SIAM J. ALG. DISC. METH.
Vol. 8, No. 3, July 1987

AN APPLICATION OF GENERALIZED TREE PEBBLING TO SPARSE MATRIX FACTORIZATION*

JOSEPH W. H. LIU†

Abstract. A generalized version of the pebble game for trees is described. It is motivated by the study of out-of-core methods for the Cholesky factorization of sparse matrices. A solution to the generalized pebbling problem will give an equivalent ordering of the sparse matrix, so that the reordered matrix requires the minimum amount of in-core storage for its out-of-core factorization using the scheme in [12]. An efficient algorithm is presented to determine such an optimal solution.

Key words. sparse matrix, factorization, out-of-core, elimination tree, tree pebbling

AMS(MOS) subject classifications. 65F50, 65F25

1. Introduction. It is well known that many sparse matrix problems can be conveniently studied using graph-theoretic approaches. For example, the problem of reducing or minimizing bandwidth for a sparse symmetric matrix structure can be examined as a linear layout problem for graphs [4]. The fill-reduction ordering problem is closely related to the graph separator problem [10].

In this paper, we consider a problem encountered in the out-of-core solution of a sparse symmetric matrix. We want to find an equivalent ordering of a given sparse matrix, which will minimize the amount of in-core storage requirement for the successful execution of an out-of-core factorization scheme. We show that this sparse matrix problem can be transformed to a graph problem as a general form of the *pebble game* for rooted trees. This pebble game is originally introduced to study register allocation in straight-line programs [2]. It has received much attention on different variations of the basic problem [6]–[9], [14]–[16].

The generalized form of the game studied in this paper is quite different from the others in the literature. The number of pebbles required to satisfy a tree node can now be more than one. We provide an efficient algorithm to solve this generalized pebble game problem, and the underlying approach is similar to the one used by Yannakakis [20] to solve the related min-cut linear arrangement for trees. It should be noted that the algorithm can also be used to determine the best possible ordering for the out-of-core multifrontal method [3], [17] in terms of primary storage reduction.

An outline of this paper is as follows. In § 2, we describe briefly the necessary background on the sparse out-of-core factorization scheme introduced by the author in [12]. We formally introduce the class of equivalent orderings to be considered in the paper. It is based on the important tree structure, called the *elimination tree*, obtained from the sparse Cholesky factor matrix. The storage requirement on a fixed ordering for the out-of-core scheme is also derived.

In § 3, the problem of finding an optimal equivalent ordering that minimizes the primary storage requirement is transformed into the generalized pebble game problem.

* Received by the editors May 12, 1986; accepted for publication (in revised form) September 29, 1986. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A5509, by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc., and by the U.S. Air Force Office of Scientific Research under contract AFOSR-ISSA-85-00083.

† Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3.

A brief summary of existing pebbling algorithms to solve special forms of this pebble game is also presented.

Sections 4 to 6 are devoted to the development of an algorithm to solve the generalized pebble game. An overview of the method is given in § 4. The overall scheme makes use of the recursive structure of trees. It determines optimal orderings for subtrees, and then combines them to yield an optimal ordering for the entire tree.

Section 5 introduces the notion of a cost sequence. It is adapted from the one used by Yannakakis [20] on the min-cut layout problem for trees. This notion is essential in developing the overall optimal algorithm. Optimality is now in terms of this cost sequence together with a newly-defined partial order. In § 6, the algorithm to combine optimal subtree orderings is described. We prove that the overall ordering found is indeed optimal. The computational complexity of this algorithm is also addressed.

Section 7 contains our concluding remarks. There are three theorems in § 6, whose proofs are quite involved and lengthy. In order not to obscure the essential ideas in the paper, these proofs are postponed and presented in an appendix.

2. Statement of the problem.

2.1. Background on sparse out-of-core factorization. Let A be a given n by n sparse symmetric positive definite matrix, ordered appropriately by some fill-reducing ordering [5] (e.g., the minimum degree ordering). Let L be the (lower-triangular) Cholesky factor of A. The notations $\eta(L_{j*})$ and $\eta(L_{*j})$ are used to denote the number of nonzeros in the jth row and jth column of L, respectively.

In [12], the author proposes an out-of-core scheme for the sparse Cholesky factorization of large sparse matrices. The scheme is demonstrated to be quite effective in computing sparse Cholesky factors of extremely large matrices using auxiliary storage. It is based on the idea of matrix storage reorganization. A working storage vector in memory is provided to store nonzero entries of the Cholesky factor L. We shall refer to it as the "primary storage vector" for L.

If this primary storage can accommodate all nonzeros in the factor L, factorization will be carried out by the conventional in-core method [5]. Otherwise, this storage vector will be reorganized when need arises during the course of factorization. In each organization, only those values that are required for subsequent steps of factorization are to be retained in memory. In this way, it allows much larger problems to be solved in a given amount of primary storage, without having to rely on excessive data I/O to and from auxiliary storage. Indeed, auxiliary storage is used only to store the computed columns of the Cholesky factor.

In this out-of-core algorithm, the minimum amount of primary storage required during the computation of the jth column of L is given by

$$\sum_{k=1}^{j-1} \{\eta(L_{*k}) - \eta(L_{k*})\} + \eta(L_{*j}).$$

It is easy to see that this is actually the number of nonzero entries in the set

$$L_{[j]} = \{l_{ik} | k \leqq j \leqq i\}.$$

This rectangular window is the shaded region as illustrated in Fig. 2.1. We shall use $\eta(L_{[j]})$ to denote the number of nonzeros in this region.

Therefore, the minimum primary storage requirement for the successful completion of the entire factorization using the out-of-core scheme is

$$\max \{\eta(L_{[j]}) | 1 \leqq j \leqq n\}.$$

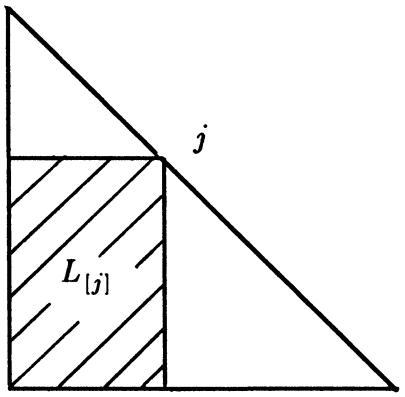

FIG. 2.1. *Region required for the computation of column j of L.*

Note that this quantity is fixed once the structure of the matrix is specified and its ordering is given.

2.2. The problem: Primary storage minimization. For a given fill-reducing ordering, it is well known that there exists a class of orderings that are equivalent in terms of fills and operations. It is based on the so-called *elimination tree* structure. This tree structure defines a class of equivalent orderings, each having the same set of filled edges as the original ordering [13], [18].

Consider the structure of the Cholesky factor L. We define the elimination tree of A to be the tree with n nodes $\{1, 2, \cdots, n\}$, and node i is the parent of node j if and only if

$$i = \min \{k | l_{kj} \neq 0\},$$

that is, i is the row subscript of the first off-diagonal nonzero in column j of L. We assume that the matrix A is *irreducible*, so that the structure is indeed a tree, and n is the root of this tree. (If A is reducible, then the elimination tree defined above is actually a forest which consists of several trees.) Figure 2.2 contains a 10-by-10 matrix example whose

$$A = \begin{pmatrix}
1 & & x & & & & & & & x \\
 & 2 & x & & & & & & & x \\
x & x & 3 & & & & & & & x \\
 & & & 4 & & x & x & & & \\
 & & & & 5 & x & x & & & \\
 & & & x & x & 6 & x & & & \\
 & & & x & x & x & 7 & x & & x \\
 & & & & & & x & 8 & x & x \\
 & & & & & & & x & 9 & x \\
x & x & x & & & & x & x & x & 10
\end{pmatrix}$$

FIG. 2.2. *A matrix example.*

diagonal entries are labeled by the corresponding equation/variable numbers. Its elimination tree is displayed in Fig. 2.3.

Any reordering that numbers nodes before parent nodes in the elimination tree is known to be equivalent to the original ordering. In other words, the number of fills and the amount of arithmetic operations to perform the factorization remain unchanged. Such orderings are referred to as *topological* orderings of the tree [19]. In this paper, we consider the problem of determining a topological ordering for a given elimination tree that will *minimize* the primary storage requirement for the out-of-core algorithm in [12].

We first re-specify the problem in graph-theoretic terms. Let $T = (X, E)$ be a given rooted tree of n nodes. For each node $x \in X$ in the tree T, two integer values are associated with it: row(x) and col(x). For any topological ordering π: $x_1, x_2, \cdots, x_n$, the *core cost* at x_j is defined to be

$$\text{core}_\pi(x_j) = \sum_{k=1}^{j-1} \{\text{col}(x_k) - \text{row}(x_k)\} + \text{col}(x_j).$$

The core cost of T with respect to the given ordering π is then

$$\max \{\text{core}_\pi(x_j) | 1 \leqq j \leqq n\}.$$

Our objective here is to determine an optimal ordering π that will minimize the core cost of T over all topological orderings of T.

