hand end-cycle is of order 3 and the right-hand one is of order 2.

Lemma 2. Let $G = (V_G, E_G)$ be a graph. Let C be a subgraph of G which is either an isolated cycle or an end-cycle of order 2 (in both cases, let C° be the cycle underlying C). Let $v_1, v_2, \ldots, v_k$ be the vertices on C° (starting from the separator, if such exists). Then $G' = (V_G, E'_G)$ where $E'_G = E_G - \{(v_1, v_2)\}$ satisfies $\pi(G') = \pi(G)$.

Proof. Clearly $\pi(G) \leqslant \pi(G')$. To prove the reverse inequality we show that every optimal cover of G defines an equal-size cover of G'.

Let S_G be an optimal cover of G. If (v_1, v_2) is not employed by S_G then we are done. Otherwise, some other edge e on C° is not employed by S_G. It is easily seen that by modifying S_G to employ e rather than (v_1, v_2), one obtains an equal-size cover of G which does not employ (v_1, v_2). This latter cover is also a cover of G'. ▌

Definitions. Let $G = (V_G, E_G)$ be a graph. Let $p_1 = (v_1, \ldots, v_k)$ and $p_2 = (u_1, \ldots, u_l)$ be two vertex-disjoint paths in G, where $(v_k, u_1) \in E_G$. Then $p_1 \parallel p_2$ is the path $(v_1, \ldots, v_k, u_1, \ldots, u_l)$, obtained by appending p_2 to p_1.

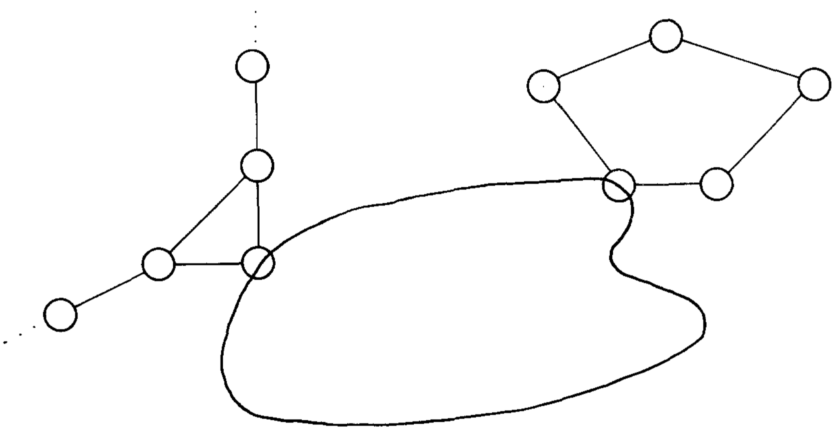
Fig. 3. A graph with end-cycles.

Given a path, say $p_1 = (v_1, ..., v_k)$, we define p_1^{-1} to be the path $(v_k, ..., v_1)$, obtained by reversing p_1.

Let C be an end-cycle of order 3 in a graph G. We say that C is an *even* (*odd*) end-cycle of order 3 if the number of vertices on C° is even (odd).

Lemma 3. Let $G = (V_G, E_G)$ be a graph. Let $C \Subset G$ be an even end-cycle of order 3, where $v_1, v_2, ..., v_k$ are the vertices on C°, starting from the separator. Then $G' = (V_G, E'_G)$ where $E'_G = E_G - \{(v_1, v_2)\}$ satisfies $\pi(G') = \pi(G)$.

Proof. Clearly $\pi(G) \leqslant \pi(G')$. To prove the reverse inequality, we show that every optimal cover of G defines an equal-size cover of G'. Let S_G be an optimal cover of G. If (v_1, v_2) is not employed by S_G then we are done. Suppose then that (v_1, v_2) is employed by a path $p = (..., v_1, v_2, ...) \in S_G$. If p covers vertices not in C, let p' be the prefix of p covering those vertices. An equal-size cover of G, denoted by $\bar{S}_G$, is defined by modifying S_G to cover C and the vertices in p' as follows. Let t_j be the trail starting at v_j, $2 \leqslant j \leqslant k$. The vertices in C are covered using the paths $p_1, ..., p_{k/2}$, where $p_i = t_{2i}^{-1} \parallel t_{2i+1}$ for $1 \leqslant i \leqslant (k/2) - 1$, and $p_{k/2}$ covers t_k and v_1. If p' is not empty, then $p_{k/2}$ extends to cover the vertices in p'. The optimality of $\bar{S}_G$ follows from the following two facts. First, $\bar{S}_G$ covers C and the vertices in p' using $k/2$ paths, which is minimum since at least $k/2$ paths must be used to cover the $k-1$ end-vertices of the trails t_j, $2 \leqslant j \leqslant k$. Second, $\bar{S}_G$ covers the rest of the vertices using the same number of paths as in S_G. Since $\bar{S}_G$ does not employ (v_1, v_2), it is also a cover of G'. ▌

