SIAM J. ALG. DISC. METH.
Vol. 6, No. 4, October 1985

[13]

PROFILE SCHEDULING OF OPPOSING FORESTS AND LEVEL ORDERS*

DANNY DOLEV† AND MANFRED K. WARMUTH‡

Abstract. The question of existence of a schedule of a given length for n unit length tasks on m identical processors subject to precedence constraints is known to be NP-complete [Ullman, J. Comput. System Sci., 10 (1976), pp. 384-393]. For a *fixed value* of m we present polynomial algorithms to find an optimal schedule for two families of precedence graphs: level orders and opposing forests. In the case of opposing forest our algorithm is a considerable improvement over the algorithm presented in [Garey et al., SIAM J. Alg. Disc. Meth., 4 (1983), pp. 72-93].

1. Introduction. The goal of deterministic scheduling is to obtain efficient algorithms under the assumption that all the information about the tasks to be scheduled is known in advance [Co76], [GL79]. One of the fundamental problems in deterministic scheduling is to schedule a collection of n partially ordered, unit length tasks on a number of identical processors. As in [GJ83], [DW84a], [DW84b] we allow the number of identical processors to vary with time. This is described by a sequence of natural numbers, called a *profile* specifying how many processors are available at each unit of time (time slot). The *breadth* m, of a profile is an upper bound on the number of processors available at any time. A profile is *straight* if the number of available processors is the same at any time.

A *schedule* for a given profile is a partitioning of all the tasks into a sequence of sets which does not violate the precedence constraints and the number of tasks in each set does not exceed the number of available processors specified by the profile for the corresponding time slot.

Various aspects of scheduling theory have been extensively studied in recent years [GL79] and many scheduling problems are known to be NP-complete [GJ79]. The first NP-completeness result on scheduling with precedence constraints was published by Ullman [Ul75]. He showed that the existence of a schedule of a given length on a straight profile for a collection of unit length tasks subjected to precedence constraints is NP-complete in case where the breadth of the profile is a variable of the problem, that is, the breadth of the profile is not bounded by a constant. This problem remains NP-complete even for precedence graphs of special forms [GJ83], [Ma81], [Wa81].

Polynomial algorithms have been developed only for a few special cases of scheduling unit length tasks with precedence constraints. The first polynomial algorithm was developed by Hu [Hu61]. It produces an optimal schedule for a straight profile of arbitrary breadth if the precedence graph is either an inforest or an outforest. Hu's algorithm produces a schedule according to the *Highest Level First* (HLF) strategy, meaning tasks of higher level are chosen over tasks of lower level and among tasks of the same level ties are broken arbitrarily. Restricted versions of HLF provide optimal schedules if the precedence graph is an interval order [PY79], [Ga81], or if the number of available processors is two [FK71], [CG72], [Ga82].

The major scheduling problem remaining open is whether the scheduling of an arbitrary graph is NP-complete or polynomial for fixed number ($m \geqq 3$) of processors. In this paper we address two special cases of the above open problem. We utilize the

* Received by the editors August 2, 1982, and in revised form May 31, 1984.

† IBM Research Laboratory, San Jose, California 95193. Current address: Institute of Mathematics and Computer Science, Hebrew University, Jerusalem, Israel.

‡ Computer Science Department, University of California, Santa Cruz, California 95064.

results presented in [Wa81], [DW84a] to obtain polynomial algorithms for two families of precedence constraints (precedence graphs): level orders and opposing forests. A graph is a level order if each connected component is partitioned into some k levels $L_0, \cdots, L_{k-1}$ such that for every two tasks $x \in L_i$ and $y \in L_j$, where $i > j$, x precedes y. We present an algorithm for finding optimal schedules for this class that requires time and space $O(n^{m-1})$. An opposing forest [GJ83] is a graph composed of intrees and outtrees only. It is a generalization of the cases solvable by Hu's [Hu61] algorithm. Garey, et al., [GJ83] presented a polynomial algorithm for finding an optimal schedule in the case of opposing forest and straight profile of fixed breadth $m \geqq 3$. Their algorithm costs $O(n^{m^2+2m-5} \log n)$ time and $O(n)$ space. The algorithm we presented for this case is bounded by $O(n^{2m-2} \log n)$ time and $O(n^{m-1})$ space. For the special case $m = 3$ there exist a linear algorithms to find an optimal schedule [DW84a], [GJ83].

Our polynomial algorithms are based on the reduction theorem, which is proved in § 3. The reduction theorem is another form of the elite theorem [DW84a]. It reduces the number of components we have to consider at each step of the algorithm to at most $m-1$ (the highest ones) and therefore enables us to obtain efficient algorithms.

Notice that if the breadth of the profile is a variable of the problem rather than fixed, then scheduling a level order or an opposing forest becomes NP-complete [GJ83], [Ma81], [Wa81]. Thus our algorithms are expected to have a high complexity (exponential in the breadth m). A similar case was published in [DW84b]. It was shown that scheduling a precedence graph of bounded height on a profile of fixed breadth is polynomial. For profiles of arbitrary breadth the problem is again NP-complete [LR78], even if there is an arbitrary number of processors in only one time slot and one processor in all other slots [Wa81], [DW84a].

In § 2 we present the main notions used in the rest of the paper. Section 3 contains the reduction theorem. In §§ 4 and 5 we present the polynomial algorithm for level orders and opposing forests, respectively.

2. Basic definitions and properties.

2.1. Graph definitions. A *(precedence) graph* G is a directed acyclic graph given as a tuple (V, E), where V is the set of n *vertices* (or *tasks*) and E the set of *edges* of G. A *(directed) path* π of length r in a precedence graph $G = (V, E)$ is a sequence of vertices $x_0, \cdots, x_r$ such that the edge (x_i, x_{i+1}), for $0 \leqq i \leqq r-1$, is in E. A precedence graph G specifies the precedence constraints between the vertices (tasks) of G. We assume that if a task x has to be executed before a task y, then there exists a (directed) path of positive length from x to y in G, that is, x is a *predecessor* of y, and y is a *successor* of x. In the case where the longest path from a vertex x to a vertex y is the edge (x, y), x is an *immediate predecessor* of y and y is an *immediate successor* of x. Vertices x and y are *incomparable*, if x is neither a predecessor nor a successor of y. A set of vertices is incomparable if for any two vertices x and y of the set, x and y are incomparable, that is, there is no path between any two distinct vertices of the set.

By $h(G)$ we mean the *height* of G, which is the length of the longest path in G. For a vertex $x \in G$ (i.e., $x \in V$) we denote by $h(x)$ the length of the longest path that starts at x. A vertex with no successors has zero height. Vertices with identical height are said to be at the same *level*. Observe that all vertices of the same level are incomparable.

The graph G' is a *(closed) subgraph* of G if every vertex of G' has the same successors in G' as it has in G. A vertex of G, is *initial* if it has no predecessors. Note that an initial vertex of G is not necessarily of maximum height in G. A set of t *highest initial vertices* of G is a subset of initial vertices containing the t highest ones. Ties are resolved arbitrarily. If there are less than t initial vertices then the set consists of all of them.

Let R be a set of initial vertices of G; then $G-R$ is the closed subgraph of G obtained by removing all the vertices of R from G. Given two graphs $G=(V, E)$ and $G'=(V', E')$, then $G \cup G'$ denotes the graph $(V \cup V', E \cup E')$. The graph $G=(V, E)$ is composed of $\{G_1, \cdots, G_r\}$ if these closed subgraphs (called *components* of G) are a decomposition of G into its connected components, that is each closed subgraph is a connected graph and there are no edges between vertices of different components; therefore, $G=\bigcup_i G_i$.

An *inforest* (respectively *outforest*) is a graph in which each vertex has at most one immediate successor (respectively one immediate predecessor). Notice that outforest is composed of components, each of which has exactly one initial vertex and it consists of this vertex and all its successors. A component of an outforest is called *outtree* and similarly a component of an inforest is called *intree*.

In a *level order* graph each component has the following form: Every vertex of level i precedes all vertices of the component from all the levels below i. Note that all vertices of the same component of a level order that are at the same level are isomorphic. Thus, we can assume that such a component is given as a tuple specifying how many vertices are in each level of the component.

2.2. Profile definitions. We partition the time scale into *time slots* of length one. The time interval $[i-1, i)$ for $i \geqq 1$ is the ith time slot. A *profile* is a sequence of positive integers specifying the number of identical processors that are available in each time slot. We shall interpret profile $M=(m_1, \cdots, m_d)$, where d is its length, to mean that for each slot i in $[0, d)$ there are m_i processors available.

The *breadth* of profile M is the upper bound on the number of processors that are available at any time slot of M. The profile of Table 2.1 has breadth 4. Throughout the paper we denote the breadth of the given profile with the letter m. We call a profile M *straight* if $m_i=m$, for all $1 \leqq i \leqq d$.

TABLE 2.1
A schedule for G fitting the profile $M=(2, 4, 2, 1, 1)$.

slot	1	2	3	4	5
P_1	4	1	3	6	
P_2	5	2	9		
P_3		7			
P_4		8			
m_i	2	4	2	1	1

2.3. Schedule definition. A *schedule* S for a precedence graph G is a sequence of sets $(S)_1, \cdots, (S)_k$ such that:

(i) the sets $(S)_i$, for $1 \leqq i \leqq k$, partition the vertices of G;

(ii) if $x \in (S)_i$ and $y \in (S)_j$, for $1 \leqq i \leqq j \leqq k$, then there is no path from y to x.

The *length* of a schedule $\lambda(S)$ is the index of the last nonempty set in the sequence. A minimal length schedule is called *optimal*. The schedule S *fits* the profile M if the length of S is not greater than the length of the profile and the cardinality of $(S)_i$ is not greater than m_i. The set of tasks $(S)_i$ get executed in the ith time slot, that is $|(S)_i|$ of the m_i processors of slot i each execute a task of $(S)_i$ during the time interval $[i-1, i)$. Note that all the tasks have unit length, which corresponds to the length of a time slot. An example is given in Fig. 2.1 and Table 2.1. The ith slot of S, $1 \leqq i \leqq \lambda(S)$,

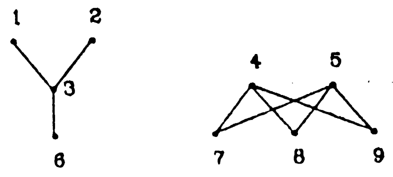

FIG. 2.1. *A precedence graph G.*

has $m_i - |(S)_i|$ *idle periods* meaning that there are this many processors idle during time slot i of S.

Given a precedence graph G and profile M, the initial problem is to determine if a schedule S exists for G and M. If a feasible schedule does exist, then we look for the shortest schedule S for G that fits M. In the first issue we allow the possibility that there does not exist a schedule for G that fits M. In the second we assume that there exists a feasible schedule and we are only interested in an optimal schedule.