In this paper, only topological orderings with respect to an elimination tree will be considered. Unless otherwise stated, we shall use ordering of a tree to refer to a topological ordering, that is, one that numbers nodes before parent nodes.

3. On generalized pebbling.

3.1. Problem transformation. In this section, we transform the problem in § 2 to a generalized form of a much-studied combinatorial problem: the *pebble game* for trees [8], [14], [16]. The game can be used as a model for register allocation in straight-line programs.

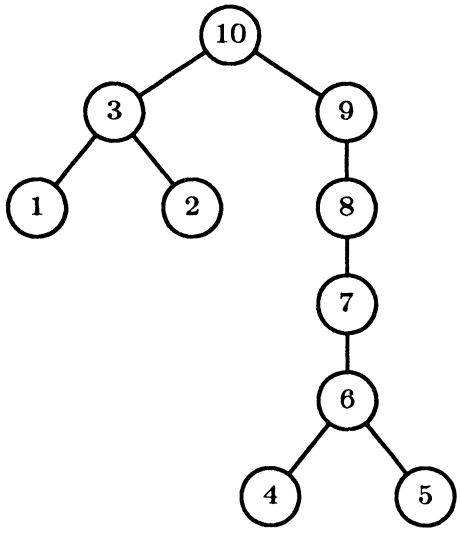

FIG. 2.3. *The elimination tree of matrix in Fig.* 2.2.

Consider a given tree T and the values row(x) and col(x) associated with each node x of T. For any (topological) ordering π: $x_1, x_2, \cdots, x_n$, the core cost as defined in the previous section can be expressed recursively as follows:

$$\text{core}_\pi(x_1) = \text{col}(x_1),$$

and for $j > 1$,

$$\text{core}_\pi(x_j) = \{\text{core}_\pi(x_{j-1}) - \text{row}(x_{j-1})\} + \text{col}(x_j).$$

It should be clear that a portion of the value $\text{core}_\pi(x_j)$ comes from nodes in the subtree rooted at x_j. This contribution from the subtree is independent of the ordering π, since nodes in the subtree under x_j are always ordered before x_j.

To aid the study of this problem, we let $T[x]$ denote the subtree of T rooted at a node x. It is also convenient to expand each original node x of T into two nodes x^+ and x^- as shown in Fig. 3.1. The node x^+ can be regarded as x during the processing of its column, with x^- as x after its processing.

Then, we can associate with each node x the two quantities:

$$\tau(x^-) = \sum_{z \in T[x]} \{\text{col}(z) - \text{row}(z)\}, \qquad \tau(x^+) = \tau(x^-) + \text{row}(x).$$

The value $\tau(x^+)$ represents the number of nonzeros in columns of L from the subtree $T[x]$, that are required *during* the processing of the column x in the factorization. On the other hand, $\tau(x^-)$ is the number of nonzeros in columns of L associated with $T[x]$ that are still required *after* the processing of x. They are the storage requirements contributed from the nodes in the subtree $T[x]$. Note that these two values depend only on the structure of the tree T, and are independent of any topological ordering.

We can now express the core cost in terms of $\tau(x^+)$ and $\tau(x^-)$. The formulation will be clearer if we introduce $\text{core}_\pi(x_j^+)$ and $\text{core}_\pi(x_j^-)$, which are the storage requirements during and after the processing of column x_j, respectively. Let

$$\text{core}_\pi(x_0^-) = 0.$$

For $j \geqq 1$, then we have

$$\text{core}_\pi(x_j^+) = \text{core}_\pi(x_{j-1}^-) + \tau(x_j^+) - \sum \{\tau(x_c^-) | x_c \text{ is a child of } x_j\},$$

$$\text{core}_\pi(x_j^-) = \text{core}_\pi(x_j^+) + \tau(x_j^-) - \tau(x_j^+).$$

This formulation actually provides a more uniform framework to study the problem. Consider the transformed tree with the $2n$ number of nodes

$$\{x_1^+, x_1^-, x_2^+, x_2^-, \cdots, x_n^+, x_n^-\}.$$

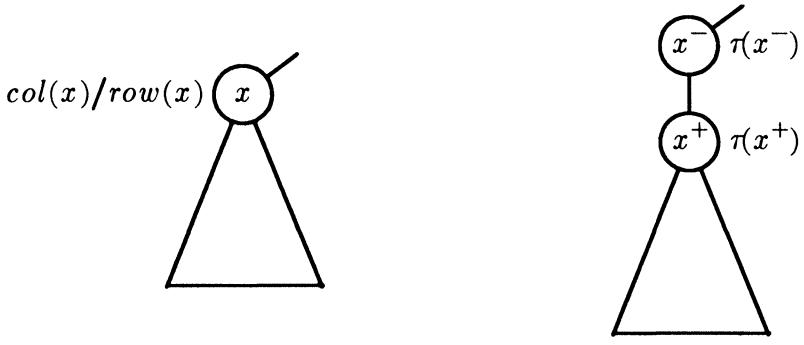

FIG. 3.1. *Tree transformation.*

Rename these nodes to $\{y_1, y_2, \cdots, y_{2n}\}$. If $\{x_j\}$ is a topological ordering on the original tree, it is clear that $\{y_j\}$ is also a topological ordering on the transformed tree. For each node, associate a τ value as follows: for $1 \leqq j \leqq n$,

$$\tau(y_{2j-1}) = \tau(x_j^+), \qquad \tau(y_{2j}) = \tau(x_j^-).$$

The transformed elimination tree of the example in Fig. 2.3 is given in Fig. 3.2. The labels in the original tree should be interpreted as "col(x)/row(x)," while that in the transformed tree are the corresponding τ-values. This tree structure will be used repeatedly throughout the paper.

Since x_j^+ is the only child node of x_j^-, the core cost in terms of the transformed tree can be collectively and conveniently expressed as:

$$\text{core}_\pi(y_0) = 0,$$

$$\text{core}_\pi(y_j) = \text{core}_\pi(y_{j-1}) + \tau(y_j) - \sum \{\tau(y_c) | y_c \text{ is a child of } y_j\}$$

for $1 \leqq j \leqq 2n$. It should be clear that an optimal topological ordering on the transformed tree in terms of the core cost will induce one on the original tree. Henceforth, we shall discard the values row(x) and col(x). Instead, we assume that a nonnegative value $\tau(y)$ is associated with each node y and $\text{core}_\pi(y)$ is defined as above in terms of $\tau(y)$.

3.2. The generalized pebble game. The transformed problem in § 3.1 can be formulated as a generalized version of the pebble game. Let T be a given rooted tree of m

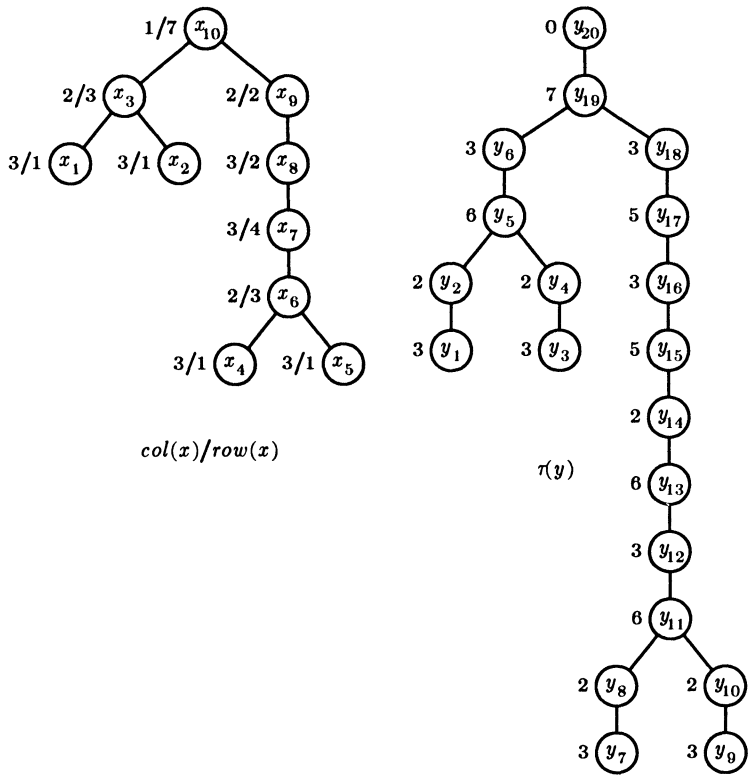

FIG. 3.2. *The transformed tree of the example in Fig.* 2.2.

nodes. For each node y in T, there is a nonnegative value $\tau(y)$ associated with it. The number $\tau(y)$ represents the number of pebbles required to satisfy the node y (a node y is said to be satisfied if there are $\tau(y)$ pebbles in this node). The generalized pebble game is played according to the following rules:

(a) If all children of an unpebbled node y are satisfied, pebbles may be placed on y (thus, a leaf node can be pebbled).