Lemma 4. Let $G = (V_G, E_G)$ be a graph. Let $C \Subset G$ be an odd end-cycle of order 3, where $v_1, v_2, ..., v_k$ are the vertices on C°, starting from the separator. If v_1 is a start point of a trail in G, then $G' = (V_G, E'_G)$ where $E'_G = E_G - \{(v_1, v_2)\}$ satisfies $\pi(G') = \pi(G)$.

Proof. The proof is similar to that of Lemma 3. Clearly $\pi(G) \leqslant \pi(G')$. To prove the reverse inequality, we show that every optimal cover of G defines an equal-size cover of G'. Let S_G be an optimal cover of G. If (v_1, v_2) is not employed by S_G then we are done. Suppose then that (v_1, v_2) is employed by a path $p = (..., v_1, v_2, ...) \in S_G$. Let t_j be the trail starting at v_j, $1 \leqslant j \leqslant k$. If p covers vertices that are not in C or t_1, let p' be the prefix of p covering those vertices. An equal-size cover of G, denoted by $\bar{S}_G$, is defined by modifying S_G to cover C, the vertices in t_1 and the vertices in p' as follows. The vertices in C are covered using the paths $p_1, ..., p_{(k+1)/2}$, where $p_i = t_{2i}^{-1} \parallel t_{2i+1}$ for $1 \leqslant i \leqslant (k+1)/2 - 1$, and $p_{(k+1)/2}$ covers t_1. If p' is not empty, then $p_{(k+1)/2}$ extends to cover the vertices in p'. The optimality of $\bar{S}_G$ follows from the following two facts. First, $\bar{S}_G$ covers C, the vertices in t_1 and the vertices in p' using $(k+1)/2$ paths, which is minimum since at least $(k+1)/2$ paths must be used to cover the k end-vertices of the trails t_j, $1 \leqslant j \leqslant k$. Second, $\bar{S}_G$ covers the rest of the vertices using the same number of paths as in S_G. Since $\bar{S}_G$ does not employ (v_1, v_2), it is also a cover of G'. ▌

Recall that no vertex of a 1-cactus belongs to more than one cycle. Thus, a 1-cactus containing an end-cycle must contain at least one end-cycle where one edge incident to the separator is leading to all other cycles (if such exist). In the following lemma we consider such end-cycles where the degree of the separator is three.

Lemma 5. Let $G=(V_G, E_G)$ be a graph. Let $C \Subset G$ be an odd end-cycle of order 3, where $v_1, v_2, \ldots, v_k$ are the vertices on C°, starting from the separator. If the degree of v_1 is three and the edge separating C from G is $e=(w, v_1)$, then $G'=(V_G, E'_G)$ where $E'_G = E_G - \{e\}$ satisfies $\pi(G')=\pi(G)$.

Proof. Clearly $\pi(G) \leqslant \pi(G')$. To prove the reverse inequality, we show that every optimal cover of G defines an equal-size cover of G'. Let S_G be an optimal cover of G. If e is not employed by S_G then we are done. Suppose then that e is employed by a path $p=(\ldots, w, v_1, \ldots) \in S_G$, where p' is the prefix of p terminating in w. An equal-size cover of G is defined by modifying S_G to cover C and the vertices in p' as follows. Let t_j be the trail starting at v_j, $2 \leqslant j \leqslant k$. The vertices in C are covered using the paths $p_1, \ldots, p_{(k-1)/2}$, where $p_1 = t_2{}^{-1} \,\|(v_1)\|\, t_k$ and, if $k>3$, $p_i = t_{2i-1}{}^{-1} \,\|\, t_{2i}$ for $2 \leqslant i \leqslant (k-1)/2$. Also, the path $p \in S_G$ is replaced by p'. The optimality of $\bar{S}_G$ follows from the following two facts. First, $\bar{S}_G$ covers C and the vertices in p' using $(k+1)/2$ paths, which is minimum since at least $(k+1)/2$ paths must be used to cover the vertices of p' and the $k-1$ end-vertices of the trails t_j, $2 \leqslant j \leqslant k$. Second, $\bar{S}_G$ covers the rest of the vertices using the same number of paths as in S_G. Since $\bar{S}_G$ does not employ (v_1, v_2), it is also a cover of G'. ▌