A schedule S is an HLF-*schedule* for G and M if $(S)_i$, $1 \leqq i \leqq \lambda(S)$, is a set of m_i highest initial tasks of the closed subgraph of G induced by all tasks scheduled in slot i of S or later. HLF-schedules have the following property. Assume task x is scheduled in slot i and y is scheduled in slot j. If $h(x) > h(y)$, then either $i \leqq j$ or there is a predecessor of x in the jth slot. We say that HLF *produces an optimal schedule* if any HLF-schedule is optimal; that is, if an optimal schedule can be constructed by choosing higher initial tasks before lower ones and choosing arbitrarily among initial tasks of the same height. Note that the schedule of Table 2.1 is not a HLF-schedule; moreover, no HLF-schedule is optimal for G (Fig. 2.1) and the profile of Table 2.1.

2.4. The median. The following definition relates the number of components of a graph and the heights of the components with m; where m is the breadth of the profile.

DEFINITION. The *median* of precedence graph G with respect to a given m, denoted by $\mu(G)$, is one plus the height of some mth highest component of G. If the graph has less than m components, then the median is 0.

For example, if the precedence graph is the one in Fig. 2.2 and the breadth of the given profile is three, then the median is three because three is one plus the height of the third highest component. For the graph described by Fig. 2.1 the median is 0 with respect to $m = 3$.

We use the median to split the precedence graph G into two subgraphs. Let $G = H(G) \cup L(G)$, where the *high-graph* $H(G)$ contains all components of G that are strictly higher than the median; the *low-graph* $L(G)$ is the remaining subgraph of G. Note that $H(G)$ has at most $m-1$ components. Fig. 2.2 presents such a splitting of a precedence graph. We sometimes write $\mu(G, m)$, $H(G, m)$ and $L(G, m)$ to denote the median, the high-graph and the low-graph, respectively, for a specific m.

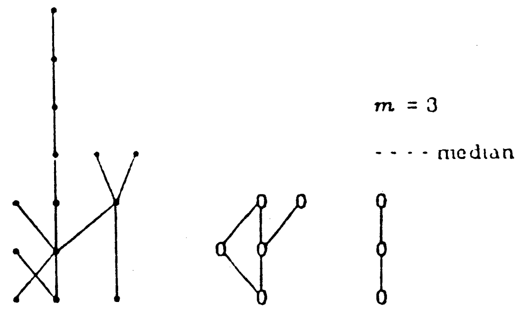

FIG. 2.2. *The decomposition of a graph G into H(G) and L(G); • denote vertices of H(G) and 0 vertices of L(G).*

The following properties of the median are used in the current paper.

PROPERTIES OF THE MEDIAN.

M1: There are at most $m-1$ components of G having height at least $\mu(G)$.

M2: If $\mu(G)>0$, then there are at least m components of G having height at least $\mu(G)-1$.

M3: If G has at most $m-1$ components of height at least h, then $\mu(G)\leqq h$.

M4: If G has at least m components of height at least $h-1$, then $\mu(G)\geqq h$.

The above properties follow directly from the definition of the median. Further properties of the median were given in [DW84a].

3. Reduction theorem.

In this section we present our main result, the reduction theorem. We also prove several related theorems that are needed in later sections. The reduction theorem is a consequence of the MERGE Algorithm. The following lemma implies the correctness of the MERGE Algorithm. A component of a graph G is called *principal* if its height is at least $h(G)-1$.

LEMMA 3.1. *Let G be a graph and let G' be a subgraph of G obtained by removing a set of q highest initial tasks from G. Then G' contains at least as many principal components as the original graph G, unless $h(G')=0$.*

Proof. If the lemma holds for $q=1$, then it clearly holds for arbitrary q. Let x be a highest vertex of G, I be the principal component of G that contains x and $G'=G-\{x\}$. Assume $h(G')>0$. To show that G' contains at least as many principal components as G observe that $h(I)=h(G)>0$ and therefore $I-\{x\}$ contains a principal component of G'. Furthermore all principal components of G other than I are also principal components of G', because $h(G')\leqq h(G)$. We conclude that the number of principal components does not decrease when x is removed. □

The following algorithm shows how one can "merge" a schedule for a collection of subgraphs with a collection of subgraphs of lower height to get a schedule for the combined graph.

ALGORITHM 3.1. (the MERGE Algorithm)

```
Input:   A graph $L=\bigcup_{i=1}^{r} L_i$, such that $h(L_i)\geqq h(L)-1$;
         a graph $H=\bigcup_{i=1}^{q} H_i$, such that $h(H_i)>h(L)$, and $q+r\geqq m$;
         a schedule $S$ for $H$ and $M$ with $p$ idle periods, where $M$ is a profile
         of breadth $m$.
Output:  A schedule $S'$ for $H\cup L$ and $M$ such that $S'$ is not longer than the
         schedule $S$ in the case where $p\geqq|L|$; and otherwise, $S'$ is longer than
         $S$ but has idle periods only in its last slot.
1. $k:=0$
   $S':=S$
2. While $h(L)>0$ do
   2.1. $k:=k+1$
   2.2. While $(S')_k$ is not full and not all initial vertices of $H$ are scheduled in
        $(S')_k$ do
        2.2.1. Transfer an initial vertex of $H$ from a slot after $k$ to $(S')_k$.
   2.3. Fill $(S')_k$ with $m_k-|(S')_k|$ highest initial vertices of $L$.
   2.4. Remove the vertices of $(S')_k$ from $L$, $H$ and its subgraphs $H_i$.
   2.5. While there is a subgraph $H_i$ of $H$, such that $h(H_i)=h(L)$ do
        2.5.1. Transfer the graph $H_i$ from $H$ to $L$.
               $q:=q-1$; $r:=r+1$
        2.5.2. Remove the vertices of $H_i$ from $S'$.
```

```
3. While $L$ is nonempty do
   3.1. $k := k+1$
   3.2. While $(S')_k$ is not full or $L$ is not empty do
        3.2.1. Add a vertex of $L$ to slot $k$ of $S'$ and remove it from $L$.
```

A high level description of the MERGE *algorithm*. The aim of the algorithm is to "merge" the schedule S for H and M with the vertices of L producing a schedule S' for $H \cup L$ and M. The length of S' depends on the relationship between p, the number of idle periods in S and the number of vertices in L. If $p > |L|$, then there is enough "space" in S for all vertices of L and the resulting schedule S' is at most as long as S. Otherwise, S does not have enough idle periods and S' is longer than S. In this case, S' only has idle periods in its last slot.

At Step 1 of the algorithm we initialize S' with the schedule S for H and M. During Steps 2 and 3 the vertices of L are added into in S'. While doing so we sometimes reschedule vertices of, H in S' (see Steps 2.2.1 and 2.5.2).

If $h(L) = 0$, then "merging" is easy (see Step 3). In this case, L is a set of single vertices. The algorithm consecutively fills the slots of S' with vertices of L until L is empty.

If $h(L) > 0$, then "merging" is slightly more involved (see Step 2). The variable q will be the number of subgraphs H_i that are left in H. All of these graphs will have height bigger than $h(L)$. If some of them drop down to height $h(L)$ during Step 2.4 then these subgraphs are transferred from H to L at Step 2.5. The variable r has the following meaning. During the algorithm it will be assumed that L has at least r principal components. The sum of q and r is at least m throughout the loop 2. This assures that there will be at least m initial vertices in $H \cup L$, at least q in H and at least r in L. We transfer components from H to L to avoid that some subgraphs H_i of H get completely scheduled and the sum of q and r drops below m.

Correctness of the MERGE *algorithm*: In the new schedule S' the precedence constraints specified by G are not violated, because we iteratively add vertices to S' (Steps 2.2.1, 2.3 and 3.2) that are initial in the unscheduled portion of $H \cup L$. Loop 2 has the following invariant: L has at least r principal components and H has q subgraphs H_i of height bigger than $h(L)$ and $q + r \geq m$.

Note that by the definition of H, L, q and r the loop invariant trivially holds after Step 1. We want to show that if the loop invariant holds before Step 2.1 and $h(L)$ is bigger than zero then it holds after Step 2.5, or $h(L)$ equals zero.

At Step 2.4 only initial vertices are removed from H, H_i and L. Therefore, their height can drop at most by one. This assures that after Step 2.4 the graph H contains q subgraphs H_i of height at least $h(L)$. By Lemma 3.2 we know that after Step 2.4 either the graph L contains at least r principal components or $h(L) = 0$. Note that at Step 2.3 $(S')_k$ was filled with highest initial vertices of L. At Step 2.5 all subgraphs H_i of H that dropped down to height $h(L)$ are transferred from H to L. The height $h(L)$ and the sum $q + r$ does not change during Step 2.5. Furthermore, if before Step 2.5.1 L has at least r principal components then L has also at least r principal components after Step 2.5.1, since each H_i that is transferred contains at least one component of height $h(L)$. This completes the proof of the invariant of Loop 2.

The following claim completes the proof of correctness. It shows that if $p \geq |L|$ then $\lambda(S') \leq \lambda(S)$, and if $p < |L|$ then $\lambda(S') > \lambda(S)$ and S' has idle periods only in its last slot.

CLAIM 3.1.

(i) *After Step 3 the schedule $(S')_1, \cdots, (S')_{k-1}$ does not have any idle periods.*

(ii) *After Step* 3 *either* $k = \lambda(S')$ *or* $\lambda(S') \leqq \lambda(S)$.

(iii) *If* $\lambda(S') > \lambda(S)$ *then* S' *can have idle periods only in its last slot.*

(iv) $p \geqq |L|$ *if and only if* $\lambda(S') \leqq \lambda(S)$.

Proof of (i). By the loop invariant we know that the current slot is filled up in Steps 2.2 and 2.3. Thus, the schedule $(S')_1, \cdots, (S')_k$ does not have any idle periods when Step 3 is reached. At Step 3 all the slots, except may be the last one, are completely filled. This completes the proof of (i).

Proof of (ii). At Step 1 the schedule S' is initialized with S, and therefore $\lambda(S') = \lambda(S)$. During Steps 2 and 3 the algorithm never adds any vertices to any slot of S' with a higher index than the current slot k. On the other hand, in Steps 2.2.1 and 2.5.2 there are vertices removed out of slots with higher indices than the current slot k. This implies that after Step 3, $k = \lambda(S')$ or $\lambda(S') \leqq \lambda(S)$.

Proof of (iii). If $\lambda(S') > \lambda(S)$ then (ii) implies that $k = \lambda(S')$ after Step 3. Applying (i) we get that S' can have idle periods only in its last slot.