(b) If all children on an unpebbled node y are satisfied, pebbles may be moved from its children nodes to y.

(c) A pebble may be removed from a node y if there are more than $\tau(y)$ pebbles in it.

The goal of the game is, starting with no pebbles in the tree, to pebble the root of the given tree. The pebbling proceeds in moves, each move is an application of one of the above rules. The problem here is to determine a sequence of moves that will achieve the goal using the minimum number of pebbles. The sequence of moves will simply correspond to a topological ordering on the given tree.

Note that the standard (black) pebble game [8], [14] is the special case with $\tau(y) = 1$ for all nodes y in the tree. It should also be clear to the reader that the optimal solution to this generalized pebble game will be an optimal one for the primary storage minimization problem of the previous section.

3.3. Existing pebbling algorithms. The original pebble game is the special case with all pebble values $\tau(y)$ equal to 1. The solution for this standard problem can be found in [8], [14]. It is helpful to compare this scheme with the general algorithm provided later, we describe the method below. The description follows that in [8].

For a given rooted tree T with $\tau(y) = 1$ for every node y, let $p(T)$ be the minimum number of pebbles required to pebble the root. If T has only one node (the root), obviously we have $p(T) = 1$. Otherwise, assume that the root has t children nodes, and let $T_1, \cdots, T_t$ be the subtrees under the root. Then

$$p(T) = \max_{1 \leqq k \leqq t} \{p(T_k) + k - 1\},$$

where the subtrees are ordered such that

$$p(T_1) \geqq \cdots \geqq p(T_t).$$

This observation will give an algorithm that computes the value $p(T)$ and at the same time determines an (topological) ordering that achieves this minimum value. It is interesting to point out that the ordering determined by this algorithm will always number nodes within any subtree of T consecutively.

In [11], the author considers the primary storage minimization of the out-of-core multifrontal method due to Duff and Reid [3], [17]. That problem can be formulated again as a tree pebble game, where the values $\tau(y)$ can now be greater than one. However, due to the nature of the multifrontal method, *postorderings* are to be considered, that is, subtree nodes should be ordered consecutively [1]. This, therefore, may be regarded as the generalized pebble game as described in § 3.2, except for the more restrictive nature of the move sequence (postorderings). A solution to this problem is also provided in [11]. We include a brief description here for future comparison.

For a given rooted tree T with $\tau(*)$ values, let $\tilde{p}(T)$ be the minimum number of pebbles required to pebble the root, subject to the restriction that subtree nodes are to be pebbled consecutively. Assume that y is the root of T with t children: $s_1, \cdots, s_t$. Let $T_1, \cdots, T_t$ be the subtrees rooted at these children nodes.

If T has only one node y (that is, $t = 0$), $\tilde{p}(T) = \tau(y)$. Otherwise, we have

$$\tilde{p}(T) = \max\left\{\max_{1 \leqq k \leqq t}\left\{\tilde{p}(T_k) + \sum_{j=1}^{k-1} \tau(s_j)\right\}, \tau(y)\right\},$$

where the subtrees are ordered such that

$$\tilde{p}(T_1) - \tau(s_1) \geqq \cdots \geqq \tilde{p}(T_t) - \tau(s_t).$$

An algorithm, based on this observation, can be formulated to compute the value $\tilde{p}(T)$ and to determine a postordering that achieves this minimum value.

It is interesting to point out that if postorderings are not required in the out-of-core multifrontal method, the problem becomes more involved. Indeed, the algorithm to be developed in this paper will be applicable in such setting. It will give the best possible topological ordering (not necessarily postordering) so that the ordered matrix will require the least amount of primary storage in its out-of-core multifrontal factorization. For clarity, the author will focus only on the use of the ordering algorithm for the out-of-core factorization method described in § 2. Its use in the context of multifrontal method will be left to the reader.

4. Overview of strategy for optimal ordering. Given a rooted tree T of m nodes, each node y having a pebble value $\tau(y)$. Our objective is to determine a topological ordering of the tree so that the pebble game following this ordering requires the least number of pebbles.

For any (topological) ordering π: $y_1, y_2, \cdots, y_m$, define the sequence of values peb(∗):

$$\text{peb}_\pi(y_0) = 0,$$

$$\text{peb}_\pi(y_j) = \text{peb}_\pi(y_{j-1}) + \tau(y_j) - \sum\{\tau(y_c) | y_c \text{ is a child of } y_j\},$$

for $1 \leqq j \leqq m$. The value $\text{peb}_\pi(y_j)$ represents the total number of pebbles used during the pebbling of the node y_j; it may be appropriately called the *accumulated pebble value* at the node y_j using ordering π. The number of pebbles required to pebble the entire tree T using this ordering is given by:

$$\text{peb}_\pi(T) = \max\{\text{peb}_\pi(y_j) | 1 \leqq j \leqq m\}.$$

In other words, our objective is to find one such topological ordering that will minimize this pebble requirement $\text{peb}_\pi(T)$.

The recursive structure of trees can often be used to design efficient algorithms to solve problems on rooted trees. The approach is to proceed bottom up in the rooted tree. For every node y with children nodes $s_1, \cdots, s_t$, solutions are determined for all the subtrees rooted at s_k $(1 \leqq k \leqq t)$. These solutions are then combined to produce one for the subtree rooted at y. A recursive use of this will solve the given problem on the overall rooted tree. Solutions to our pebbling problem are topological orderings that minimize the number of pebbles. We shall use this bottom up approach to combine optimal subtree orderings.

Let us first introduce some relevant terminology for tree orderings. Consider any rooted subtree of T, say $T[y]$, rooted at the node y. Let π be an ordering on T. The restriction of this ordering π on $T[y]$ is itself an ordering for this subtree. We shall denote this subtree ordering by $\pi[y]$ and refer to it as the *induced ordering* of π on $T[y]$. On the other hand, let ψ be an ordering on the subtree $T[y]$. ψ is said to be *compatible* with π if ψ is the same as the induced ordering $\pi[y]$.

For example, in Fig. 3.2, the induced ordering $\pi[y_{11}]$ on the subtree $T[y_{11}]$ is given by the node sequence:

$$y_7, y_8, y_9, y_{10}, y_{11}.$$

However, the following ordering on $T[y_{11}]$:

$$y_9, y_7, y_{10}, y_8, y_{11}$$

is not compatible with the original tree ordering.

Using this bottom up approach to our pebble minimization problem, we can describe our strategy as follows. Here, y is the input node with children nodes $s_1, \cdots, s_t$; and π is the returned optimal ordering for the tree $T[y]$ rooted at y.

ALGORITHM 4.1. Pebble-Ordering ($T[y]$, π).

```
begin
  If t = 0 then
    return the sequence π: y
  else
    begin
      For k := 1 to t do
        Pebble-Ordering (T[s_k], ψ_k);
      Combine the optimal subtree orderings ψ_k, k = 1, ··· , t
          to give an optimal ordering π for T[y] such that
          π is compatible with each ψ_k;
    end;
end.
```

Therefore, a strategy for optimal ordering can be obtained if we can provide an efficient solution to the *one-level* problem: combining optimal orderings of subtrees to form one for the tree. Each subtree ordering ψ_k is optimal, that is, it minimizes the value of $\text{peb}_{\psi_k}(T[s_k])$. However, this condition is not sufficient to guarantee the existence of an optimal ordering for $T[y]$ compatible with each one of the subtree orderings ψ_k.

A simple example is provided in Fig. 4.1 to illustrate this point. The ordering z_1, z_2, z_3, z_4, z_5, z_6 minimizes the pebble cost of 10 on the subtree $T[z_6]$. It is easy to verify that for all orderings on the entire tree compatible with this subtree ordering on $T[z_6]$, the pebble cost will be *at least* 14. Yet, the following ordering

$$z_1, z_2, z_4, z_5, z_7, z_8, z_3, z_6, z_9$$

will have a pebble cost of only 10.

In the next section, we introduce a new criterion for optimal orderings. We shall show that with this more involved criterion, there *always* exists an optimal compatible ordering for $T[y]$.