A basic, simple version of our covering algorithm is given next. A faster version is outlined in a later paragraph. Informally, the basic algorithm repeatedly applies Lemmas 1–5 to the input 1-cactus, G, transferring the isolated paths thus created to a set S_G. Eventually, S_G constitutes an optimal cover of G. At any given time during execution of the algorithm, the graph induced by the yet-uncovered vertices of G is denoted by G'. The graph G' is updated whenever any of Lemmas 1–5 is applied to it, or when a nonempty set of isolated paths is transferred to S_G. The latter task is performed by a procedure named *Transfer-Paths.*

Definitions. Let $G=(V_G, E_G)$ be a graph. By *applying* any of

Lemmas 1–5 to a vertex or an end-cycle in G we mean the deletion of the edge(s) considered in that lemma.

A vertex $v_1 \in V_G$ is a *fork* if the degree of v_1 is three or more, at least one trail $(v_1, v_2, \ldots, v_k)$ starts at v_1, and v_1 is adjacent to a vertex $w \neq v_2$ of degree one or two.

Basic Optimal Covering Algorithm for 1-Cacti.

Input: A 1-cactus $G = (V_G, E_G)$.
Output: A set of paths S_G, forming an optimal cover of G.
Method:

```
1. Initialize $S_G \leftarrow \varnothing$, $G' \leftarrow G$.
2. Transfer-Paths.
3. While $G'$ contains forks, do
         Choose a fork $v$.
         Apply Lemma 1 to $v$.
         Transfer-Paths.
   od
4. If $G'$ contains a subgraph $C$ which is either an isolated cycle or an
   end-cycle of order 2, do
         Apply Lemma 2 to $C$.
         Transfer-Paths.
         Go to step 3.
   od
5. If $G'$ contains an even end-cycle of order 3, $C$, do
         Apply Lemma 3 to $C$.
         Go to step 3.
   od
6. If $G'$ contains an odd end-cycle of order 3, $C$, where the separator is
   the start-point of a trail, do
         Apply Lemma 4 to $C$.
         Go to step 3.
   od
7. If $G'$ contains an odd end-cycle of order 3, $C$, where the degree of the
   separator is three, do
         Apply Lemma 5 to $C$.
         Transfer-Paths.
         Go to step 3.
   od
8. Stop.
```

The following argument establishes the correctness of the algorithm.

The reader can verify that, given a 1-cactus G which is not a set of isolated paths, at least one of the conditions tested in Steps 3–7 is satisfied. Whenever one of these conditions is satisfied, some edges are deleted from G. Since the isolated paths are removed from G (being transferred to S_G), eventually V_G is empty, and the algorithm terminates. Moreover, upon termination, S_G is a cover of the input 1-cactus. Since the algorithm deletes edges from G only by applying Lemmas 1–5, upon termination $|S_G| = \pi(G)$.

The basic algorithm can be implemented in $O(|V_G|^2)$ time. A DFS can be applied at each step to each connected component, in order to find a fork or to detect an end-cycle if no fork exists. Determining the type of an end-cycle C can be done by re-traversing each vertex on C in an order reversed to that of the DFS. Transferring the isolated paths to S_G can also be done using DFS. The time complexity of DFS is $O(|E_G|)$ and each application of the DFS results in a deletion of at least one edge, so the overall runtime of the algorithm is $O(|E_G|^2)$ time. Since G is a planar graph (being a 1-cactus), $E_G = O(|V_G|)$. Thus, the time complexity of the basic algorithm is $O(|V_G|^2)$.