Proof of (iv). Assume $\lambda(S') > \lambda(S)$; then by (iii) we know that S' can have idle periods only in its last slot. In particular, there are no idle periods in slots 1 through $\lambda(S)$ of S', which implies that $\sum_{i=1}^{\lambda(S)} m_i < |H| + |L|$. Since p can be expressed as $(\sum_{i=1}^{\lambda(S)} m_i) - |H|$, it follows that $p < |L|$.

To prove the opposite direction of (iv) assume that $p < |L|$. Expressing p as $\sum_{i=1}^{\lambda(S)} m_i - |H|$ implies that $\sum_{i=1}^{\lambda(S)} m_i < |H| + |L|$. Since S' is a schedule for $H \cup L$, we have $|H| + |L| \leqq \sum_{i=1}^{\lambda(S')} m_i$. Combining both inequalities we get $\sum_{i=1}^{\lambda(S)} m_i < \sum_{i=1}^{\lambda(S')} m_i$, which implies $\lambda(S) < \lambda(S')$.

Herewith we completed the proof of the claim and the proof of correctness of the MERGE algorithm. □

The MERGE algorithm is linear even if G is not transitively reduced [AH74].

LEMMA 3.2 [Wa81]. *The* MERGE *algorithm can be implemented in time and space* $O(n+e)$, *where* n *is the number of vertices and* e *the number of edges in* $H \cup L$.

Proof. We only give a general idea of the implementation of the MERGE algorithm. A complete description appears in [Wa81]. We keep track of the set of current initial vertices of H and L; call these sets I_H and I_L, respectively. Whenever we remove vertices from these sets we add the vertices that become initial to the list.

In Step 2.2.1 we can choose any vertex of I_H that is not already in $(S')_k$. On the other hand, vertices of I_L should be scheduled according to their height (Step 2.3). Thus we need a data structure that will enable us to retrieve vertices from I_L efficiently. We represent I_L as an array of lists, where the entry $I_L(h)$ points to a linked list of all the initial vertices of height h (in arbitrary order); see [DW84a], [Wa81] for details. As shown in the proof of correctness there are always enough initial vertices in $I_L(h(L))$ and $I_L(h(L)-1)$ to fill $(S')_k$ in Step 2.3. Thus it is enough to pick vertices of the last and second to last nonempty list of I_L. This is the main reason for the fact that the MERGE algorithm can be implemented in $O(n+e)$ time. We do not have to do a complicated search to find highest vertices in Step 2.3.

For Step 2.5 we need to keep track of the heights of the subgraphs H_i. This is easy since during each iteration of the loop the height of a subgraph H_i can drop at most by one. To be able to transfer components easily we need to keep track of the vertices of each H_i and keep pointers from each vertex of G to all its occurrences in the data structures. This completes the summary of the proof. □

The reduction theorem is an immediate consequence of the following theorem, in which we apply the MERGE algorithm 3.1.

THEOREM 3.1. *Let* G *be a graph and* M *be a profile of breadth* m. *Given a schedule* S *for the high-graph of* G *and* M *that has* p *idle periods, then with the* MERGE *algorithm* *one can find a schedule S' for the whole graph G and M in time and space $O(n+e)$ that has the following form*:

(i) *if* $p \geqq |L(G)|$ *then S' is at most as long as S*;

(ii) *if* $p < |L(G)|$ *then S' is longer than S and has idle periods only in its last slot.*

Proof. We run the MERGE algorithm on the following input parameters:

H is the high-graph and L the low-graph of G;

q is the number of components of $H(G)$ and $H_1, \cdots, H_q$ are the components of $H(G)$;

$r = m - q$ and $L_1, \cdots, L_{r-1}$ are some $r-1$ principal components of $L(G)$;

L_r is the remaining subgraph of $L(G)$ after removing $L_1, \cdots, L_{r-1}$.

Note that $h(H_i) > h(L)$, for $1 \leqq i \leqq q$, since H consists of all components of G that have height higher than the median of G, and L consists of all components which are at most as high as the median. By property M1 of the median we know that $H(G)$ has less than m components, and therefore $q < m$. Note that $H(G)$ might be empty and $q = 0$. Property M2 says that G has at least m components of height $\mu(G, m) - 1$. This implies that $L_1, \cdots, L_r$ exist and that $h(L_i) \geqq h(L(G)) - 1$, for $1 \leqq i \leqq r$. Note that $h(L(G)) \leqq \mu(G)$. It is easy to see that the input parameters can be found in time $O(n+e)$. Using Lemma 3.2 the proof is completed. □

We are now ready to present the main result of this section, the reduction theorem. It shows that finding an optimal schedule for G and M reduces to finding an optimal schedule for $H(G)$ and M.

THEOREM 3.2 (the reduction theorem). *Let G be a graph and M be a profile of breadth m. Then given an optimal schedule for the high-graph of G and M, the* MERGE *algorithm finds an optimal schedule for the whole graph G and M in time and space $O(n+e)$.*

Proof. Let S be the given optimal schedule for $H(G)$ and M. In Theorem 3.1 we showed that with the MERGE algorithm one can find a schedule S' for G and M in time and space $O(n+e)$ which has the following form:

(i) if $p \geqq |L(G)|$, then $\lambda(S') \leqq \lambda(S)$;

(ii) $p < |L(G)|$, then $\lambda(S') > \lambda(S)$ and S' has idle periods only in its last slot.

We want to show that the optimality of S for $H(G)$ and M implies the optimality of S' for G and M. Every schedule that has only idle periods in its last slot is optimal. Therefore, if $p < |L(G)|$ then S' is optimal. In the case $p \geqq |L(G)|$, S' is at most as long as S. An optimal schedule for G and M has to be at least as long as an optimal schedule for $H(G)$ and M, since $H(G)$ is a closed subgraph of G. Thus in the case $p \geqq |L(G)|$ we get $\lambda(S') = \lambda(S)$ and the optimality of S implies the optimality of S'. □

The following corollary of the reduction theorem implies that in the case where $H(G)$ is empty finding an optimal schedule is linear.

COROLLARY 3.1. *If $H(G)$ is empty then* HLF *is optimal for G and M and an* HLF *schedule can be found in time and space $O(n+e)$.*

Proof. The "empty schedule" is an optimal schedule for the empty graph $H(G)$ and M. The MERGE algorithm (applied as in Theorem 3.1) produces an arbitrary HLF schedule. Such a schedule has idle periods only in its last slot and is therefore optimal. □

The fact that HLF is optimal in the case where $H(G)$ is empty is also implied by the elite theorem of [DW84a].

The following theorem shows that the length of an optimal schedule is determined by the high-graph and the cardinality of the low-graph. The structure of the low-graph is not important.

THEOREM 3.3. *Let G and I be graphs such that $H(G, m) = H(I, m)$ and $|L(G, m)| =$*

$|L(I, m)|$. Let M be a profile of breadth m. Then the optimal schedules for G and M and for I and M have the same length.

Proof. Let S be some optimal schedule for $H(G)=H(I)$ and M. Let p be the number of idle periods in the schedule S. Note that all optimal schedules for $H(G)=H(I)$ and M have the same number of idle periods, since they have the same length and since they contain the same vertices.

In Theorem 3.1 we showed that with the MERGE algorithm one can find a schedule S' for G and M whose length only depends on the relationship between p and $|L(G)|$. In the same way we can find a schedule $\bar{S}$ for I and M by "merging" S with $L(I)$. Since S is a schedule for $H(G)=H(I)$ and since $|L(G)|=|L(I)|$ we have that $\lambda(S')=\lambda(\bar{S})$. In the reduction theorem we showed that both S' and $\bar{S}$ are optimal for G and I, respectively. This completes the proof of the theorem. □

In the following theorem we show which subsets of the set of initial vertices of a graph start an optimal schedule for this graph. Iterating this theorem we can find an optimal schedule for the whole graph. The elite theorem of [DW84a] is a stronger version of this theorem.

THEOREM 3.4. *Let G be a graph, M be a profile of breadth m and I be the set of initial vertices of $H(G, m)$. If there exists a schedule for G and M then:*

Case $|I|>m_1$. *There exists a set, R of m_1 vertices of I which starts some schedule for $H(G)$ and M, and for any such set R there exists a schedule for G and M starting with R.*

Case $|I|\leqq m_1$. *For any set T of $m_1-|I|$ highest initial vertices of $L(G)$ there exists a schedule for G and M starting with $I\cup T$.*

Proof. We first show that if there exists a schedule for G and M, then there exists a schedule for $H(G)$ and M that has min $(m_1, |I|)$ vertices in its first slot. Let S be a schedule for G and M. By removing the vertices of $L(G)$ from S we get a schedule $\bar{S}$ for $H(G)$ that fits M. Now, if the first slot of $\bar{S}$ has idle periods and not all vertices of I are scheduled in the first slot of $\bar{S}$, then we can move vertices of I from higher slots to the first slot of $\bar{S}$. We keep on doing this until either the first slot becomes filled up or all the vertices of I are scheduled in the first slot. The resulting schedule has the form we are looking for. It has min $(m_1, |I|)$ vertices in its first slot.

Case $|I|>m_1$. Let S be a schedule for $H(G)$ and M starting with a set R of m_1 vertices of I. As shown above such a schedule always exists. We now "merge" S with $L(G)$ as done in Theorem 3.1. The schedule S' for G and M constructed by the MERGE algorithm starts also with R, since Steps 2.2 and 2.3 are redundant for $k=1$. To see that the schedule S' for G fits the profile M we observe that there exists a schedule for G and M and therefore there is enough "space" in the profile M.

Case $|I|\leqq m_1$. Let S be a schedule for $H(G)$ and M starting with the set I. We showed already that such a schedule exists. Let T be a set of $m_1-|I|$ highest initial vertices of $L(G)$. We again "merge" S with $L(G)$ as in Theorem 3.1. The MERGE algorithm constructs a schedule S' for G and M that starts with I and a set of $m_1-|I|$ highest initial vertices of $L(G)=L$. Note that Step 2.2 is redundant for $k=1$, since $(S)_1$ contains all initial vertices of $H(G)$. Assume the set T is chosen at Step 2.3 as a set of $m_1-|I|$ highest initial vertices of $L(G)$. Then S' starts with $I\cup T$, which we wanted to show. □

4. Level orders.

In this section, we present a polynomial algorithm for finding an optimal schedule in the case where the graph is a level order of q components (where q is a positive constant) and a profile of unbounded breadth m. Our algorithm runs in time $O(m^q n^q)$ and uses space $O(n^q)$.

By property M1 of the median we know that $H(G, m)$ has less than m components. Therefore, combining the $O(m^q n^q)$ algorithm with the reduction theorem 3.2 we get the following result: An optimal schedule for a graph G, such that $H(G, m)$ is a level order, and a profile of constant breadth $m \geqq 3$ can be found in time and space $O(n^{m-1})$.

