SIAM J. COMPUT.
Vol. 14, No. 3, August 1985

© 1984 Society for Industrial and Applied Mathematics
010

SCHEDULING FLAT GRAPHS*

DANNY DOLEV† AND MANFRED WARMUTH‡

Abstract. The problem of scheduling a partially ordered set of unit length tasks on m identical processors is known to be NP-complete. There are efficient algorithms for only a few special cases of this problem. In this paper we analyze the effect of the structure of the precedence graph and the availability of the processors on the construction of optimal schedules. We prove that to find an optimal schedule it suffices to consider at each step only initial tasks which belong to the $m-1$ highest components of the precedence graph. This result reduces the number of cases we have to check during the construction of an optimal schedule. Our method leads to polynomial algorithms if the number of processors is fixed and the precedence graph has a certain form. In particular, if the precedence graph contains only intrees and outtrees, this result leads to linear algorithms for finding an optimal schedule on two or three processors.

Key words. identical processors, profile, optimal schedule, intree and outtree

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. One of the fundamental problems in deterministic scheduling is to schedule a set of unit length tasks, subjected to precedence constraints, on a system of identical processors. The precedence constraints between tasks are represented by a *precedence graph*, which is a directed acyclic graph. As in [GJ81] we allow the number of identical processors to vary with time. A *profile* is a sequence of natural numbers specifying how many processors are available at each time slot. A schedule for a given profile is a partitioning of all the tasks into a sequence of sets which does not violate the precedence graph. The ith set of the sequence is scheduled in the ith time slot (i.e. interval $[i-1, i)$). Thus, the cardinality of the ith set cannot exceed the number of processors which are available in the ith time slot of the profile. A profile is *straight* if it has the same number of processors available at each time slot. The *breadth* of a profile is the maximum number of processors available at any time slot.

Various aspects of scheduling theory have been studied extensively in recent years [GL79] and many scheduling problems are known to be NP-complete [Ul75], [GJ79], [LR78], [Wa81], [GJ81], [Ma81]. 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 some given precedence constraints is NP-complete, if the number of available processors is a variable of the problem. Notice that the breadth of the profile is not bounded by a constant. The problem remains NP-complete even for certain classes of precedence graphs [GJ81], [Ma81], [Wa81]. To support the idea that the breadth of the profile is the main source of NP-completeness we prove in [DW82b] that scheduling unit length tasks is NP-complete even if the precedence graph has height one and the profile has one processor available in each slot except for one slot that has an arbitrary number (see Table 1.1). Polynomial algorithms have been developed for only a few special cases. The first polynomial algorithm was developed

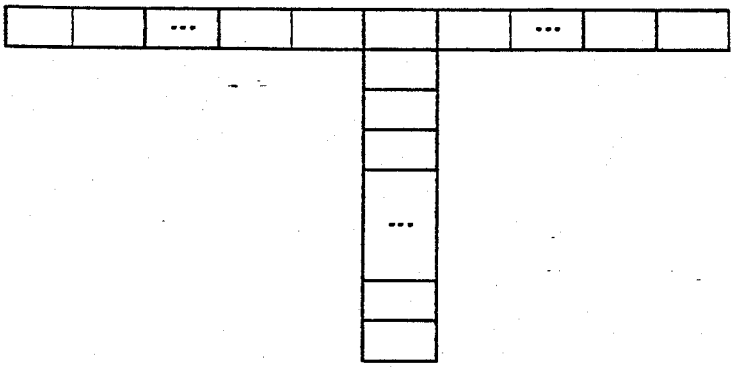
* Received by the editors December 23,1980, and in final revised form March 23, 1984. This paper is a revision of IBM Research Report RJ3398, 1982.

† Institute of Mathematics and Computer Science, Hebrew University, Jerusalem, Israel. Part of this work was done while this author visited IBM Research, San Jose, California.

‡ Computer Science Department, University of California, Santa Cruz, California 95064. Part of this work was done while this author visited the Hebrew University, Jerusalem. The research was supported by the Fulbright Commission of West Germany, grants from Univac Corporation and Storage Technology Corporation, and by the United States-Israel Binational Science Foundation under grant 2439/82.