5. Pebble cost sequence and partial order.

5.1. Definition of pebble cost sequence. Let T be a given rooted tree of m nodes. Our objective is to find an optimal ordering $\bar{\pi}$ that minimizes the overall pebble cost in the generalized pebble game:

$$\text{peb}_{\bar{\pi}}(T) = \min \{\text{peb}_{\pi}(T) | \pi \text{ is a topological ordering}\}.$$

As noted in § 4, it is not sufficient to combine subtree orderings that minimize only the pebble costs of the subtrees. We need a more elaborate pebble cost function. This function

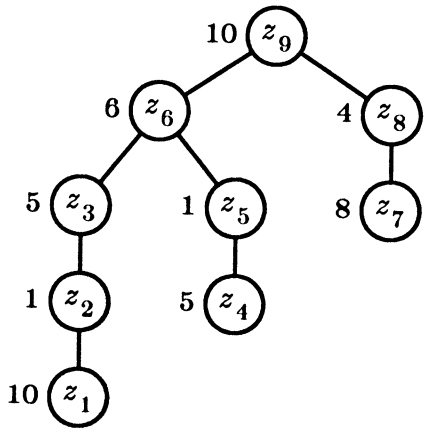

FIG. 4.1. *Example to show compatible subtree ordering.*

is adapted from the one used by Yannakakis [20] in his polynomial algorithm for the related problem of min-cut linear arrangement for trees.

Consider a topological ordering for the given tree T:

$$\pi: y_1, y_2, \cdots, y_m.$$

This defines the following sequence of values:

$$\text{peb}_\pi(y_1), \text{peb}_\pi(y_2), \cdots, \text{peb}_\pi(y_m).$$

We now introduce the *pebble cost* sequence/function. Put $v_0 = 0$. Let h_1 be the *largest* subscript of the y's such that

$$H_1 = \text{peb}_\pi(y_{h_1}) = \max \{\text{peb}_\pi(y_j) | v_0 < j \leqq m\},$$

and v_1 be the *largest* subscript such that

$$V_1 = \text{peb}_\pi(y_{v_1}) = \min \{\text{peb}_\pi(y_j) | h_1 \leqq j \leqq m\}.$$

We then define recursively h_i, v_i, and H_i, V_i as follows: h_i is the largest subscript where the *maximum* pebble cost value H_i occurs from v_{i-1} to m, and v_i the largest subscript where the *minimum* pebble cost value V_i occurs from h_i to m. Thus, we have a cost sequence, denoted by Pcost(T, π):

$$(H_1, V_1, H_2, V_2, \cdots, H_r, V_r)$$

and these values occur at the following sequence of nodes:

$$y_{h_1}, y_{v_1}, y_{h_2}, y_{v_2}, \cdots, y_{h_r}, y_{v_r}.$$

Since the tree is rooted at y_m, the last value V_r must occur at this node, that is, $v_r = m$ or $y_{v_r} = y_m$. Note also that the value of r depends on the tree structure, the pebble values and the ordering.

To illustrate the notion of this cost sequence, we consider the example in Fig. 3.2. It is clear that the pebble cost sequence for the tree is given by:

$$\text{Pcost}(T, \pi) = (9, 0),$$

and they occur at the nodes:

$$(y_{13}, y_{20}).$$

However, if we only consider the subtree $T[y_{18}]$ in this example with the induced ordering $\pi[y_{18}]$, the pebble cost sequence is then:

$$\text{Pcost}(T[y_{18}], \pi[y_{18}]) = (6, 2, 5, 3),$$

and these values occur at the nodes:

$$(y_{13}, y_{14}, y_{17}, y_{18}).$$

Note that the pebble requirement in the subtree $T[y_6]$ does not affect the pebble sequence for $T[y_{18}]$.

We shall sometimes refer to the locations y_{h_i} as the *hills* and y_{v_i} as the *valleys* of the given tree and ordering. The quantities H_i and V_i are also referred to as the *hill* and *valley* values, respectively. The motivation for the choice of these terminologies should be clear from the plot of accumulated pebble cost values $\text{peb}_\pi(y_j)$ against y_j. The plot for the subtree $T[y_{18}]$ of Fig. 3.2 is illustrated in Fig. 5.1.

Let $\text{Pcost}(T, \pi) = (H_1, V_1, \cdots, H_r, V_r)$ be a cost sequence. It is clear from definition that

$$H_1 = \text{peb}_\pi(T).$$

The following property is also obvious.

LEMMA 5.1. $H_1 > H_2 > \cdots > H_r \geqq V_r > \cdots > V_1 > V_0 = 0$.

5.2. A partial order for pebble cost sequences. We want to compare different topological orderings on a rooted tree with respect to their pebble cost sequences. To prepare for that, we introduce a *partial order* on these sequences. Let α and β be two pebble cost sequences:

$$\alpha = (\tilde{H}_1, \tilde{V}_1, \cdots, \tilde{H}_{\tilde{r}}, \tilde{V}_{\tilde{r}}), \qquad \beta = (H_1, V_1, \cdots, H_r, V_r).$$

We say that $\alpha \prec \beta$ if and only if for every i $(1 \leqq i \leqq \tilde{r})$, there exists a j $(1 \leqq j \leqq r)$ such that

$$\tilde{H}_i \leqq H_j \quad \text{and} \quad \tilde{V}_i \leqq V_j.$$

THEOREM 5.2. "$\prec$" *is a partial order on cost sequences.*

Proof. It is obvious that "$\prec$" is transitive and reflexive. It remains to show that it is anti-symmetric. Let

$$\alpha = (\tilde{H}_1, \tilde{V}_1, \cdots, \tilde{H}_{\tilde{r}}, \tilde{V}_{\tilde{r}}),$$
$$\beta = (H_1, V_1, \cdots, H_r, V_r).$$

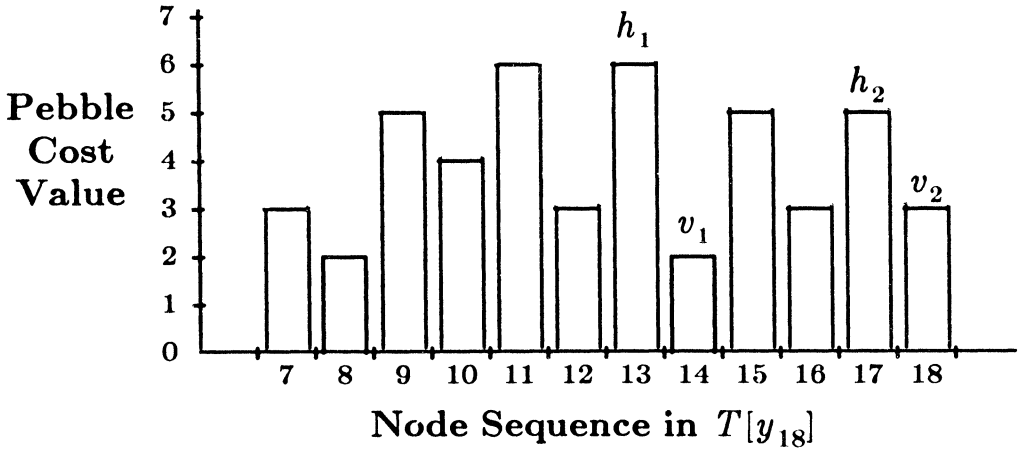

FIG. 5.1. *Plot of pebble cost for subtree* $T[y_{18}]$ *in Fig.* 3.2.

Assume that $\alpha \prec \beta$ and $\beta \prec \alpha$. Consider any $\tilde{H}_i$ and $\tilde{V}_i$. By the definition of "$\prec$", there exists a j such that $\tilde{H}_i \leqq H_j$ and $\tilde{V}_i \leqq V_j$. Since $\beta \prec \alpha$, for this j, there is a k such that $H_j \leqq \tilde{H}_k$ and $V_j \leqq \tilde{V}_k$. Combining we have $\tilde{H}_i \leqq \tilde{H}_k$ and $\tilde{V}_i \leqq \tilde{V}_k$ so that by Lemma 5.1, we must have $i = k$. This implies that $\tilde{H}_i = H_j$ and $\tilde{V}_i = V_j$.

It remains to show that for every $1 \leqq i \leqq \tilde{r}$,

$$\tilde{H}_i = H_i \quad \text{and} \quad \tilde{V}_i = V_i.$$

We prove this by induction on i. For $i = 1$, by the property established above, there exists a j such that $\tilde{H}_1 = H_j$. Assume for contradiction that $j \neq 1$, so that by Lemma 5.1 $H_j < H_1$. Then since $\beta \prec \alpha$, for H_1, there must be a k such that $H_1 \leqq \tilde{H}_k$. Combining, we have

$$\tilde{H}_1 \leqq H_j < H_1 \leqq \tilde{H}_k.$$

This contradicts Lemma 5.1 on the cost sequence α. Therefore, $\tilde{H}_1 = H_1$ (so that $\tilde{V}_1 = V_1$).

The same argument can be used for the inductive step. Therefore the sequence α must be an initial subsequence of β. By symmetry, β must also be an initial subsequence of α. Hence, α and β must be identical cost sequences. □

The next theorem follows directly from definition. It shows the relevance of the pebble cost sequence and the partial order "$\prec$" in the context of pebble minimization.