Level orders are a proper subclass of the class of series parallel digraphs [TL79] which in turn is a proper subclass of the class of totally interacting digraphs [Go76]. In [Go76] it was shown that HLF produces an optimal schedule if the graph is totally interacting and the profile is straight and of breadth two. It is an easy exercise to see that this result holds for nonstraight profiles of breadth two. Note that HLF does not produce an optimal schedule if $m = 2$ and the graph is arbitrary. In this case, restricted forms of HLF produce an optimal schedule [CG72], [Ga80a].

HLF also produces an optimal schedule for a single level order component (in linear time). By applying the reduction theorem we obtain an $O(n+e)$ time bound for any graph whose high-graph consists of at most one level order component. On the other hand, neither HLF nor restricted HLF produce an optimal schedule even if the whole graph is a level order of two components and the profile is straight and of breadth three. In Fig. 4.1, we give an example to show this. An optimal schedule S for this graph and the straight profile of breadth three is: $\{11, 10', 9'\}$, $\{10, 8', 7'\}$, $\{9, 6', 5'\}$, $\{8, 7, 4'\}$, $\{6, 5, 3'\}$, $\{4, 3, 2'\}$, $\{2, 1, 1'\}$. Note that this schedule has no idle periods, while any HLF schedule will have idle periods in its second slot.

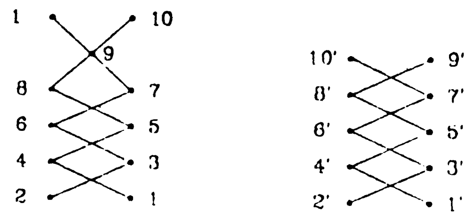

FIG. 4.1. HLF *is not optimal for three processors.*

To describe some special properties of level orders we use the following definitions: Given two graphs $G = (V, E)$ and $G' = (V', E')$. Then G is (*transitively*) *isomorphic* to G' if and only if there exists a bijective function f: $V \to V'$, such that for all vertices x and y of V, we have the following: x precedes y in G if and only if $f(x)$ precedes $f(y)$ in G'. Note that the fact that f is bijective implies that $|V| = |V'|$. Two vertices x and y of the same graph G are isomorphic to each other if and only if x maps into y in an isomorphism of G onto itself. Many closed subgraphs of a level order are isomorphic. This is the main reason why scheduling a constant number of level order components is polynomial. There will be only a polynomial number of possible closed subgraphs that we need to handle in the algorithm.

LEMMA 4.1. *Let G be a level order with one component. Then all closed subgraphs of G that have the same number of vertices are transitively isomorphic.*

Proof. We want to show that all closed subgraphs of G with k vertices ($k \leqq n$) are isomorphic. Let h be the maximum height such that G has less than k vertices of height smaller than h. Let n_k be the number of vertices in G of height smaller than h. It is easy to see that every closed subgraph of G with k vertices is of height h, it contains all vertices of G of height smaller than h, and $k - n_h$ initial vertices of height h. This completes the proof since within each component of a level order graph all vertices of the same level are isomorphic. □

We now apply Lemma 4.1 to level orders with q components.

LEMMA 4.2. *Let G be a level order with components $H_1, \cdots, H_q$. If I and J are two closed subgraphs of G containing the same number of vertices from each component, then I is transitively isomorphic to J.*

Proof. Let $I_1, \cdots, I_q$ be the subgraphs of I that contain all vertices of I from components $H_1, \cdots, H_q$, respectively. Define $J_1, \cdots, J_q$ similarly. Since I and J are closed subgraphs of G, we conclude that I_r and J_r are closed subgraphs of H_r for $1 \leqq r \leqq q$. The graphs I and J contain the same amount of vertices from each H_r, that is, $|I_r| = |J_r|$. By Lemma 4.1 we conclude that I_r is isomorphic to J_r. This implies that I is isomorphic to J. □

DEFINITION 4.1. Let G be a level order with components $H_1, \cdots, H_q$ and I be a closed subgraph of G. Then I is *represented by the tuple* $\langle n_1, \cdots, n_q\rangle$ if I contains n_r vertices of H_r, $1 \leqq r \leqq q$.

In the following corollary we rewrite Lemma 4.2 using this definition.

COROLLARY 4.1. *Let G be a level order with the components $H_1, \cdots, H_q$. If two subgraphs of G are represented by the same tuple, then they are isomorphic. All closed subgraphs of G correspond to $O(n^q)$ distinct tuples.*

Proof. The first part of the corollary follows directly from Lemma 4.2 and Definition 4.1.

Every closed subgraph of G can be represented by some tuple $\langle n_1, \cdots, n_q\rangle$. By Definition 4.1 we know that $0 \leqq n_r \leqq |H_r|$, for $1 \leqq r \leqq q$. Since $|H_r| \leqq n$, all closed subgraphs of G can be represented by at most $(n+1)^q$ tuples. Clearly, $(n+1)^q \leqq (q+1)n^q$; since q is constant, this implies that $(n+1)^q = O(n^q)$. □

The above corollary describes the key property of level orders that guarantees the polynomial algorithms. A level order with q components contains at most $O(n^q)$ equivalence classes of closed subgraphs. During the scheduling algorithm, we will keep track of all of these closed subgraphs via dynamic programming.

The following length function is used recursively in the polynomial algorithms we present later.

DEFINITION 4.2. Let G be a graph. Denote by $\lambda(G, m)$ the length of an optimal schedule for G fitting the straight profile of breadth m.

LEMMA 4.3. *The length function λ can be calculated by the following recursive formula:*

$$\lambda(\phi, m) = 0;$$

$$\lambda(G, m) = 1 + \min(\{\lambda(G-R, m) \mid R \text{ is a set of initial vertices of } G,\ 1 \leqq |R| \leqq m\}). \tag{4.1}$$

Proof. The proof is clear from the following fact. Let R be any set of initial vertices of G such that $1 \leqq |R| \leqq m$. Then $\lambda(G-R, m) = \lambda(G, m) - 1$ if and only if there exists an optimal schedule for G fitting a straight profile of breadth m with R in its first slot. □

We will now apply the recursive formula (4.1) to evaluate λ for all closed subgraphs of a level order.

LEMMA 4.4. *Let G be a level order with a constant number q of components. Then $\lambda(I, m)$ can be evaluated for all closed subgraphs I of G in time $O(m^q n^q)$ and space $O(n^q)$.*

Proof. The following algorithm evaluates $\lambda(G)$ via dynamic programming, within time $O(m^q n^q)$.

```
ALGORITHM 4.1.
1. $\lambda(\phi)=0$
2. for $k := 1$ to $n$ do
   2.1. for all closed subgraphs $I$ of $G$ with $k$ vertices do
        2.1.1. $\lambda(I) := 1+\min(\{\lambda(I-R) |\ R$ is a set of initial vertices of $I$ and
               $1 \leqq |I| \leqq m\})$.
```

The correctness follows from Lemma 4.3. To obtain the time bound we need to show more explicitly how we gather the information during the execution of the algorithm. Let $H_1, \cdots, H_q$ be the components of a level order graph. For every component H_r, denote by TOP (p, r) the number of initial vertices in the closed subgraph of H_r that contains p vertices. By Lemma 4.1 all such closed subgraphs are isomorphic. Furthermore, a level order component is completely specified by a sequence of natural numbers specifying how many vertices are in each level. Therefore, TOP (p, r), for every $0 \leqq p \leqq |H_r|$ and $1 \leqq r \leqq q$, can be created in linear time.

By Corollary 4.1, all closed subgraphs of G can be represented by $O(n^q)$ equivalence classes. Each equivalence class is determined by a vector $n = (n_1, n_2, \cdots, n_q)$ in $|H_1| \times |H_2| \times \cdots \times |H_q|$, where n_i is the number of vertices from component H_i. For every such a vector $\bar{n}$ denote by $\mathscr{R}(\bar{n}, m)$ the set of all vectors $\bar{n}'$ obtained from $\bar{n}$ by removing for every i, $n_i - n_i'$ initial vertices from the closed subgraph of H_i (represented by n_i), such that $1 \leqq \sum_{r=1}^{q} (n_r - n_r') \leqq m$ and $n_i - n_i' \leqq \text{TOP}(n_i, i)$.

Note that $n_r - n_r' \leqq m$, which implies that $|\mathscr{R}(\bar{n}, m)| = O((m+1)^q)$. Clearly $(m+1)^q \leqq (q+1)m^q$; since q is a constant, we have $|\mathscr{R}(\bar{n}, m)| = O(m^q)$.

Algorithm 4.1 can be rewritten as follows:

```
ALGORITHM 4.1'.
1'. $\lambda(\phi)=0$
2'. for $k := 1$ to $n$ do
    2.1'. for all $\bar{n} \in |H_1| \times |H_2| \times \cdots \times |H_q|$, such that $\sum_{r=1}^{q} n_r = k$ do
          2.1.1'. $\lambda(\bar{n}) := 1+\min(\{\lambda(\bar{n}') | n' \in \mathscr{R}(\bar{n}, m)\})$.
```

At Step 2.1', we partition all $O(n^q)$ tuples according to the number of vertices they contain. This can be done in time $O(n^q)$. To implement Step 2.1.1', we make use of the data structure for TOP (p, r). Since there are $O(m^q)$ choices for $\bar{n}'$, we get the time bound $O(m^q n^q)$, which completes the proof of the time bound. The algorithm needs $O(n^q)$ space to represent all equivalence classes. The array TOP and all remaining data structures require only $O(n)$ space. □

Having analyzed the function λ for all closed subgraphs of a level order G with q components, we can retrieve an optimal schedule for G and the straight profile of breadth m.

THEOREM 4.1. *Let G be a level order with a constant amount $q \geqq 2$ of components. An optimal schedule for G and the straight profile of breadth m can be found in time $O(m^q n^q)$ and space $O(n^q)$.*

Proof. Lemma 4.4 implies that we can prepare the values $\lambda(I, m)$ for all closed subgraphs I of G in time $O(m^q n^q)$ and space $O(n^q)$. We use these values to find an optimal schedule for G.

```
ALGORITHM 4.2.
1. $k := 0$
2. while $G$ is nonempty do
   2.1. $k := k+1$
```

```
2.2. Let $R$ be a subset of initial vertices of $G$ such that $1 \leqq |R| \leqq m$ and
     $\lambda(G-R)=\lambda(G)-1$
2.3. $S_k := R$
2.4. $G := G-R$.
```

If we use in Step 2.2 the vector representation for the closed subgraphs then it is easy to see that the total number of different R we scan in Step 2.2 is bounded by $O(m^q)$. Clearly every other step is bounded by $O(q)$. This implies that the total time complexity for both algorithms (Algorithm 4.1 and Algorithm 4.2) is $O(m^q n^q)$. □

We extend Theorem 4.1 to nonstraight profiles of breadth m. For that we need to refine the definition of the function λ.