Having described the basic algorithm, we next outline a linear time algorithm for optimal covering of 1-cacti. Each connected component in the input 1-cactus, G, is scanned using DFS. Immediately before backtracking from a vertex v the following rules are used:

1. If v is a fork, Lemma 1 is applied. The disconnected paths are transferred to S_G.
2. Suppose that a back-edge of the DFS enters v, i.e. v is on a cycle, R, in the input graph. By now, if no edge in R has yet been deleted, R underlies a crown. The DFS is temporarily suspended, and the remaining edges of R are re-traversed to determine if it is still a cycle. If some edge on R has been previously deleted, the vertices in R (and the trails starting on R, if any exist), are covered by repeatedly applying Lemma 1 and transferring isolated paths to S_G (in certain cases, there may not be a need to apply Lemma 1 at all). Otherwise, that is, if R now underlies a crown, H, an appropriate edge-deletion is performed according to the type of H. The vertices on H are then covered by repeatedly applying Lemma 1 to the remains of H and transferring isolated paths to S_G (in certain cases, there may not be a need to apply Lemma 1 at all). When this is done, the DFS is continued from v.

Figure 4 illustrates an execution of the latter algorithm. The numbers indicate the order in which the DFS scans the vertices. Thick edges belong to the DFS tree. At point I, when backtracking from vertex 5, Lemma 1 is applied to vertex 5 and the path (6, 5, 7, 8) is being transferred to the cover

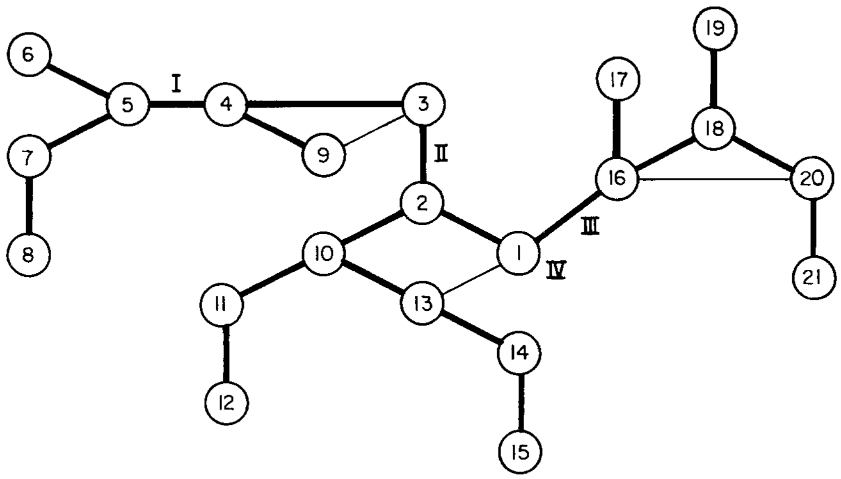

Fig. 4. An execution of the algorithm.

S_G. Lemma 2 is applied at point II to delete the edge (3, 9). At point III (backtracking from 16) Lemma 4 is applied to delete the edge (16, 18). Then Lemma 1 is applied to vertex 20 and we add the path (19, 18, 20, 21) to S_G. When the DFS is ended (point IV, back at vertex 1), Lemma 3 is applied to delete the edge (1, 13). Then Lemma 1 is applied to vertex 10 and the paths (12, 11, 10, 13, 14, 15) and (17, 16, 1, 2, 3, 4, 9) are being transferred to S_G.

The optimality of the cover obtained using this algorithm follows from Lemmas 1–5. The time complexity of this algorithm is inherited from that of the DFS, which is $O(|E_G|) = O(|V_G|)$. Thus, we have established the following:

Theorem 2. Given a 1-cactus $G = (V_G, E_G)$, an optimal cover of G can be found in $O(|V_G|)$ time.

Corollary 1. Given a 1-cactus $G = (V_G, E_G)$ and a line $H = (V_H, E_H)$ where $|V_G| \leqslant |V_H|$, an optimal (G, H)-augmentation can be found in $O(|V_G|)$ time.

Proof. Immediate, by Theorems 1 and 2. ▌

5. SUMMARY

We have shown that the augmenting number of a line-like NC with respect to a given program graph G is uniquely determined by the size of an optimal path cover of G. This gives rise to a "cover and augment" scheme for implementing parallel programs on computer networks.

We have also presented an algorithm for finding an optimal cover of 1-cacti. Being sparse, 1-cacti program graphs are natural candidates for implementation on line-like NCs, and this algorithm may come useful when augmenting a line to accommodate such programs.

ACKNOWLEDGMENT

The second author thanks Shlomo Moran for helpful discussions.