TABLE 1.1

The question of existence of a schedule for a precedence graph of height one and a profile of the below form is NP-*complete.*



by Hu [Hu61]. It produces an optimal schedule for a straight profile of arbitrary breadth if the precedence graph is an inforest. 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 tasks of the same level are chosen arbitrarily. HLF also produces an optimal schedule for outforests and straight profiles of arbitrary breadth [Br81]. A restricted version of HLF provides an optimal schedule when the precedence graph is an interval order [PY79], [Ga82], or if the number of available processors is two [Ga81]. Recently, polynomial algorithms have been published [GJ81], [Wa81], [DW82c] for scheduling certain classes of precedence graphs on profiles of fixed breadth. In [Wa81], [DW82a] it was also shown that scheduling an arbitrary graph on a profile of fixed breadth is polynomial, if the height of the graph is bounded by a constant.

The major scheduling problem remaining open is whether the scheduling of an arbitrary graph of unbounded height is NP-complete or polynomial for a fixed number ($m \geqq 3$) of processors.

Let m be the breadth of the profile. The median (see § 3) of the precedence graph is defined to be one plus the height of the mth highest component of the precedence graph (see Fig. 3.1) and if the precedence graph contains less than m components, then the median is zero. A task is *initial* if it does not have any predecessors. The *Elite* of the precedence graph is the set of all initial tasks that belong to components that are higher than the median.

Our main result, the Elite theorem (§ 4), states that it is enough to choose tasks from the Elite of the precedence graph for the first slot of an optimal schedule. If we do not have enough tasks in the Elite, then we choose tasks according to highest height from the set of initial tasks that are not in the Elite. After filling the first slot, the Elite theorem can be applied to the remaining precedence graph and the next slot, and so on. The theorem restricts the number of cases that need to be considered for constructing an optimal schedule. Variations of the Elite theorem are the basis for the algorithms in [Wa81], [DW82c].

In § 5, we generalize results of [Hu61], [Br81], and [GJ81] by applying the Elite theorem. We show that HLF produces an optimal schedule if the precedence graph is either an inforest or an outforest and the profile is of a certain type. In § 6 we prove some properties of graphs containing only inforest and outforest components (opposing forests). In § 7 we use these properties to develop a linear algorithm for scheduling an opposing forest on a straight profile of breadth three improving the $O(n \log n)$ time

bound of Garey et al. [GJ81].[1] Furthermore, we give an $O(n \log n)$ algorithm for scheduling an opposing forest on a profile that has two or three processors available at any time slot. The algorithm is essentially the one described in [Do80].

2. Basic definitions and properties. A precedence graph G is denoted by a tuple (V, E), where V is the set of n *tasks* and E the set of *edges* of G. A (*directed*) *path* π of length r in G is a sequence of tasks $x_0, \cdots, x_r$ such that the edge (x_i, x_{i+1}), for $0 \leqq i \leqq r-1$, is in E. We assume that if a task x has to be executed before a task y, then there exists a (directed) path from x to y in G. Note that G is acyclic.

If there exists a path from x to y, then x is a *predecessor* of y, and y is a *successor* of x. In the case where the longest path from a task x to a task y is the edge (x, y), we call x the *immediate predecessor* of y and y the *immediate successor* of x.

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

The graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$, denoted by $G' \subseteq G$, if $V' \subseteq V$ and for all x and y in G', x is a predecessor of y in G' if and only if x is a predecessor of y in G. A subgraph G' of G is called a *closed subgraph* if every task in G' has the same successors in G' as it has in G. For two graphs $G = (V, E)$ and $G' = (V', E')$, $G \cup G'$ denotes the graph $(V \cup V', E \cup E')$. The graph $G = (V, E)$ is composed of $\{G_1, \cdots, G_r\}$ if these subgraphs (called *components* of G) are a decomposition of G into its connected components, that is, each subgraph is a nonempty connected graph and there are no edges between tasks of different components; therefore, $G = \bigcup_i G_i$. A task of G is *initial* if it has no predecessors. Note that an initial task of G is not necessarily of maximum height in G. A set of k *highest initial tasks* is a subset of the set of initial tasks consisting of some k highest ones; when there are less than k initial tasks, it contains all of them. Let R be a set of initial tasks of a precedence graph G. Then $G - R$ is the subgraph of G obtained by removing the tasks of R.

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*, M, is a sequence of positive integers, $(m_1, m_2, \cdots, m_d)$, specifying the number of identical processors, m_i, that are available in each time slot i, for $i \leqq i \leqq d$ (see Table 2.1); d is the *length* of the profile M. The *breadth* of profile M is the maximum number of processors that is available at any time slot of M. Throughout the paper we denote the breadth of the given profile with the letter m. The profile of Table 2.1 has breadth three. We call a profile M *straight* if $m_i = m$, for all $1 \leqq i \leqq d$.