THEOREM 5.3. *For two orderings ψ and π of the tree T, if*

$$\text{Pcost}(T, \psi) \prec \text{Pcost}(T, \pi)$$

then $\text{peb}_\psi(T) \leqq \text{peb}_\pi(T)$.

The implication of this simple observation is that in order to determine an optimal ordering that minimizes the overall pebble requirement, we can restrict our search for an ordering $\bar{\pi}$ (if it exists) such that

$$\text{Pcost}(T, \bar{\pi}) \prec \text{Pcost}(T, \pi)$$

for all orderings π.

6. Combining subtree orderings.

6.1. Combine algorithm based on subtree segments. In this section, we show how to solve the *one-level* problem: combining optimal subtree orderings to give an optimal ordering for the overall tree. Here, optimality is with reference to the pebble cost sequence and the partial order "$\prec$" introduced in the last section.

Let T be a given tree rooted at the node y, and $s_1, s_2, \cdots, s_t$ be the children nodes of y. Assume that $\psi_1, \cdots, \psi_t$ are given orderings on the respective subtrees $T[s_1], \cdots, T[s_t]$.

We want to construct an optimal ordering for T which is compatible with each subtree ordering. Obviously, the last node in this ordering must be y, the root. The problem is how to interleave nodes from the t subtrees under y so that the resulting pebble cost sequence is minimized. The idea is quite simple: for each hill value in a subtree, we should try to use appropriately-chosen valley values for the remaining subtrees. This will help to reduce the impact of the hill value on the pebble cost sequence.

To facilitate the discussion, we introduce the notion of *valley segments* for an ordered tree. Consider a subtree $T[y]$ with an ordering ψ. Let its pebble cost sequence be:

$$\text{Pcost}(T[y], \psi) = (H_1, V_1, \cdots, H_r, V_r),$$

and let these values occur at the nodes

$$y_{h_1}, y_{v_1}, \cdots, y_{h_r}, y_{v_r}.$$

There are r valley segments of $T[y]$; for $1 \leqq k \leqq r$, the kth valley segment consists of the nodes

$$y_{v_{k-1}+1}, y_{v_{k-1}+2}, \cdots, y_{v_k}.$$

In other words, it is the sequence of nodes in between two valley nodes (including y_{v_k} but not $y_{v_{k-1}}$). We shall define its *segment value* to be $H_k - V_k$.

For example, consider the subtree $T[y_{18}]$ in Fig. 3.2. There are two valley segments:

$$y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, \qquad y_{15}, y_{16}, y_{17}, y_{18}$$

and their segment values are 4 and 2, respectively. But the subtree $T[y_6]$ has only one segment:

$$y_1, y_2, y_3, y_4, y_5, y_6$$

which is the entire subtree, and its segment value is 3.

Valley nodes are appropriate locations to switch from one subtree to another when combining subtree orderings. Valley segments are relevant notions, and nodes within each segment can be treated as an entity. Indeed, the following algorithm combines the given subtree orderings based on an arrangement of the segments in all subtrees. As before, ψ_k is a subtree ordering of $T[s_k]$, where $s_1, \cdots, s_t$ are children nodes of the root y in the tree T.

ALGORITHM 6.1. Combine $(T[y], \psi)$

```
begin
  For k := 1 to t do
    Determine the valley segments of the subtree T[s_k]
      using the cost sequence Pcost(T[s_k], ψ_k);
  Arrange the segments from all the subtrees in descending order of their
    segment values: {hill value-valley value};
  Based on this segment arrangement, order the nodes in each segment
    consecutively, followed by the root y;
  Return this ordering as ψ
end.
```

We shall use the notation $\Phi(\psi_1, \psi_2, \cdots, \psi_t)$ to refer to the ordering ψ on $T[y]$ obtained by Algorithm 6.1. When $t = 1$, this ordering can be obtained simply by appending the root y to the subtree ordering ψ_1 of its only subtree.

It is easy to see that the ordering obtained by Algorithm 6.1 is compatible with each subtree ordering ψ_k. Indeed, the segments within each subtree are already in descending sequence with respect to their segment values (it follows from Lemma 5.1). This means the relative order of nodes in each subtree is always preserved by the new ordering.

On applying Algorithm 6.1 to the subtree $T[y_{19}]$ of the example in Fig. 3.1, we note that the root y_{19} has two children nodes y_6 and y_{18}. The subtree $T[y_6]$ has one segment of value 3; while the subtree $T[y_{18}]$ has two segments of value 4 and 2, respectively. Therefore the ordering returned by Algorithm 6.1 will be the nodes in the segment (with value 4):

$$y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14},$$

followed by the segment (with value 3):

$$y_1, y_2, y_3, y_4, y_5, y_6,$$

then by (with value 2):

$$y_{15}, y_{16}, y_{17}, y_{18},$$

and finally by the node y_{19}. With this new ordering, the pebble cost sequence for T is reduced from (9, 0) to (8, 0).

6.2. Properties of the combine algorithm. We shall state some important properties of the ordering $\Phi(\psi_1, \cdots, \psi_t)$ obtained from Algorithm 6.1. The detailed proofs are lengthy, and we shall provide them at the end of the paper in the Appendix. The following sequence of theorems is to establish the optimality of the "Combine" algorithm when used recursively in the "Pebble-Ordering" algorithm of § 4.

THEOREM 6.1. *Let $\psi = \Phi(\psi_1, \cdots, \psi_t)$. For any ordering π' that orders nodes within each subtree segment consecutively and is compatible with each ψ_k,*

$$\text{Pcost}(T, \psi) \prec \text{Pcost}(T, \pi').$$

THEOREM 6.2. *Let π be any topological ordering on the tree $T[y]$, which is compatible with each subtree ordering ψ_k. There exists an ordering π' on $T[y]$, that orders nodes in subtree segments consecutively, such that*

$$\text{Pcost}(T, \pi') \prec \text{Pcost}(T, \pi).$$

THEOREM 6.3. *Let $\bar{\psi}_k$ be another subtree ordering for $T[s_k]$, where*

$$\text{Pcost}(T[s_k], \bar{\psi}_k) \prec \text{Pcost}(T[s_k], \psi_k).$$

If $\bar{\pi} = \Phi(\psi_1, \cdots, \bar{\psi}_k, \cdots, \psi_t)$, and $\psi = \Phi(\psi_1, \cdots, \psi_k, \cdots, \psi_t)$, then

$$\text{Pcost}(T, \bar{\pi}) \prec \text{Pcost}(T, \psi).$$

The proofs of Theorems 6.1–6.3 are left to the Appendix. Theorem 6.1 says that the cost sequence returned from Algorithm 6.1 is the smallest possible (in terms of "$\prec$") among all orderings that are based on the valley segments. Theorem 6.2 implies that if it is the smallest among segment-based orderings, it will also be the smallest among all orderings compatible with the individual subtree orderings. Finally, Theorem 6.3 points out the effect of an improved subtree ordering on the combined ordering Φ. We can now use these results to establish the optimality of our overall ordering algorithm.

THEOREM 6.4. *Let $\bar{\pi}$ be the ordering on $T[y]$ returned from Algorithm* 4.1 (*"Pebble-Ordering"*), *where subtree orderings are combined by Algorithm* 6.1 (*"Combine"*). *Then for any topological ordering π of $T[y]$,*

$$\text{Pcost}(T, \bar{\pi}) \prec \text{Pcost}(T, \pi).$$

Proof. We prove the result by induction on the number m of nodes in the tree $T[y]$. The result is obviously true if $m = 1$. Assume that the result is true for all trees with less than m modes. Let the children nodes of y be $s_1, \cdots, s_t$; and $\bar{\psi}_k$ be the ordering obtained from the execution of "Pebble-Ordering ($T[s_k]$, $\bar{\psi}_k$)." So $\bar{\pi}$ can be expressed as $\Phi(\bar{\psi}_1, \cdots, \bar{\psi}_t)$.

Consider any ordering π of $T[y]$, and their induced subtree orderings $\pi[s_k]$, for $1 \leqq k \leqq t$. Let π' be the ordering $\Phi(\pi[s_1], \cdots, \pi[s_t])$. By Theorems 6.1 and 6.2, π' has the best pebble cost sequence relative to all orderings compatible with each subtree ordering $\pi[s_k]$. In other words,

$$\text{Pcost}(T, \pi') \prec \text{Pcost}(T, \pi).$$

But, by the inductive assumption, in each subtree $T[s_k]$,

$$\text{Pcost}(T[s_k], \bar{\psi}_k) \prec \text{Pcost}(T[s_k], \pi[s_k]).$$

A repeated application of Theorem 6.3 and the transitive property of the partial order "$\prec$" (Theorem 5.2) will give

$$\text{Pcost}(T, \bar{\pi}) \prec \text{Pcost}(T, \pi').$$

Therefore, the result follows. □

Theorem 6.4 shows that Algorithms 4.1 and 6.1 can be used to yield a topological ordering $\bar{\pi}$ that minimizes the pebble cost sequence Pcost(T, $\bar{\pi}$) and hence the pebble cost value $\text{peb}_{\bar{\pi}}(T)$. We now determine the time complexity of this algorithm. We show that Algorithm 4.1 (Pebble-Ordering) and Algorithm 6.1 (Combine) can be implemented in time $O(m^2)$, where m is the number of nodes in the tree.