DEFINITION 4.3. Let G be a graph and $M=(m_1, \cdots, m_d)$ be a profile of breadth m. Then

$$\lambda(G, M) := \min(\{k \mid \text{there exist a schedule for } G \text{ and } (m_{d-k+1}, \cdots, m_d)\}).$$

Note that k is the length of the profile $(m_{d-k+1}, \cdots, m_d)$. Note also that if m is straight, then Definition 4.3 degenerates to Definition 4.2.

LEMMA 4.5. *The function λ can be expressed recursively as follows*:

$$\lambda(\phi, M)=0;$$

$$\lambda(G, M)=1+\min(\{\lambda(G-R, M) \mid R \text{ is a set of initial vertices of } G \text{ and } 1 \leqq |R| \leqq m_{d-\lambda(G-R,M)}\}). \tag{4.2}$$

Proof. The proof follows directly from the following observations:

(i) $\lambda(G, M)$ is undefined if and only if there exists no schedule for G fitting M.

(ii) If $\lambda(G, M)$ is defined, then there exists a set R such that $1 \leqq |R| \leqq m_{d-\lambda(G,M)+1}$, and $\lambda(G-R, M)=\lambda(G, M)-1$.

(iii) If $\lambda(G, M)$ is defined, then for any set R, such that $|R| \leqq m_{d-\lambda(G,M)+1}$ and $\lambda(G-R, M)=\lambda(G, M)-1$, there exists a schedule for G fitting the profile $(m_{d-\lambda(G,M)+1}, \cdots, m_d)$ which starts with R. □

As in Lemma 4.4 we evaluate the function λ for all closed subgraphs of the level order G achieving the same time bound as for straight profiles.

LEMMA 4.6. *Let G be a level order with a constant amount q of components, and let M be a profile of breadth m. Then $\lambda(I, M)$ can be evaluated for all closed subgraphs I of G in time $O(m^q n^q)$ and space $O(n^q)$.*

Proof. As with Lemma 4.4 and Algorithm 4.1, the only change is in Step 2.1. We replace the recursive formula for straight profiles (4.1) by the recursive formula for arbitrary profiles (4.2). Since $m_{d-\lambda(G-R,M)+1} \leqq m$, we get the same time bound. □

Knowing the function λ we are ready to retrieve a schedule in a similar fashion as in Theorem 4.1 and Algorithm 4.2.

THEOREM 4.2. *Let G be a level order with a constant amount of components, and let M be a profile of breadth m. Then a schedule for G and $M'=(m_{d-\lambda G,M)+1}, \cdots, m_d)$ can be found in time $O(m^q n^q)$ and space $O(n^q)$.*

Proof. Applying Lemma 4.6 we find $\lambda(I, M)$ for all closed subgraphs I of G. To retrieve a schedule for G and M' we use Algorithm 4.2 of Theorem 4.1. Let b be $\lambda(G, M)$, where G is the original graph and M the profile. Since M is not necessarily straight, we change the bound of $|R|$ from m to m_{d-b+k}. □

The *reversed graph* G^R of a graph G is a graph obtained by reversing all the edges in G. For a profile $M=(m_1, m_2, \cdots, m_d)$ we define the *reversed profile* M^R to be $M^R=(m_d, m_{d-1}, \cdots, m_1)$. The reversed schedule S^R is defined accordingly.

In the subsequent corollary we apply Theorem 4.2 on the reversed graph G^R and reversed profile M^R to find an optimal schedule for G and M.

COROLLARY 4.3. *Let G, q and M be defined as in Theorem 4.2. An optimal schedule for G and M has length $\lambda(G^R, M^R)$ and can be found in time $O(m^q n^q)$ and space $O(n^q)$.*

Proof. Rewriting Definition 4.3 for the case where the arguments of λ are the graph G^R and the profile $M^R = (m_d, m_{d-1}, \cdots, m_1)$ we get the following:

$$\lambda(G^R, M^R) = \min(\{k \mid \text{there exists a schedule for } G^R \text{ and } (m_k, m_{k-1}, \cdots, m_1)\}).$$

Reversing the graph G^R and the profile $(m_k, m_{k-1}, \cdots, m_1)$ the above formula can be rewritten as:

$$\lambda(G^R, M^R) = \min(\{k \mid \text{there exist a schedule for } G \text{ and } (m_1, \cdots, m_k)\}).$$

We conclude that $\lambda(G^R, M^R)$ is the length of an optimal schedule for G and M.

To prove the second part of the corollary we observe that when G is a level order G^R is also. Applying Theorem 4.2 to G^R and M^R we get a schedule S for G^R and M'^R in time $O(m^q n^q)$, where M'^R is the profile $(m_{\lambda(G^R, M^R)}, \cdots, m_1)$. Since $\lambda(G^R, M^R)$ is the length of an optimal schedule for G and M, we conclude that S^R is an optimal schedule for G and M. □

Note that $\lambda(G, M)$ is not the length of an optimal schedule for G and M. It is also not the length of an optimal schedule for G and M' (defined as in Theorem 4.2). We could have defined $\lambda(G, M)$ as the length of an optimal schedule for G and M replacing Definition 4.3. Then the recursive formula corresponding to (4.2) would be:

$$\lambda(\phi, M) = 0$$

$$\lambda(G, M) = 1 + \min(\{k \mid \lambda(H, M) = k, \text{ where } H \text{ is a subgraph of } G \text{ obtained by removing at least one and no more than } m_{k+1} \text{ terminal vertices}\}), \tag{4.3}$$

where a *terminal* vertex is a vertex with no successors.

The scheduling algorithms described in the paper obtain optimal schedules by iteratively removing sets of initial vertices from the remaining graph and scheduling them in the first, second, $\cdots$ time slot (for instance, see Algorithm 4.2). Formula (4.3) corresponds to doing the scheduling process "backwards": Iteratively remove sets of terminal vertices from the remaining graph and schedule them in the last, second to last, $\cdots$ time slot. We choose the standard way of scheduling—that is, to iteratively remove sets of initial vertices—even though scheduling "backwards" would make Corollary 4.3 unnecessary.

Combining Corollary 4.1 with Theorem 3.2 we prove the final result of this section.

COROLLARY 4.4. *Let G be a graph such that $H(G, m)$ is a level order and M a profile of constant breadth $m \geqq 3$. Then an optimal schedule for G and M can be found in $O(n^{m-1})$ time and space.*

Proof. Let q be the number of components of $H(G, m)$. If $q = 0$, then by Corollary 3.1 an optimal schedule for G and M can be found in time and space $O(n+e) \sim O(n^{m-1})$, since $m \geqq 3$. If $q \geqq 1$, then Corollary 4.3 shows that an optimal schedule for $H(G, m)$ and M can be found in time $O(m^q n^q)$ and space $O(n^q)$. Property M1 of the median implies that $q < m - 1$; therefore $O(m^q n^q) = O(m^{m-1} n^{m-1})$, which equals $O(n^{m-1})$, since m is constant.

So far we have shown that an optimal schedule for $H(G)$ and M can be found in time and space $O(n^{m-1})$. By the reduction theorem, we conclude that an optimal schedule for the whole graph G and M can be found within the same time and space bounds. □

5. Inforests and outforests. In this section we give polynomial algorithms for finding an optimal schedule if the precedence graph is an inforest, an outforest or an opposing forest and the profile has constant breadth $m \geqq 3$. For inforest we present an $O(n^{m-1})$ algorithm, for outforest an $O(n^{m-1} \log n)$ algorithm and an $O(n^{2m-2} \log n)$ algorithm for opposing forest. All three algorithms require $O(n^{m-1})$ space. The algorithm for opposing forest assumes that the profile is straight, whereas the algorithms for inforest and outforest work for arbitrary profile.

A profile M is called *nondecreasing* (respectively, nonincreasing) if $m_i \leqq m_{i+1}$ (respectively, $m_i \geqq m_{i+1}$), for $1 \leqq i \leqq d$. The algorithms of [GJ83] for obtaining optimal schedules for inforests, outforests and opposing forests are less time efficient than ours: the algorithm for opposing forests requires $O(n^{m^2+2m-5} \log n)$ time, and the algorithms for outforests and inforests assume nondecreasing and nonincreasing profiles, respectively, and require $O(n^{m^2+m-6} \log n)$ time.

As in the case of scheduling level orders, deciding whether a feasible schedule exists for each of the three types of forests becomes NP-complete if the breadth of the profile is a variable of the problem instance [GJ83], [Ma81], [Wa81]. The corresponding problems stay NP-complete even in the following restricted cases: nondecreasing profile and outforest graph, nonincreasing profile and inforest and straight profile and opposing forest [GJ83], [Ma81], [Wa81]. In other related cases HLF produces an optimal schedule even if the breadth of the profile is unbounded: straight and inforest [Hu61] or outforest [Br81], [DW84a], nonincreasing profile and outforest [DW84a] and nondecreasing profile and inforest [DW84a]. Note that forests are special cases of series parallel digraphs [LT79] and therefore HLF produces an optimal schedule for forests and profiles of breadth 2 [Go76].

In this section we first present an algorithm for scheduling an outforest on a profile of $O(1)$ breadth. To do this we observe that there are at most $O(n^{m-1})$ choices for the high-graph of a closed subgraph of an outforest. This fact is used to define an equivalence relation on the set of all closed subgraphs of an outforest. The equivalence relation partitions this set into a polynomial amount of equivalence classes. Two subgraphs of the same equivalence class have the same high-graph and the same number of vertices in their low-graph. We then define a length function on the equivalence classes similar to the previous section. All closed subgraphs of one equivalence class have the same length. This length is related to the length of the optimal schedules of the subgraphs of the equivalence class. As in the previous section, the length function is evaluated via dynamic programming. We then use the length function in an algorithm which finds an optimal schedule for an outforest and a profile of constant breadth. This algorithm is similar to the MERGE Algorithm 3.1. To get an optimal schedule for an inforest we apply the outforest algorithm to the reversed profile and the reversed inforest, which is an outforest. Our algorithm for opposing forest is obtained by combining the inforest algorithm with a result of [GJ83].

Let T be a subset of the vertices of G, then CLOSE (T) is the closed subgraph induced by T, that is, the subgraph which contains the vertices of T and all the successors.

The following theorem implies that we have to keep track of $O(n^{m-1})$ high-graphs while scheduling an outforest. This result will imply the $O(n^{m-1})$ time and space bound for scheduling an outforest on a profile of constant breadth m.