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 tasks 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 S, denoted by $\lambda(S)$, is the index of the last nonempty set in the sequence. A minimum 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 $(S)_i$ gets executed in the ith time slot, that is $|(S)_i|$ of the m_i processors of slot i are executing the tasks of $(S)_i$ during the time interval $[i-1, i)$. Note that the length of a task equals the length of a time slot. We call the schedule S an *M-schedule* for G.

[1] In a revised version of [GJ81] Garey et al. also obtain a linear algorithm.

As an example assume we have a set of twelve tasks subject to the precedence aph G presented in Fig. 2.1. We name the tasks by numbers. Throughout the paper e always assume that the edges of the graphs are directed downwards. We look for schedule for G that fits the profile $M = (2, 3, 3, 1, 3, 2)$. The following sequence S is valid schedule:

$$\{1, 2\}|\{3, 4, 5\}|\{6, 7\}|\{8\}|\{9, 10, 11\}|\{12\}.$$

he M-schedule S can be shown as in Table 2.1. Notice that $\lambda(S) = 6$. In Table 2.2 a chedule S' of length 5 is given for the same precedence graph, which is optimal.

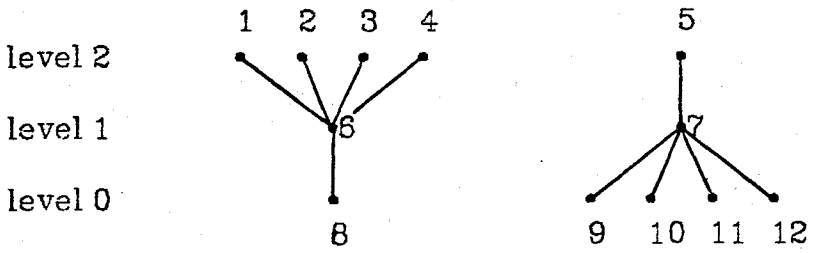


FIG. 2.1. *The precedence graph G.*

TABLE 2.1
The schedule S for the precedence graph G of Fig. 2.1 and the profile M = (2, 3, 3, 1, 3. 2).

slot	1	2	3	4	5	6
P_1	1	3	6	8	9	12
P_2	2	4	7		10	
P_3		5			11	
m_i	2	3	3	1	3	2

TABLE 2.2
The schedule S' for G and M.

slot	1	2	3	4	5	6
P_1	1	2	4	6	8	
P_2	5	3	9		11	
P_3		7	10		12	
m_i	2	3	3	1	3	2

The ith slot of a schedule S, $1 \leqq i \leqq \lambda(S)$, has $m_i - |(S)_i|$ *idle periods.* Such an idle eriod corresponds to a processor being idle during time slot i of S.

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 ighest initial tasks of the subgraph of G induced by all tasks scheduled in slot i of or later. Note that in the above example S is a HLF-schedule, whereas S' is not. LF-schedules have the following property. Assume task x is scheduled in slot i and is scheduled in slot j. If $h(x) > h(y)$, then either $i \leqq j$ or there is a predecessor of x 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 arbtrarily among initial tasks of the same height.

A schedule for G is greedy if whenever there is an idle period in some slot i then this slot contains all initial tasks of the subgraph of G induced by the tasks that appear in slot i or later. It is easy to see that any schedule can be made into a greedy one without increasing its length; thus there exists greedy schedules which are optimal.

3. The median. In this paper we study graphs that have more than m components ("flat" graphs), where m is the breadth of the profile. We use the notion of the median to characterize this property of a precedence graph.

DEFINITION. The *median* of a precedence graph G with respect to a given breadth m, denoted by $\mu(G)$, is one plus the height of the mth highest component of the precedence graph.

Thus the graph of Fig. 3.1 has median 3 with respect to $m=3$. If the graph has fewer than m components the median is 0. For example, in the graph described by Fig. 2.1 the median with respect to $m=3$ is 0.