Assume that the given tree $T[y]$ is rooted at y with m nodes, and the root y has t children nodes. Since the valley segments within each subtree are already in descending sequence with respect to their segment values, we need only to *merge* the segments from the t subtrees. This can be implemented efficiently by the multiway merge [1], and it will take at most $\{m \log_2 t\}$ time units to perform the t-way merge. Furthermore, the computation of the new pebble cost sequence on the tree requires at most m time units. Therefore, if $f(m)$ is the amount of work to execute Algorithm 4.1 using Algorithm 6.1 for combining subtree orderings, then

$$f(m) = m \log_2 t + m + \sum_{1 \leq k \leq t} f(m_k),$$

where m_k is the number of nodes in the kth subtree under the node y. This means that $\sum_k m_k = m - 1$.

A simple induction on m will show that $f(m) \leqq m^2$. This upper bound, though attainable, is often too pessimistic. In practice, the number of hill/valley values in the pebble cost sequence Pcost is often much smaller than the number of nodes in the subtree, so that the amount of work required for the merging of segments is usually much smaller than $\{m \log_2 t\}$. Indeed, in the application of this algorithm for storage minimization for the out-of-core sparse matrix factorization, the execution time will usually be linear with respect to the order of the matrix.

7. Concluding remarks. We have shown that the core storage minimization problem for the out-of-core factorization scheme in [12] can be studied using a generalized form of the combinatorial problem of pebble game. An efficient algorithm is provided to solve this generalized pebble game problem. It is based on the notion of cost sequences, adapted from Yannakakis [20].

It is interesting to compare the algorithm provided in § 6 with the two existing algorithms in § 3 for solving the standard pebble game and for solving the general game by postorderings. In the case of the standard pebble game where each pebble value is 1, the cost sequence of each subtree is of the form:

$$(H, 1) = (p(T), 1),$$

where $H = p(T)$ is the hill value for the subtree, and 1 is (necessarily) the valley value at the root of the subtree. Ordering the subtrees in descending sequence of the subtree hill values $\{p(T_k)\}$ is obviously equivalent to ordering them in descending sequence of the subtree segment values $\{p(T_k) - 1\}$. Therefore, the algorithm in § 3 for this standard game is a special case of the general algorithm in § 4.

On the other hand, the use of postorderings implies the use of a restricted form of the cost sequence. The restricted cost sequence can be taken to be of the form:

$$(H, V) = (\tilde{p}(T), \tau(y)),$$

where $H = \tilde{p}(T)$ is the first hill value for the subtree T and $V = \tau(y)$ is the pebble value for the root y of this subtree. Indeed, the algorithm described in § 3 can be viewed as one that orders the subtrees in descending sequence of their segment values $\{\tilde{p}(T_k) - \tau(s_k)\}$ (except that there is only one segment for each subtree).

The methodology provided in this paper to solve the generalized tree pebble game should be of theoretical and algorithmic interest. Currently, in the out-of-core sparse factorization scheme of [12], postorderings are used. In practice, it is simple to implement, and is demonstrated to be very effective. Although one can construct matrix structures to show that postorderings are not sufficient in general for primary storage *minimization*, it should still be highly recommended. More practical justification seems to be warranted for the use of the optimal algorithm presented in this paper in the context of out-of-core factorization.

Appendix.

A.1. Best subtree segment arrangement (Theorem 6.1). In this appendix, we provide detailed proofs for Theorems 6.1–6.3. We first establish the following lemma which is useful to compare two cost sequences based on the partial order "$\prec$."

LEMMA A.1. *Let* π: $y_1, y_2, \cdots, y_m$ *and*

$$\text{Pcost}(T, \pi) = (H_1, V_1, \cdots, H_r, V_r).$$

For two values $\tilde{H}$ *and* $\tilde{V}$, $\tilde{H} \leqq H_j$ *and* $\tilde{V} \leqq V_j$ *for some* j *if and only if there exists some node* y_q *in the sequence* π *such that*

$$\tilde{H} \leqq \text{peb}_\pi(y_q), \qquad \tilde{V} \leqq \min\{\text{peb}_\pi(y_p) | q \leqq p \leqq m\}.$$

Proof. Let $(y_{h_1}, y_{v_1}, \cdots, y_{h_r}, y_{v_r})$ be the nodes at which the values of the pebble cost sequence Pcost(T, π) occur.

"*if part.*" Let the node y_q in the lemma be in the segment between the valley nodes $y_{v_{j-1}}$ and y_{v_j}. From definition, we have

$$\tilde{H} \leqq \text{peb}_\pi(y_q) \leqq \text{peb}_\pi(y_{h_j}) = H_j.$$

Moreover, $q \leqq v_j$, so that by the condition on $\tilde{V}$ in this lemma,

$$\tilde{V} \leqq \text{peb}_\pi(y_{v_j}) = V_j.$$

"*only if part.*" Let $\tilde{H} \leqq H_j$ and $\tilde{V} \leqq V_j$. Then take $q = h_j$. The result is obvious. □

Let us follow the same notations as in § 6.1. That is, let the given tree T be rooted at the node y, which has $s_1, \cdots, s_t$ as its children nodes. To help the discussions and formal proofs, we first introduce a definition.

Consider a node z in the tree. The pebble cost of z in T with ordering π is given by $\text{peb}_\pi(z)$. We shall use the notation $\text{peb}_{\pi[s_k]}(z)$ to denote the pebble contribution to the value $\text{peb}_\pi(z)$ from nodes in the subtree $T[s_k]$. Some properties of this value are expressed in the next lemma, and the proofs are straightforward and are omitted.

LEMMA A.2. (a) $\text{peb}_\pi(z) = \sum_{k=1}^{t} \text{peb}_{\pi[s_k]}(z)$.

(b) *If the node* z *belongs to the subtree* $T[s_k]$, *then* $\text{peb}_{\pi[s_k]}(z)$ *is simply the pebble cost at the node* z *of the tree* $T[s_k]$ *using the induced ordering* $\pi[s_k]$.

(c) *If the node* z *does not belong to the subtree* $T[s_k]$, *then*

$$\text{peb}_{\pi[s_k]}(z) = \text{peb}_{\pi[s_k]}(x_k)$$

where x_k *is the last node from the subtree* $T[s_k]$ *appearing before* z *in the sequence* π. □

COROLLARY A.3. *Consider two orderings* $\tilde{\pi}$ *and* π *on the tree* T *such that* $\tilde{\pi}[s_k] = \pi[s_k]$, *that is, the same subtree ordering when restricted to* $T[s_k]$. *For two nodes*

$\tilde{z}$ and z, let $\tilde{x}_k$ and x_k be the last node from $T[s_k]$ appearing before and including $\tilde{z}$ and z in the ordering $\tilde{\pi}$ and π, respectively:

$$\tilde{\pi}\colon \cdots \cdots \tilde{x}_k \cdots \tilde{z} \cdots, \qquad \pi\colon \cdots \cdots x_k \cdots z \cdots,$$

(a) *if* $\tilde{x}_k = x_k$, *then* $\text{peb}_{\tilde{\pi}[s_k]}(\tilde{z}) = \text{peb}_{\pi[s_k]}(z)$,

(b) *if* $\text{peb}_{\tilde{\pi}[s_k]}(\tilde{x}_k) \leqq \text{peb}_{\pi[s_k]}(x_k)$, *then* $\text{peb}_{\tilde{\pi}[s_k]}(\tilde{z}) \leqq \text{peb}_{\pi[s_k]}(z)$. □

Note that in Corollary A.3, $\tilde{\pi}$ and π can be the same ordering. To illustrate the results, consider the ordering in Fig. 3.2 and the two subtrees under the node y_{19} with children $s_1 = y_6$ and $s_2 = y_{18}$, we have

$$\text{peb}_{\pi}(y_4) = \text{peb}_{\pi[y_6]}(y_4) + \text{peb}_{\pi[y_{18}]}(y_4) = 4 + 0 = 4,$$

$$\text{peb}_{\pi}(y_9) = \text{peb}_{\pi[y_6]}(y_9) + \text{peb}_{\pi[y_{18}]}(y_9) = 3 + 5 = 8.$$

Note also that

$$\text{peb}_{\pi[y_6]}(y_j) = 3 \quad \text{for all } 7 \leqq j \leqq 18,$$

$$\text{peb}_{\pi[y_{18}]}(y_j) = 0 \quad \text{for all } 1 \leqq j \leqq 6.$$

We are now ready to examine properties of orderings that are based on subtree segments, that is, nodes in each subtree segment are ordered consecutively. As in § 6.1, let $\psi_1, \psi_2, \cdots, \psi_t$ be given orderings on the respective subtrees $T[s_1], T[s_2], \cdots, T[s_t]$. These orderings on the subtrees define segments based on their individual valley values. We shall use the term *segment ordering* to refer to any ordering on the entire tree that numbers nodes in each subtree segment consecutively. In other words, each segment ordering corresponds to an arrangement of the subtree segments followed by the node y. The proof of the next lemma is straightforward and is omitted.