THEOREM 5.1. *The high-graph of a closed subgraph of an outforest G and breadth m contains less than m initial vertices. All closed subgraphs of G have $O(n^{m-1})$ different high-graphs.*

Proof. Since G is an outforest, every closed subgraph J of G is, as is the high-graph of J. By property M1 of the median we know that $H(J, m)$ consists of less than m

components, which in this case are outtrees. Each outtree corresponds to exactly one initial vertex and therefore $H(J, m)$ corresponds to a set of less than m initial vertices that are incomparable with each other. On the other hand, each set of up to $m-1$ incomparable vertices of G induces an outforest which is the high-graph of some closed subgraph of G. Since m is constant, there are $O(n^{m-1})$ choices for a set of up to $m-1$ vertices. This completes the proof of the theorem. □

DEFINITION 5.1. Let J and K be two closed subgraphs of a graph G. Then the subgraphs J and K are *equivalent*, $J \equiv K$, if and only if they have the same high-graph, and the number of vertices of J and K that do not have any predecessors above the corresponding medians is the same. That is,

$$J \equiv K \text{ iff } (H(J) = H(K) \text{ and } |L(J)| = |L(K)|).$$

Theorem 3.3 and Theorem 5.1 are the motivation for the definition of the equivalence relation. From Theorem 3.3 we know that if two closed subgraphs are equivalent then the length of their optimal schedules for a given profile of breadth m is the same. Theorem 5.1 implies that if m is a constant, then there is a polynomial number of different equivalence classes.

Let J be a closed subgraph of a graph G. Then INIT(J) denotes the set of all initial vertices of J. Note that the closed subgraph J is completely determined by INIT(J) in the sense that J consists of INIT(J) plus all successors of the vertices of this set, that is $J = \text{CLOSE}(\text{INIT}(J))$. The equivalence class to which J belongs is completely specified by $H(J)$ (or INIT$(H(J))$), and $|L(J)|$. We denote the equivalence class of J as the tuple $[\text{INIT}(H(J)), |L(J)|]$. Applying this notation we get the following: A closed subgraph K of G is in the equivalence class $[I, w]$ iff $\text{INIT}(H(K)) = I$ and $|L(K)| = w$.

The length function we use in this section is a function of an equivalence class instead of a graph as in § 4.

DEFINITION 5.2. Let G be an outforest, let $[I, w]$ be an equivalence class of G, and M be a profile of breadth m. Then the length $\lambda(I, w, M)$ is the minimum k for which there exists a schedule for the members of $[I, w]$ fitting the profile $(m_{d-k+1}, \cdots, m_d)$.

The function λ is well defined because of Theorem 3.3. This theorem implies that for any two subgraphs K and J, such that $K \equiv J$, there exists a schedule for J and $M' = (m_{d-k+1}, \cdots, m_d)$ if and only if there exists one for K and M'. Notice that the length $\lambda(I, w, M)$ is undefined if and only if there exist no schedules for the closed subgraphs of $[I, w]$ that fit M.

In the following lemma we show how to calculate the value of $\lambda(I, w)$ from $\lambda(I, 0)$.

LEMMA 5.1. *Let $[I, 0]$ be an equivalence class, $M = (m_1, \cdots, m_d)$ be a profile, and p be the number of idle periods in a schedule for* CLOSE(I) *and $M' = (m_{d-\lambda(I,0)+1}, \cdots, m_d)$. Then*

$$\lambda(I, w, M) = \begin{cases} \textit{undefined} & \textit{if } \lambda(I, 0, M) \textit{ is undefined}, \\ \lambda(I, 0, M) & \textit{if } \lambda(I, 0, M) \textit{ is defined and } p \geqq w, \\ \bar{\lambda} & \textit{if } \lambda(I, 0, M) \textit{ is defined and } p < w, \end{cases}$$

where

$$\bar{\lambda} = \min\left(\left\{k \mid k \geqq 1 \textit{ and } \sum_{i=d-k+1}^{d} m_i \geqq |\text{CLOSE}(I)| + w\right\}\right).$$

Proof. Case $\lambda(I, 0, M)$ *undefined.* Since the closed subgraphs of $[I, w]$ have at least as many vertices as the closed subgraphs of $[I, 0]$, the value of $\lambda(I, w)$ is also undefined; and if $\lambda(I, 0)$ is defined, then $\lambda(I, w) \geqq \lambda(I, 0)$.

For the case where $\lambda(I, 0)$ is defined, let S be a schedule for CLOSE (I) and M'. This schedule S has p idle periods and

$$p = \left(\sum_{i=d-\lambda(I,0)+1}^{d} m_i \right) - |\text{CLOSE}(I)|.$$

Case $p \geqq w$. By Theorems 3.1 and 3.3, we know that for any graph J of $[I, w]$ there exists a schedule S' for J and M'. Note that $J \in [I, w]$ if and only if INIT $(H(J)) = I$ and $|L(J)| = w$ and S' has $p - w$ idle periods. We conclude that if $p \geqq w$, then $\lambda(I, w) = \lambda(I, 0)$.

Case $p < w$. Clearly $\lambda(I, w) \geqq \bar{\lambda}$ since the subgraphs of $[I, w]$ contain $|\text{CLOSE}(I)| + w$ vertices. If $\bar{\lambda}$ is undefined, then there is not enough "space" in the profile M to schedule a graph of $[I, w]$ and therefore $\lambda(I, w)$ is undefined also.

For the case where $\bar{\lambda}$ is defined we want to show that for each member of $[I, w]$ there exists a schedule that fits $\bar{M} = (m_{d-\bar{\lambda}+1}, \cdots, m_d)$. Since $\bar{\lambda} > \lambda(I, 0)$, the schedule S for CLOSE (I) and M' can be embedded into the profile $\bar{M}$. Therefore, there exists a schedule for CLOSE (I) and $\bar{M}$, and such a schedule has more than w idle periods, since $\sum_{i=d-\bar{\lambda}+1}^{d} m_i \geqq |\text{CLOSE}(I)| + w$. By applying Theorems 3.1 and 3.3 again, we conclude that there exists a schedule for any member of $[I, w]$ and $\bar{M}$; therefore, $\lambda(I, w) = \bar{\lambda}$, if $p < w$. □

We now want to show that the calculation of $\lambda(I, w)$ from $\lambda(I, 0)$ as described in the previous lemma can be implemented efficiently.

LEMMA 5.2. *Let G be an outforest and M be a profile of constant breadth $m \geqq 3$. Given the appropriate data structures, which can be created in time and space $O(n^{m-1})$, then for any equivalence class $[I, w]$ of G, $\lambda(I, w, M)$ can be calculated from $\lambda(I, 0, M)$ in constant time.*

Proof. The following data structures can be created in time and space $O(n^{m-1})$ and allow us to calculate $\lambda(I, w, M)$ in constant time from $\lambda(I, 0, M)$.

Data structure $A = (U, N, L)$. Let G be an outforest and M be a variable profile of breadth m.

(i) For every $x \in G$, $U[x]$ is one plus the number of successors of x in G.

(ii) For every set T of up to $m-1$ incomparable vertices of G, $U[T] = \sum_{y \in T} U[y]$. Note that if T is a set of incomparable vertices then $U[T] = |\text{CLOSE}(T)|$.

(iii) For every k, $1 \leqq k \leqq d$, $N[k]$ is the total number of available processors in the subprofile $(m_{d-k+1}, \cdots, m_d)$. That is, $N[k] = \sum_{i=d-k+1}^{d} m_i$. Note that k is the length of the subprofile.

(iv) For every r, $1 \leqq r \leqq N[d]$, $L[r]$ is the length of the shortest profile $(m_{d-k+1}, \cdots, m_d)$ having a total amount of r available processors. Therefore, $L[r] = \min(\{k \mid N[k] \geqq r\})$.

The properties of data structure A which we need for proving Lemma 5.2 are:

A1. Given the value of $\lambda(I, 0)$ then the value of $\lambda(I, w)$ can be calculated in constant time.

A2. The data structure A can be created in time and space $O(n^{m-1})$.

Proof of Property A1. By Theorem 5.1 and Definition 5.1 we know that $|I| \leqq m-1$, since G is an outforest. In Lemma 5.1 a formula was given to calculate $\lambda(I, w)$ from $\lambda(I, 0)$. Using the arrays U, N and L we can rewrite this formula in the following way:

$$\lambda(I, w) = \begin{cases} \lambda(I, 0) & \text{if } p \geqq w, \\ L[U[I] + w] & \text{if } p < w. \end{cases}$$

Furthermore, the number of idle periods p in a schedule for CLOSE (I) and profile $(m_{d-\lambda(I,0)+1}, \cdots, m_d)$ can be expressed as: $p = N[\lambda(I,0)] - U[I]$. Therefore, if data structure A is given, and $\lambda(I,0)$ and I is known, then $\lambda(I,w)$ can be calculated in constant time.

Proof of Property A2. (i) Determining the number of successors of each vertex of the outforest G can be done in one traversal of the outforest. Thus the array U can be evaluated for all vertices of G in time $O(n)$.

(ii) There are $O(n^{m-1})$ choices for a set T of up to $m-1$ vertices. For a given T the value $U[T]$ can be found in constant time, since m is constant. Therefore, U can be evaluated for all sets T in time $O(n^{m-1})$.

(iii) and (iv) The matrices N and L can easily be created in time $O(n)$.

This completes the proof of the properties of data structure A and therefore also the proof of Lemma 5.2. □

As in the previous section we give a recursive formula for the function λ. While scheduling a graph we repeatedly remove sets of initial vertices from the graph. The notation $A \xrightarrow{R} B$ denotes that B is obtained from A by removing R, which is a set of initial vertices.

Using the above notation we can give a recursive formula for $\lambda(I,0)$:

$$\lambda(I,0) = 1 + \min(\{\lambda(I',w') \mid ([I,0]) \xrightarrow{R} ([I',w']) \wedge 1 \leqq |R| \leqq m_{d-\lambda(I',w')}\}). \tag{5.1}$$

The notation $([I,0]) \xrightarrow{R} ([I',w'])$ means the following: Let J be a graph of $[I,0]$, then by removing R, which is a subset of I, from J, we obtain a subgraph J', where $J' \in [I',w']$.

The correctness of the above formula is obvious. We make all possible choices to remove sets of initial vertices and we recurse on the remaining graph. This formula is used to evaluate $\lambda(I,0)$ for all sets I of up to $m-1$ incomparable vertices of G via dynamic programming.