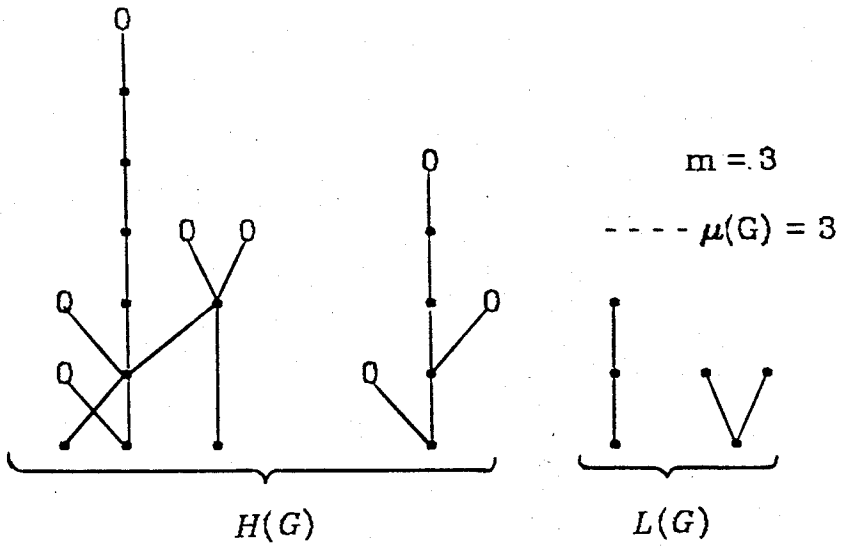


FIG. 3.1. *The decomposition of a graph G into H(G) and L(G); 0 denotes tasks of E(G).*

We use the median to partition the components of G into two sets, $H(G)$ and $L(G)$ (see Fig. 3.1).

DEFINITION. The closed subgraph $H(G)$ consists of all components of height higher than the median, and $L(G)$ contains all the components that are at most as high as the median.[2] The set of all initial tasks of $H(G)$ is called the *Elite* of G, denoted by $E(G)$.

During the construction of a schedule, the median is a dynamic line. When a set of initial tasks is removed, the median might increase, because some components of the graph might split into several components. On the other hand, the median can drop at most by one. If it drops by one, then some initial tasks of $L(G)$ were removed. If only tasks of $H(G)$ are removed, then the median does not drop. This leads to the following properties which are used in the current paper and in [DW82c].

[2] All the results of this paper will still hold if we define $H(G)$ to be the closed subgraph that contains all initial tasks of height higher than the median plus all their successors, and $L(G)$ to be the remaining subgraph. See also [DW82b].

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 ;t $\mu(G)-1$.

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 $\mu(G)\geqq h$.

M5: Let R be a set of initial tasks of G. Then $\mu(G-R)\geqq\mu(G)-1$.

M6: Let R be a subset of $E(G)$. Then $\mu(G-R)\geqq\mu(G)$. Furthermore, $H(G-$ $\subseteq H(G)-R$ and $L(G-R)\supseteq L(G)$.

M7: Let T be a set of highest initial tasks of $L(G)$. Then $H(G-T)\subseteq H(G)$ and $G-T)\supseteq L(G)-T$.

M8: Let R be a subset of $E(G)$ and T be a set of highest initial tasks of $L(G)$. Then

$$H(G-(R\cup T))\subseteq H(G)-R \text{ and } L(G-(R\cup T))\supseteq L(G)-T.$$

ıe proofs of the properties M1 through M4 follow directly from the definition of edian. The proof of M8 is a simple consequence of M6 and M7. To prove the maining properties the following claim is needed.

CLAIM 3.1. *Let I be a component of G and let R be a set of initial tasks of G. Then*

$$h(I)\geqq h(I-R)\geqq h(I)-1.$$

Proof. The claim trivially holds if $h(I)=0$. Thus assume that $h(I)$ is positive. The et R contains only initial tasks of I; therefore, the longest path in $I-R$ is by at most ıne shorter than the longest path of I. Thus, $h(I-R)\geqq h(I)-1$ and clearly, $h(I)\geqq$ ı$(I-R)$, which completes the proof. □

Proof of M5.

M5. Let R be a set of initial tasks of G. Then $\mu(G-R)\geqq\mu(G)-1$.

M5 is clearly true if $\mu(G)\leqq 1$. Thus assume that $\mu(G)>1$. By property M4 we ɔnly have to show that $G-R$ contains at least m components having height at least $\mu(G)-2$. To do this observe that by property M2 the graph G contains at least m components of height at least $\mu(G)-1$. By Claim 3.1, $h(I-R)\geqq\mu(G)-2$ for every component I of G that satisfies $h(I)\geqq\mu(G)-1$. Therefore, the subgraph $I-R$ of $G-R$ has at least one component of height at least $\mu(G)-2$, and $G-R$ has at least m components of height at least $\mu(G)-2$.