LEMMA A.4. *Let π be a segment ordering on T that is compatible with each subtree ordering ψ_k. Let (H, V) be a hill/valley value pair in the pebble cost sequence* $\text{Pcost}(T, \pi)$.

(a) *The hill value H occurs either at a hill location in some subtree $T[s_k]$ or at the node y.*

(b) *The valley value V occurs either at a valley location in some subtree $T[s_k]$ or at the node y.*

(c) *If H occurs at a hill node x in the subtree $T[s_k]$, then V occurs at the valley node in this subtree immediately following x or at the root y.* □

THEOREM A.5. *Let π be a segment ordering on T that is compatible with each subtree ordering ψ_k. Interchanging any two neighboring (subtree) segments that are not in descending sequence of their segment values will not increase the pebble cost sequence.*

Proof. Consider two neighboring (subtree) segments that are not in sequence with respect to their segment values. The two segments must come from two different subtrees, since π maintains the relative order of segments for each subtree and segments from the same subtree are already in descending sequence.

For concreteness, let the first segment belong to the subtree $T[s_a]$ with h_a and v_a as its hill and valley nodes, respectively. Also let $T[s_b]$, h_b, v_b correspond to the second segment. We can view the given ordering as:

$$\pi\colon \cdots(\cdots h_a \cdots v_a)(\cdots h_b \cdots v_b)\cdots$$

where parentheses are used here to identify the two segments. The given condition in the theorem can be expressed as:

$$(**) \qquad \text{peb}_{\pi[s_a]}(h_a) - \text{peb}_{\pi[s_a]}(v_a) \leqq \text{peb}_{\pi[s_b]}(h_b) - \text{peb}_{\pi[s_b]}(v_b)$$

since by Lemma A.2(b), the left- and right-hand sides are the segment values of the two subtree segments.

Now consider the new ordering $\tilde{\pi}$ by interchanging *only* these two segments:

$$\tilde{\pi}\colon \cdots(\cdots h_b \cdots v_b)(\cdots h_a \cdots v_a)\cdots.$$

We are to show that $\mathrm{Pcost}(T, \tilde{\pi}) \prec \mathrm{Pcost}(T, \pi)$. By Lemma A.1, for each hill/valley pair of the new sequence $\tilde{\pi}$, it is sufficient to find a node y_q in the sequence π satisfying the conditions in that lemma. Consider any hill/valley value pair (H, V) in the sequence $\tilde{\pi}$. Let the hill value occur at the node x.

Case 1. x is outside the two segments under consideration. Then by Lemma A.4(a), x must be either the root y or a hill location in one subtree. Choose y_q to be the same node x in the π sequence, and it is easy to verify that this node satisfy the conditions in Lemma A.1.

Case 2. x belongs to the segment $(\cdots h_b \cdots v_b)$. By Lemma A.4(a), x must be the node h_b. The node y_q for Lemma A.1 will be chosen to be h_b $(=x)$ in the π sequence. Indeed, applying Corollary A.3, we have

$$\begin{aligned}
\mathrm{peb}_{\tilde{\pi}}(x) &= \sum_{k \neq a} \mathrm{peb}_{\tilde{\pi}[s_k]}(h_b) + \mathrm{peb}_{\tilde{\pi}[s_a]}(h_b) \\
&= \sum_{k \neq a} \mathrm{peb}_{\pi[s_k]}(h_b) + \mathrm{peb}_{\tilde{\pi}[s_a]}(h_b) \\
&\leqq \sum_{k \neq a} \mathrm{peb}_{\pi[s_k]}(h_b) + \mathrm{peb}_{\pi[s_a]}(v_a) \\
&= \sum_{k \neq a} \mathrm{peb}_{\pi[s_k]}(h_b) + \mathrm{peb}_{\pi[s_a]}(h_b) = \mathrm{peb}_{\pi}(h_b).
\end{aligned}$$

By Lemma A.4, we have

$$V = \min\{\mathrm{peb}_{\tilde{\pi}}(v_b), \mathrm{peb}_{\tilde{\pi}}(y)\}.$$

Applying Corollary A.3, we have

$$\mathrm{peb}_{\tilde{\pi}[s_a]}(v_b) \leqq \mathrm{peb}_{\pi[s_a]}(v_b)$$

so that

$$\mathrm{peb}_{\tilde{\pi}}(v_b) \leqq \mathrm{peb}_{\pi}(v_b).$$

Therefore, the value V must be less than the accumulated pebble value of any node after h_b in π.

Case 3. x belongs to the segment $(\cdots h_a \cdots v_a)$. This means that $x = h_a$. We shall choose y_q for Lemma A.1 again to be the node h_b in π. Again by Corollary A.3, we have

$$\begin{aligned}
\mathrm{peb}_{\tilde{\pi}}(x) &= \sum_{k \neq a,b} \mathrm{peb}_{\tilde{\pi}[s_k]}(h_a) + \mathrm{peb}_{\tilde{\pi}[s_a]}(h_a) + \mathrm{peb}_{\tilde{\pi}[s_b]}(h_a) \\
&= \sum_{k \neq a,b} \mathrm{peb}_{\pi[s_k]}(h_a) + \mathrm{peb}_{\pi[s_a]}(h_a) + \mathrm{peb}_{\tilde{\pi}[s_b]}(v_b) \\
&= \sum_{k \neq a,b} \mathrm{peb}_{\pi[s_k]}(h_a) + \mathrm{peb}_{\pi[s_a]}(h_a) + \mathrm{peb}_{\pi[s_b]}(v_b).
\end{aligned}$$

But, by the given condition $(**)$ on the two segments in π, this value must be no greater than

$$\begin{aligned}
\sum_{k \neq a,b} \mathrm{peb}_{\pi[s_k]}(h_a) + \mathrm{peb}_{\pi[s_b]}(h_b) + \mathrm{peb}_{\pi[s_a]}(v_a) &= \sum_{k \neq a,b} \mathrm{peb}_{\pi[s_k]}(h_b) + \mathrm{peb}_{\pi[s_b]}(h_b) + \mathrm{peb}_{\pi[s_a]}(h_b) \\
&= \mathrm{peb}_{\pi}(h_b).
\end{aligned}$$

The condition on the value V can be verified in the same way as in Case 2.

Therefore, in all cases, for a given pair (H, V) in $\text{Pcost}(T, \tilde{\pi})$, we can bound them by a corresponding pair from $\text{Pcost}(T, \pi)$. It follows from definition that

$$\text{Pcost}(T, \tilde{\pi}) \prec \text{Pcost}(T, \pi). \qquad \square$$

Proof of Theorem 6.1. Let $\psi = \Phi(\psi_1, \cdots, \psi_t)$. Consider any given segment ordering π' compatible with each subtree ordering ψ_k. A finite number of neighboring subtree segment interchanges will transform π' to ψ. Repeated applications of Theorem A.5 for such interchanges together with the transitivity of "$\prec$" (Theorem 5.2) will show that

$$\text{Pcost}(T, \psi) \prec \text{Pcost}(T, \pi'). \qquad \square$$

A.2. Segment orderings are sufficient (Theorem 6.2). Theorem 6.2 says that in order to search for an ordering that will minimize the cost sequence, it is sufficient to look for a segment ordering that is compatible with the subtree orderings. The following is a constructive proof.

Proof of Theorem 6.2. Let π be any given ordering on the tree T rooted at y, which is compatible with each subtree ordering ψ_k of the subtree $T[s_k]$. We shall prove the result by constructing a segment ordering π' such that

$$\text{Pcost}(T, \pi') \prec \text{Pcost}(T, \pi).$$

Construct the new ordering π' from π as follows:

(a) Remove all nodes from the sequence π except the root y and hill locations of the subtrees;

(b) Replace each hill location by the subtree segment associated with it.

It should be clear that π' is still compatible with each subtree ordering ψ_k, and orders nodes in each subtree segment consecutively. Moreover, it maintains the relative order of all the subtree hill nodes in the original ordering π. It remains to show that the pebble cost sequence of π' is no greater than that of π.