REFERENCES

1. J. R. Goodman and C. H. Sequin, Hypertree: A Multiprocessor Interconnection Topology, *IEEE Transactions on Computers*, **C-30**(12):923–933 (December 1981).
2. C. L. Seitz, The Cosmic Cube, *Communications of the ACM*, **28**(1):22–33 (January 1985).
3. C. H. Sequin and R. M. Fujimoto, X-Tree and Y-Components, in *VLSI Architecture*, (eds.), B. Randell and P. C. Trelevan, Prentice-Hall International, pp. 299–326 (1983).
4. S. H. Bokhari, On the Mapping Problem, *IEEE Transactions on Computers*, **C-30** (3):207–214 (March 1981).
5. F. Berman and L. Snyder, On Mapping Parallel Algorithms into Parallel Architectures, *Proceedings of the 13th International Conference on Parallel Processing*, pp. 307–309 (1984).
6. K. Efe, Heuristic Models of Task Assignment Scheduling in Distributed Systems, *Computer*, pp. 50–56 (June 1982).
7. J. P. Fishburn and R. A. Finkel, Quotient Networks, *IEEE Transactions on Computers*, **C-31** (4):288–295 (April 1982).
8. G. Kar, C. N. Nikolaou, and J. Reif, Assigning Processes to Processors: a Fault-Tolerant Approach, *Proceedings of the 14th International Conference on Fault-Tolerant Computing*, (June 1984).
9. H. S. Stone, Multiprocessor Scheduling with Aid of Network Flow Algorithms, *IEEE Transactions on Software Engineering*, **SE-3** (1):85–93 (January 1977).
10. S. B. Wu and M. T. Liu, Assignment of Tasks & Resources for Distributed Processing, *Proceedings-Distributed Computing, Compcon '80*, pp. 655–662.
11. W. W. Chu, L. J. Holloway, M. Lan, and K. Efe, Task Allocation in Distributed Data Processing, *Computer*, pp. 57–69 (November 1980).
12. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., Problems GT40, GT42 and ND43 (1979).
13. L. Snyder, Introduction to the Configurable, Highly Parallel Computer, *Computer*, pp. 47–56 (January 1982).
14. K. P. Eswaran and R. E. Tarjan, Augmentation Problems, *SIAM Journal on Computing*, **5**(4):653–665 (1975).
15. S. Bokhari and A. D. Raza, Reducing The Diameter of Computer Networks, *IEEE Transactions on Computers*, **C-35, 8**:757–761 (August 1986).
16. H. T. Kung, The Structure of Parallel Algorithms, in *Advances in Computers*, (ed.), M. C. Yovits, Academic Press, New York, **19**:73.
17. T. Gross, H. T. Kung, M. Lam, and J. Webb, Warp as a Machine for Low Level Vision, *Proceedings of the Conference on Robotics and Automation*, pp. 790–799 (March 1985).

18. A. V. Kularni and D. W. L. Yen, Systolic Processing and an Implementation for Signal and Image Processing, *IEEE Transactions on Computers*, **C-31**(10):1000–1009 (October 1982).
19. G. Baudet and D. Stevenson, Optimal Sorting Algorithms for Parallel Computers, *IEEE Transactions on Computers*, **C-27**(1):84–87 (January 1978).
20. H. T. Kung, Let's Design Algorithms for VLSI Systems, *Proceedings of the Caltech Conference on Very Large Scale Integration*, Pasadena, California (1979).
21. K. A. Doshi and P. J. Varman, Optimal Graph Algorithms on a Fixed-Size Linear Array, *IEEE Transactions on Computers*, **C-36**(4):460–470 (April 1987).
22. M. Garey, D. Johnson, and R. E. Tarjan, The Planar Hamiltonian Circuit Problem is NP-complete, *SIAM Journal on Computing*, 5:704–714 (1976).
23. F. T. Boesch, S. Chen, and J. A. M. McHugh, On Covering the Points of a Graph with Point Disjoint Paths, *Proc. 1973 Capital Conf. on Theory and Combinatorics*, Springer-Verlag, New York pp. 201–212 (1974).
24. F. T. Boesch, and J. F. Gimpel, Covering the Points of a Digraph with Point-Disjoint Paths and Its Application to Code Optimization, *Journal of the ACM*, **24**(2):192–198 (April 1977).