Proof of M6.

M6. Let R be a subset of $E(G)$. Then $\mu(G-R)\geqq\mu(G)$. Furthermore, $H(G-$ $R)\subseteq H(G)-R$ and $L(G-R)\supseteq L(G)$.

The second part of M6 is a simple consequence of the fact that $\mu(G-R)\geqq\mu(G)$. Readily this inequality holds if $\mu(G)=0$. So assume $\mu(G)$ is positive. To prove that $\mu(G-R)\geqq\mu(G)$ we need to show that $G-R$ contains at least m components of height $\mu(G)-1$ (see property M4). If I is a component with $h(I)\leqq\mu(G)$, then I is in $L(G)$ and therefore $I=I-R$. Also, if $h(I)>\mu(G)$, then by Claim 3.1, $h(I-R)\geqq$ $\mu(G)$. This completes the proof, because by property M2, G contains at least m components I, satisfying $h(I)\geqq\mu(G)-1$, and for each such component I the corresponding subgraph $I-R$ of $G-R$ contains at least one component of height at least $\mu(G)-1$.

Proof of M7.

M7: Let T be a set of highest initial tasks of $L(G)$. Then $H(G-T)\subseteq H(G)$ and $L(G-T)\supseteq L(G)-T$.

Assume that M7 does not hold for some G and T. Then $\mu(G-T)<\mu(G)$, and by M5, $\mu(G-R)=\mu(G)-1$. For the median to drop, T must contain an initial task

of $L(G)$ of height $\mu(G)-1$. Since T is a set of highest initial tasks of $L(G)$ it follows that T contains all tasks of $L(G)$ of height $\mu(G)$ (which are all initial). Therefore, we conclude that $L(G-T)\subseteq L(G)-T$ and thus $H(G-T)\supseteq H(G)$. But this contradicts the assumption. □

4. The Elite theorem. In this section we present our main result, the Elite theorem. Let $M=(m_1,\cdots,m_d)$ be a given profile of breadth m. The Elite theorem states that to find an optimal schedule for G it suffices to "look" at the Elite of G. In particular, if the cardinality of the Elite is larger than m_1, then there exists an optimal M-schedule for G that starts with a subset of the Elite. Otherwise, there exists an optimal M-schedule starting with $E(G)$ and $m_1-|E(G)|$ highest initial tasks from $L(G)$, choosing arbitrarily among tasks of the same height.

The Elite theorem enables us to ignore large portions of the graph at each step of the construction of an optimal schedule. As a special case, the Elite theorem also implies that if there is no initial task above the median, then HLF produces an optimal schedule.

Results similar to the Elite theorem were developed in [Wa81] and [DW82c]. They are the basis for several polynomial algorithms which find optimal schedules for certain restricted classes of precedence graphs and profiles of constant breadth. The Elite theorem is easily derived from the following theorem.

THEOREM 4.1. *Let S be a greedy M-schedule for $H(G)$ not longer than the length of an optimal M-schedule for G. Let f be the number of idle processors at the first slot of S. For any set T of f highest initial tasks of $L(G)$, there exists an optimal M-schedule S' for G with the properties*:

i) $(S')_1=(S)_1\cup T$;

ii) *if $\lambda(S')>\lambda(S)$ then S' has idle periods only in its last slot.*

Proof. The proof is by induction on r, the number of tasks of G of positive height. In the case $r=0$ the graph does not contain any tasks of positive height and therefore, all the tasks of G are initial and the theorem obviously holds.

Assume that the theorem holds for every precedence graph of fewer than $r+1$ tasks of positive height and let G be a graph with $r+1$ such tasks. We distinguish between two cases, according to T', the set of initial tasks in $L(G)$.

Case $|T'|<f$. Thus, the number of initial tasks in $L(G)$ is less than f which is the number of idle processors at the first slot of S. In this case $(S)_1$ contains all the initial tasks of $H(G)$, since we assumed that S is greedy. Furthermore, $T=T'$ and $(S)_1\cup T$ is the set of all initial tasks of G. This implies that G contains fewer than $m_1\leqq m$ components; therefore, $\mu(G)=0$, all the tasks in $L(G)$ are initial and $L(G)=T$. Thus, the schedule

$$S'=((S)_1\cup T)|(S)_2|\cdots|(S)_{\lambda(S)}$$

for G has the same length as S, which implies the optimality of the schedule S' for G and M.