Consider any hill/valley value pair (H, V) in $\text{Pcost}(T, \pi')$. Let the hill value occur at the node x. It is sufficient to find a node y_q in the original sequence π satisfying the conditions in Lemma A.1. If x is the root y, pick this root as the node y_q and the conditions in Lemma A.1 are clearly satisfied. Otherwise, by Lemma A.4(a), x must be a hill node in one of the subtrees. Let $x = h_a$ belonging to the subtree $T[s_a]$, and v_a be the valley node immediately following h_a in the pebble cost sequence of this subtree. That is

$$\pi' \colon \cdots (\cdots h_a \cdots v_a) \cdots .$$

We now show that y_q for Lemma A.1 can be chosen to be the node x. We first claim that for each k, $\text{peb}_{\pi'[s_k]}(x) \leqq \text{peb}_{\pi[s_k]}(x)$. By Corollary A.3, since x belongs to $T[s_a]$, we have

$$\text{peb}_{\pi'[s_a]}(x) = \text{peb}_{\pi[s_a]}(x).$$

For $k \neq a$, let the last segment from the subtree $T[s_k]$ before x in π' be $(\cdots h_k \cdots v_k)$. (If no such segment exists, then $\text{peb}_{\pi'[s_k]}(x) = 0$, and the result holds.) By Corollary A.3,

$$\text{peb}_{\pi'[s_a]}(x) = \text{peb}_{\pi'[s_a]}(v_k) = \text{peb}_{\pi[s_a]}(v_k).$$

Let x_k be the last node from $T[s_k]$ before x in π. Since π' maintains the relative order of all the subtree hill values, x_k must appear after the hill node h_k of $T[s_k]$. By the definition of the valley node v_k and Corollary A.3, we have

$$\text{peb}_{\pi[s_k]}(v_k) \leqq \text{peb}_{\pi[s_k]}(x_k) = \text{peb}_{\pi[s_k]}(x).$$

Combining, we have proved the claim.

Using the result of the claim, we then have

$$\text{peb}_{\pi'}(x) = \sum_k \text{peb}_{\pi'[s_k]}(x)$$

$$\leqq \sum_k \text{peb}_{\pi[s_k]}(x) = \text{peb}_{\pi}(x).$$

Finally, by Lemma A.4(c),

$$V = \min \{\text{peb}_{\pi'}(v_a), \text{peb}_{\pi'}(y)\}$$

and we need to show that $V \leqq \text{peb}_{\pi}(z)$, for all nodes z after x in the π sequence. If $z = y$, it is obviously true. Otherwise, it can be verified that for all k, $\text{peb}_{\pi'[s_k]}(v_a) \leqq \text{peb}_{\pi[s_k]}(z)$. This implies that $\text{peb}_{\pi'}(v_a) \leqq \text{peb}_{\pi}(z)$, and hence the result. □

A.3. Monotonicity of Combine algorithm (Theorem 6.3). Theorem 6.3 provides the monotone property of the "Combine" algorithm with respect to subtree orderings. In words, better subtree orderings will yield a better overall ordering by Algorithm 6.1. Before the proof, we introduce a lemma.

LEMMA A.6. *Given two cost sequences with*

$$(\tilde{H}_1, \tilde{V}_1, \cdots, \tilde{H}_{\tilde{r}}, \tilde{V}_{\tilde{r}}) \prec (H_1, V_1, \cdots, H_r, V_r).$$

For $1 \leqq i \leqq \tilde{r}$, *define the function* $f(*)$ *by*

$$f(i) = \min \{k | \tilde{H}_i \leqq H_k, \tilde{V}_i \leqq V_k\}.$$

If $i < j$, *then* $f(i) \leqq f(j)$.

Proof. By definition of $f(i)$ and Lemma 5.1, we have

$$\tilde{H}_j \leqq H_{f(j)}, \qquad \tilde{V}_i < \tilde{V}_j \leqq V_{f(j)}.$$

If $\tilde{H}_i \leqq H_{f(j)}$, then by definition of $f(i)$, we must have $f(i) \leqq f(j)$. On the other hand, if $H_{f(j)} < \tilde{H}_i$, which together with $\tilde{H}_i \leqq H_{f(i)}$, we have $H_{f(j)} < H_{f(i)}$. By Lemma 5.1, we must have $f(i) < f(j)$. □

Proof of Theorem 6.3. Let $\bar{\psi}_k$ and ψ_k be two subtree orderings on $T[s_k]$, with

$$\text{Pcost}(T[s_k], \bar{\psi}_k) \prec \text{Pcost}(T[s_k], \psi_k).$$

As in the theorem, let

$$\psi = \Phi(\psi_1, \cdots, \psi_k, \cdots, \psi_t).$$

We shall prove the result by first improving on the ordering ψ. We are going to replace subtree segments from $\text{Pcost}(T[s_k], \psi_k)$ in the ordering ψ by those from $\text{Pcost}(T[s_k], \bar{\psi}_k)$. Since $\text{Pcost}(T[s_k], \bar{\psi}_k) \prec \text{Pcost}(T[s_k], \psi_k)$, we can associate each segment from $\bar{\psi}_k$ to one in ψ_k using the mapping $f(*)$ of Lemma A.6. From ψ, construct the new ordering ψ' as follows:

(a) For the ith subtree segment from $\text{Pcost}(T[s_k], \bar{\psi}_k)$, insert it before the corresponding $f(i)$th segment of $\text{Pcost}(T[s_k], \psi_k)$ in ψ;

(b) Remove all subtree segments of $\text{Pcost}(T[s_k], \psi_k)$ from the ordering.

It is clear that ψ' is a segment ordering using the new $\bar{\psi}_k$. Furthermore, by Lemma A.6, it is compatible with the new subtree ordering $\bar{\psi}_k$. We claim that

$$\text{Pcost}(T, \psi') \prec \text{Pcost}(T, \psi).$$

The proof uses the same technique (Lemma A.1) as before and will be skipped.

Finally, let

$$\bar{\pi} = \Phi(\psi_1, \cdots, \bar{\psi}_k, \cdots, \psi_t).$$

By Theorem 6.1, we have

$$\text{Pcost}(T, \bar{\pi}) \prec \text{Pcost}(T, \psi'),$$

and hence the result. □

Acknowledgment. The author would like to thank Professor Andranik Mirzaian for many helpful discussions.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.

[2] S. A. COOK AND R. SETHI, *Storage requirements for deterministic polynomial finite recognizable languages*, J. Comput. System Sci., 13 (1976), pp. 25–37.

[3] I. S. DUFF AND J. K. REID, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Trans. Math Software, 9 (1983), pp. 302–325.

[4] M. R. GAREY, R. L. GRAHAM, D. S. JOHNSON AND D. E. KNUTH, *Complexity results for bandwidth minimization*, SIAM J. Appl. Math, 34 (1978), pp. 477–495.

[5] J. A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[6] J. GILBERT, T. LENGAUER AND R. TARJAN, *The pebbling problem is complete in polynomial space*, SIAM J. Comput., 9 (1980), pp. 513–525.

[7] T. LENGAUER, *Black-white pebbles and graph separation*, Acta Informatica, 16 (1981), pp. 465–475.

[8] T. LENGAUER AND R. E. TARJAN, *The space complexity of pebble games on trees*, Inform. Process. Lett., 10 (1980), pp. 184–188.

[9] ———, *Asymptotically tight bounds on time-space trade-offs in a pebble game*, J. ACM, 29 (1982), pp. 1087–1130.

[10] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 37 (1979), pp. 177–189.

[11] J. W. H. LIU, *On the storage requirement in the out-of-core multi-frontal method for sparse factorization*, Tech. report CS-85-02, Dept. of Computer Science, York University, 1985; ACM Trans. on Math Software, to appear.

[12] ———, *An adaptive general sparse out-of-core scheme for sparse Cholesky factorization*, SIAM Sci. Statist. Comput., 8 (1987), pp. 585–599.

[13] ———, *A compact row storage scheme for sparse Cholesky factors using elimination trees*, ACM Trans. Math Software, 12 (1986), pp. 127–148.

[14] M. C. LOUI, *The space complexity of two pebble games on trees*, M.I.T. Technical Report MIT/LCS/TM-133, 1979.

[15] F. MEYER AUF DER HEIDE, *A comparison of two variations of a pebble game on graphs*, Theoret. Comput. Sci., 13 (1981), pp. 315–322.

[16] N. PIPPENGER, *Pebbling*, IBM Research Report RC8258, 1980.

[17] J. K. REID, TREESOLVE, *a Fortran package for solving large sets of linear finite element equations*, CSS 155, Comput. Sci. Syst. Division, AERE Harwell, Oxfordshire, 1984.

[18] R. SCHREIBER, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math Software, 8 (1982), pp. 256–276.

[19] R. E. TARJAN, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conference Series in Applied Math, Society for Industrial and Applied Mathematics Publication, 1983.

[20] M. YANNAKAKIS, *A polynomial algorithm for the min cut linear arrangement of trees*, J. ACM, 32 (1985), pp. 950–988.