LEMMA 5.3. *Let G be an outforest and M be a profile of constant breadth m. Then the function λ can be evaluated for all equivalence classes $[I,0]$ of G in time and space $O(n^{m-1})$.*

Proof. The following algorithm evaluates λ for all $[I,0]$ of G in time $O(n^{m-1})$.

```
ALGORITHM 5.1.
1. $\lambda(\phi, 0) := 0$;
2. for $i = 1$ to $n$ do
   2.1. for all sets $I$ of up to $m-1$ incomparable vertices of $G$,
        such that $|\mathrm{CLOSE}(I)| = i$ do
        2.1.1. $\lambda(I,0) := 1 + \min(\{\lambda(I',w') \mid ([I,0]) \xrightarrow{R} ([I',w'])$
                                   and $1 \leqq |R| \leqq m_{d-\lambda(I',w')}\})$.
```

Proof of correctness. The correctness follows from (5.1). Notice also that at Step 2.1.1 $|\mathrm{CLOSE}(I')| < |\mathrm{CLOSE}(I)|$ since $|R| \geqq 1$. As shown in Lemma 5.1 the value of $\lambda(I',q')$ is determined by $\lambda(I,0)$ and w.

Proof of the bounds. By Theorem 5.1 and Definition 5.1 we know that for any equivalence class $[I,w]$ of an outforest G, $|I| \leqq m-1$ and I is a set of incomparable vertices. Note that $|I|$ is constant when m is constant. There are $O(n^{m-1})$ different sets of up to $m-1$ vertices of G. Claim 5.1 below implies that with an appropriate data structure, we can determine in constant time whether the vertices of a given set of cardinality up to $m-1$ are incomparable or not. Therefore, all sets of up to $m-1$ incomparable vertices of G can be found in time and space $O(n^{m-1})$. We then create data structure A in time and space $O(n^{m-1})$ and bucket sort all sets T of up to $m-1$ incomparable vertices of G according to $U[T]$. Thus, Claim 5.1 and Lemma 5.2 imply that Steps 2 and 2.1 can be implemented in time $O(n^{m-1})$.

CLAIM 5.1. *Given the preorder number and the number of successors for every vertex of G, then for any set T of up to $m-1$ vertices of G it can be determined in constant time whether T is a set of incomparable vertices or not.*

Proof of the claim. Let $p(x)$ denote the preorder number and $n(x)$ the number of successors of the vertex x of G. Now for any two vertices x and y of G, x precedes y if and only if $p(x) \leqq p(y) \leqq p(x)+n(x)$ (see [AH74] for details). To decide in constant time whether some set T of up to $m-1$ vertices of G is incomparable or not we use the following fact: T is a set of incomparable vertices if and only if for every x and y of T, x does not precede y. Since T has a constant size, this can be done in constant time, which completes the proof of the claim. □

Since the preorder number and the number of successors of every vertex can be found in $O(n)$ time the claim implies that Steps 2 and 2.1 can be implemented in time $O(n^{m-1})$. By Theorem 5.1 there are $O(n^{m-1})$ sets of up to $m-1$ incomparable vertices of G. Thus Step 2.1.1 gets executed $O(n^{m-1})$ times and to get an overall $O(n^{m-1})$ time bound we need to show that Step 2.1.1 can be implemented in constant time.

In Lemma 5.2 we showed that $\lambda(I', w')$ can be calculated in constant time given data structure A and $\lambda(I', 0)$. At Step 2.1.1 the value of $\lambda(I', 0)$ has been calculated already since $|\text{CLOSE}(I')| < |\text{CLOSE}(I)|$.

The set R is a subset of the set I and $|I| < m$. Since m is constant, there is only a constant amount of choices for R. Thus to prove that Step 2.1.1 is constant we have left to show that given a set R then $[I', w']$ can be determined in constant time. This is achieved by the following data structure.

Data structure B. The outforest G is represented by its adjacency lists [AH74], in which the immediate successors of every vertex of x are given in a linked list sorted according to decreasing height.

Properties of data structure B.

B1. Let J be a closed subgraph of G, let R be a subset of $\text{INIT}(H(J))$, and let $(\text{INIT}(H(J)), \mu(J), |L(J)|) \xrightarrow{R} (I', \mu', w')$. Then (I', μ', w') can be obtained from $(\text{INIT}(H(J)), \mu(J), |L(J)|)$ in constant time.

B2. Data structure B can be created in time $O(n)$.

Proof of Property B1. For every vertex $x \in R$, let T_x be a set of m highest immediate successors of x. If x has less than m immediate successors then let T_x be all immediate successors of x. Define T to be the following set of vertices: $T := \{\text{INIT}(H(J)) - R\} \cup (\cup_{x \in R}(T_x))$.

Obviously T can be found in $O(m^2)$ time, since the immediate successors of $x \in R$ are given in decreasing height. $|\text{INIT}(H(J))| < m$ and $|T| < m^2$. Note that $O(m^2) = O(1)$, because m is constant. Since G is an outforest every vertex of T corresponds to an outtree in J', which is the subgraph of J obtained by removing the set R from J. All the roots of height at least as high as the mth highest component of J' are contained in T. Therefore, $\mu(J') = \mu'$ is one plus the height of an mth highest vertex of T. If $|T| < m$ then μ' is set to zero. Furthermore, $I' = \text{INIT}(J')$ is a subset of T, i.e., I' is the set of all vertices of T of height bigger than μ'. Since the size of T is constant, μ' and I' can be determined in constant time. Finally w' is computed as follows: $w' = |L(J')| = U[I] + |L(J)| - U[I'] - |R|$. This completes the proof of Property B1 of data structure B.

Proof of Property B2. All vertices of G can be bucket sorted according to their height in linear time. Create the adjacency lists of G as follows: starting at the highest vertices and continue according to decreasing height, insert each vertex to the end of the adjacency list of the immediate predecessor of it (in constant time). Thus data structure B can be constructed in time $O(n)$.

To complete the proof of the time bound we still have to show that in Step 2.1.1 $[I', w']$ can be determined in constant time when I and R are given. This follows from Property B1 of data structure B. Note that at Step 2.1.1, $J = \text{CLOSE}(I) = H(J)$, $\mu(J) = 0$ and $|L(J)| = 0$. □

We are now ready to present the main result of this section.

THEOREM 5.2. *Let G be an outforest and $M = (m_1, \cdots, m_d)$ be a profile of constant breadth m. Then it can be determined in time and space $O(n^{m-1})$ whether there exists a schedule for G and M. If such a schedule exists, then we can find a schedule for G fitting the profile $(m_{d-\lambda(I,w)+1}, \cdots, m_d)$, where $G \in [I, w]$, in time $O(n^{m-1})$.*

Proof. To determine whether there exists a schedule for G and M, we apply Lemma 5.3 and evaluate λ for all equivalence classes $[I, 0]$ of G (Algorithm 5.1). This can be done in time and space $O(n^{m-1})$. Given the value of $\lambda(I, 0)$, it is easy to calculate $\lambda(I, w)$ (see Lemma 5.3). Note that $\lambda(I, w)$ is defined if and only if there exists a schedule for G and M (see Definition 5.2). Thus we showed that one can decide in time and space $O(n^{m-1})$ whether there exists a schedule for G and M.

If such a schedule exists then the following algorithm finds a schedule for G fitting the profile $M' = (m_{d-\lambda(I,w)+1}, \cdots, m_d)$, such that $G \in [I, w]$, in time and space $O(n^{m-1})$.

```
ALGORITHM 5.2.
1. $I_H := \text{INIT}(H(G))$  $I_L := \text{INIT}(L(G))$  $\mu := \mu(G)$;
   $\lambda := \lambda(I_H, |L(G)|)$
2. for $k := d - \lambda + 1$ to $d$ do
   2.1. $j := k - d + \lambda$
        if $|I_H| > m_k$
        2.1.1. then Find $R$ such that $([I_H, 0]) \xrightarrow{R} ([I', w'])$,
                    $|R| = m_k$, and $\lambda(I', w') \leqq d - k$
                    $(S)_j := R$
        2.1.2. else Find a set $T$ of $m_k - |I_H|$ highest vertices of $I_L$
                    $(S)_j := I_H \cup T$
   2.2. Determine $I'_H, I'_L, \mu'$ such that $(I_H, I_L, \mu) \xrightarrow{(S)_j} (I'_H, I'_L, \mu')$
        $I_H := I'_H$  $I_L := I'_L$  $\mu := \mu'$.
```

Proof of correctness. Assume we are before Step 2.1 and the Loop 2 has been executed already several times, that is, vertices of the original graph G have been put into the slots $1, 2, \cdots, k - d + \lambda - 1$ of S. Let G be the remaining graph at this point, that is, the closed subgraph of the original graph that has not been scheduled yet. Then applying the notation of the algorithm we have: $I_H = I(H(G))$, $I_L = I(L(G))$ and $\mu = \mu(G)$.

It is easy to see that both at Step 2.1.1 and 2.1.2 the set $(S)_j$ is a subset of the initial vertices of G. Thus in the constructed schedule S the precedence constraints specified by the graph G are not violated.

The correctness of Algorithm 5.2 is shown by proving the following loop invariant: There exists a schedule for G and $(m_k, \cdots, m_d)$.

At Step 1 we set λ to $\lambda(I_H, |L(G)|)$ and we know that this value is defined. The definition of the function λ (Definition 5.2) implies that there exists a schedule for G and $(m_{d-\lambda+1}, \cdots, m_d)$ after Step 1. Therefore, the invariant holds for $k = d - \lambda + 1$ before the first execution of Loop 2.

We now want to prove the following: If there exists a schedule for G and $(m_k, \cdots, m_d)$ before Step 2.1, then there exists a schedule for G' and $(m_{k+1}, \cdots, m_d)$ after Step 2.2, such that $(G) \xrightarrow{(S)_j} (G')$. The proof of the above implication follows from Theorem 3.4.

Case $|I_H| > m_k$. By Theorem 3.4 there exists a set R of m_k vertices of I_H that starts a schedule for $H(G)$ and $(m_k, \cdots, m_d)$. Define I' and w' such that $([I_H, 0]) \xrightarrow{R} ([I', w'])$. Since there exists a schedule for $H(G)$ and $(m_k, \cdots, m_d)$ starting with R we have $\lambda([I', w']) \leqq d - k$. So far we have shown that the set R as defined in Step 2.1.1 exists.

On the other hand, any set R that is defined as in Step 2.1.1 starts a schedule for $H(G)$ and $(m_k, \cdots, m_d)$. This is implied by the fact that $\lambda(I', w') \leqq d - k$. Define $\bar{H}$ such that $(H(G)) \xrightarrow{R} (\bar{H})$, then $\bar{H} \in [I', w']$ and there exists a schedule for $\bar{H}$ and $(m_{k+1}, \cdots, m_d)$, since $\lambda(I', w') \leqq d - k$. Note that $(m_{k+1}, \cdots, m_d)$ has length $d - k$. By Theorem 3.4 we conclude that R as defined in Step 2.1.1 starts a schedule for G and $(m_k, \cdots, m_d)$, since it starts one for $H(G)$ and $(m_k, \cdots, m_d)$. This implies that there exists a schedule for G' and $(m_{k+1}, \cdots, m_d)$, since $(S)_j = R$ and $(G) \xrightarrow{R} (G')$.

Case $|I_H| \leqq m_k$. Then by Theorem 3.4 we know that for any set T of $m_k - |I_H|$ highest vertices of I_L, there exists a schedule for G and $(m_k, \cdots, m_d)$ starting with $I_H \cup T$. This implies that there exists a schedule for G' and $(m_{k+1}, \cdots, m_d)$. Note that at Step 2.1.2 $(S)_j = I_H \cup T$ and $(G) \xrightarrow{(S)_j} (G')$. This completes the proof of the loop invariant and the proof of correctness of Algorithm 5.2.

Proof of time bound. First, we create the data structures A and B in time and space $O(n^{m-1})$. Represent I_H as a doubly linked list. Implement I_L as an array of linked lists, where the linked list $I_L(h)$ contains all vertices of I_L of the height h.

Step 1. Evaluate the function λ for all equivalence classes $[I, 0]$ of G. By Lemma 5.3 this can be done in time and space $O(n^{m-1})$. Create all the above data structures, and evaluate μ and λ in the same time bound.

Step 2. We want to show that the loop can be implemented in time $O(n)$.

Case $|I_H| > m_k$. *Step* 2.1.1. Since $|I_H| < m$ and m is constant, there is only a constant amount of subsets R of I_H such that $|R| = m_k$. For each set R we can determine in constant time whether $\lambda(I', w') \leqq d - k$. Note that we know $\lambda(I', 0)$ and therefore by Lemma 5.2, $\lambda(I', w')$ can be determined in constant time. We conclude that Step 2.1.1 can be implemented in constant time.

Step 2.2. In the case $|I_H| > m_k$. Step 2.2 can be easily implemented in overall time $O(n)$. By Property B1 of data structure B, I'_H and μ' can be determined in constant time. Note that $(S)_j \subseteq I_H$. To determine I'_L we look at all immediate successors of the vertices of $(S)_j$. If such an immediate successor has height at most μ', then we add it to the appropriate list of I_L in constant time. Since each vertex gets added exactly once to the array of list I_L, this costs overall time $O(n)$.

Case $|I_H| \leqq m_k$. *Step* 2.2.1. By property M2 of the median we know that G has at least m components of height at least $\mu - 1$. Exactly $|I_H|$ of these components have height bigger than μ and therefore, G has at least $m - |I_H| \geqq m_k - |I_H|$ components of height μ and $\mu - 1$. Thus I_L has at least $m_k - |I_H|$ vertices in the lists $I_L(\mu)$ and $I_L(\mu - 1)$, and the set T of Step 2.1.2 can be found in constant time.

Step 2.2. In the case $|I_H| \leqq m_k$ we do Step 2.2 in two steps:

(i) $(I_H, I_L, \mu) \xrightarrow{T} (I_H, \bar{I}_L, \bar{\mu})$,

(ii) $(I_H, \bar{I}_L, \bar{\mu}) \xrightarrow{I_H} (I'_H, I'_L, \mu')$.

That is, we first remove the set T and determine $\bar{I}_L$ and $\bar{\mu}$, and then we remove the set I_H and determine I'_H, I'_L, μ'. Note that $(S)_j = I_H \cup T$. The reason why we can do Step 2.2 in two steps is that $L(G)$ and $H(G)$ are disjoint.

To show that Step (i) can be done in overall time $O(n)$, we observe that T can be removed from I_L in constant time. Note that T is a set of $m_k - |I_H|$ highest vertices of lists $I_L(\mu)$ and $I_L(\mu - 1)$. To find $\bar{I}_L$ we insert all immediate successors of the vertices of T into the appropriate list of I_L. Since each vertex gets added at most once this can be done in overall time $O(n)$. To determine $\bar{\mu}$ we observe that $\bar{\mu} = \mu - 1$ if T contains all vertices of $I_L(\mu)$ and $\bar{I}_L(\mu - 1)$; otherwise $\bar{\mu} = \mu$. Note that if $\bar{\mu} = \mu - 1$ then T contains all vertices of $I_L(\mu)$. Thus $H(G)$ and therefore I_H does not change when T is removed from G.

We showed already that Step 2.2(ii) can be done in overall time $O(n)$ (see implementation of Step 2.2 in the case where $|I_H| > m_k$).

This completes the proof of the time bound of Algorithm 5.2. Note that the expensive part was to evaluate the function λ in time $O(n^{m-1})$ retrieving a schedule is linear. This also completes the proof of Theorem 5.2. □

We now apply Theorem 5.2 to find an optimal schedule for an outforest.

COROLLARY 5.1. *Let G be an outforest and M be a profile of constant breadth m. Then an optimal schedule for G and M can be found in time $O(n^{m-1} \log m)$ and $O(n^{m-1})$ space.*

Proof. We do a binary search to determine

$$\min(\{d' | d' \leqq d \text{ and there exists a schedule for } G \text{ and } (m_1, \cdots, m_{d'})\}).$$

For every $d' \leqq d$ we can, by Theorem 5.2, decide in time and space $O(n^{m-1})$ whether there exists a schedule for G and $(m_1, \cdots, m'_d)$. Since we can assume that $d \leqq n$, we have to do this $O(\log n)$ times during the binary search. This completes the proof of the $O(n^{m-1} \log n)$ time bound. □

COROLLARY 5.2. *Let G be an inforest and M be a profile of constant breadth m. Then an optimal schedule for G and M can be found in time and space $O(n^{m-1})$.*

Proof. Since G is an inforest, G^R is an outforest. We apply Theorem 5.2 to the outforest G^R and M^R and find a schedule for G^R and $(m_{\lambda([l,w])}, \cdots, m_1)$ in time and space $O(n^{m-1})$, where $[l, w]$ is the equivalence class of G^R of which G^R is an element of. Interpreting the definition of the function λ (see Definition 5.2) we see that $\lambda(l, w)$ is the length of an optimal schedule for G and M. We used the same trick to prove Corollary 4.1. □

To prove our time bound for an opposing forest we use the following result of [GJ83].

THEOREM 5.3. *A schedule for an opposing forest fitting a straight profile of breadth m and length d can be found in time $O(d^{m-1}t(n, m, d))$ and space $O(s(m, n, d) + m)$, where $t(n', m', d')$ and $s(n', m', d')$ are the time and space, respectively, that it takes to find a schedule for an inforest with n' vertices and a nondecreasing profile of constant breadth m' and length d'.*

Proof. Corollary 2.2.1 of [GJ83]. □

We now combine Corollary 5.2 with Theorem 5.3:

THEOREM 5.4. *Let G be an opposing forest and M a straight profile of constant breadth m. Then a schedule S for G fitting M can be found in time $O(n^{2m-2})$ and space $O(n^{m-1})$.*

Proof. Let $t(n, m, d)$ and $s(n, m, d)$ be defined as in Theorem 5.3. By Corollary 5.2 we know that $t(n, m, d) = O(n^{m-1})$ and $s(n, m, d) = O(n^{m-1})$. Applying Theorem 5.3 we follow that it takes time $O(d^{m-1}n^{m-1})$ and space $O(n^{m-1})$ to find a schedule for G fitting M. We can easily assume that $d < n$. Otherwise it is trivial to find a schedule for G fitting M. Using the fact that $d < n$, we get the $O(n^{2m-2})$ time bound. □

Note that Corollary 5.2 gives a more general result than we need to prove the above theorem. The $O(n^{m-1})$ time bound is for arbitrary profiles of constant breadth m and not only for nondecreasing profiles. Furthermore, in Corollary 5.2 we showed that one can find an optimal schedule in time and space $O(n^{m-1})$ and not just any schedule that fits the profile.

REFERENCES

[AH74] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[Br81] J. BRUNO, *Deterministic and stochastic scheduling problems with treelike precedence constraints*, NATO Conference, Durham, England, July 1981.

[CG72] E. G. COFFMAN, JR. AND R. L. GRAHAM, *Optimal scheduling for two-processors systems*, Acta Inform., 1 (1972), pp. 200–213.

[Co76] E. G. COFFMAN, JR., ed., *Computer and Job Shop Scheduling Theory*, John Wiley, New York, 1976.

[DW84a] D. DOLEV AND M. K. WARMUTH, *Scheduling flat graphs*, Reseach Report 84-04, Hebrew Univ., Jerusalem, March 1984.

[DW84b] ——— , *Scheduling precedence graphs of bounded height*, J. Algorithms 5 (1984), pp. 48–59.

[FK71] M. FUJII, T. KASAMI AND K. NINOMIYA, *Optimal sequencing of two equivalent processors*, SIAM J. Appl. Math., 17 (1969), pp. 784–789; *Erratum*, 20 (1971), p. 141.

[Ga82] H. N. GABOW, *An almost linear algorithm for two processor scheduling*, J. ACM, 29 (1982), pp. 766–780.

[Ga81] ———, *A linear-time recognition algorithm for interval dags*, Inform. Proc. Lett., 12 (1981), pp. 20–22.

[GJ79] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[GJ83] M. R. GAREY, D. S. JOHNSON, R. E. TARJAN AND M. YANNAKAKIS, *Scheduling opposing forests*, this Journal, 4 (1983), pp. 72–93.

[GL79] R. L. GRAHAM, E. L. LAWLER, J. K. LENSTRA AND A. H. G. RINNOOY KAN, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Ann. Discr. Math., 5 (1979), pp. 287–326.

[Go76] D. K. GOYAL, *Scheduling series parallel structured tasks on multiprocessor computing systems*, Technical Report CS-76-034, Dept. of Computer Science, Washington State Univ., Pullman, September, 1976.

[Hu61] N. C. HU, *Parallal sequencing and assembly line problems*, Oper. Res., 9 (1961), pp. 841–848.

[LR76] J. K. LENSTRA AND A. H. G. RINNOOY KAN, *Complexity of scheduling under precedence constraints*, Oper. Res., 26 (1976), pp. 22–25.

[LT79] E. L. LAWLER, R. E. TARJAN AND J. VALDES, *The recognition of series parallel digraphs*, Proc. 11th Annual Symposium on Theory of Computing, Atlanta, GA, April 30–May 2, 1979, pp. 1–12.

[Ma81] E. W. MAYR, *Well structured parallel programs are not easier to schedule*, Technical Report STAN-CS-81-880, Dept. of Computer Science, Stanford Univ., Stanford, CA, September, 1981.

[PY79] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Scheduling interval-ordered tasks*, SIAM J. Comput., 10 (1979), pp. 405–409.

[Ul75] J. D. ULLMAN, *NP-complete scheduling problems*, J. Comput. System Sci., 10 (1976), pp. 384–393.

[Wa81] M. K. WARMUTH, *Scheduling on profiles of constant breadth*, Ph.D. Thesis, Dept. Computer Science, Univ. Colorado, Boulder, August 1981.