Case $|T'|\geqq f$. Let λ be the length of an optimal M-schedule for G, then by assumption, $\lambda(S)\leqq\lambda$. Let T be a set of f highest initial tasks of $L(G)$. Denote $\bar{G}=G-((S)_1\cup T)$ and let S^* be the schedule obtained from $(S)_2|\cdots|(S)_{\lambda(S)}$ by removing all tasks not in $H(\bar{G})$ and making the resulting schedule greedy. Clearly $\lambda(S^*)\leqq\lambda-1$.

By M8, $H(\bar{G})\subseteq A(G)-(S)_1$, which assures that S^* contains all the tasks of $H(\bar{G})$. Denote by $\bar{\lambda}$ the length of an optimal schedule for $\bar{G}$ that fits $\bar{M}=(m_2,\cdots,m_d)$. Since

the length of an optimal schedule for G, $\lambda - 1 \leqq \bar{\lambda}$, and therefore $\lambda(S^*) \leqq \bar{\lambda}$. eover, the graph $\bar{G}$ contains fewer than $r+1$ tasks of positive height, because $\cup T$ contains at least one task of positive height. Thus, the inductive hypothesis be applied to S^*, ensuring the existence of $\bar{S}$, an optimal *$\bar{M}$-schedule* for $\bar{G}$, with property of having idle processors in its last slot in the case $\lambda(\bar{S}) > \lambda(S^*)$.

Define $S' = ((S)_1 \cup T)|\bar{S}$. We have to prove that S' is an optimal M-schedule for and that it satisfies ii). Consider the following two subcases:

a) If $\lambda(S) \leqq \lambda(S')$, then S' is clearly an optimal M-schedule for G, since by mption S is not longer than the length of an optimal M-schedule for G.

b) Otherwise, $\lambda(S') = \lambda(\bar{S}) + 1 > \lambda(S)$. By the definition of S^* we get $\lambda(S) \geqq$ $^*) + 1$, which implies $\lambda(\bar{S}) > \lambda(S^*)$. Now, the inductive assumption guarantees that as idle processors only in its last slot. Since $(S)_1 \cup T$ has no idle processors, this plies that S' is an optimal M-schedule for G with idle processors only in its last t. This completes the proof. □

Now we are ready to prove the basic result of the paper.

THEOREM 4.2. The Elite theorem. *Let G be a precedence graph and M be a profile breadth m. Then*

i) *If $E(G)$ contains more than m_1 tasks, then there exists an optimal schedule for and M that starts with m_1 initial tasks of $E(G)$.*

ii) *If $E(G)$ contains m_1 tasks or fewer, then any set of m_1 highest initial tasks of is a first slot of some optimal M-schedule for G.*

iii) *If $E(G) = \varnothing$, then* HLF *produces an optimal schedule for G and M that has le periods only in its last slot.*

Proof. The proof follows from Theorem 4.1 and the fact that if $|E(G)| \leqq m_1$, then ere exists an optimal schedule for $H(G)$ and M that starts with $E(G)$; otherwise ere exists an optimal schedule which starts with a subset of $E(G)$ of size m_1. If (G) is empty, then $H(G)$ is empty and the empty sequence is an optimal schedule r $H(G)$. □

We will demonstrate the Elite theorem on a few examples. Assume we need to hedule the precedence graphs of Figs. 4.1 and 4.2 on the profile $M = (3, 3, 2, 1, 3)$, hich has breadth three.

In Fig. 4.1, $\mu(G_1) = 0$ and therefore, in finding an optimal schedule, we start by hoosing $m_1(3)$ of the four tasks that are above the median. These four tasks are the Elite of the graph. Not every subset of three tasks of the Elite begins an optimal schedule. For example, if we choose $\{1, 2, 3\}$ as the first slot we would not get an optimal schedule. By the Elite theorem we know that there exists a set of three tasks, among the tasks in the Elite, that starts an optimal schedule. In G_1, $\{1, 2, 5\}$ are such asks.

If the precedence graph is the one given in Fig. 4.2, the situation is much simpler: here exists an optimal schedule starting with $E(G_2)$, and if we remove this set from the graph we obtain the graph described in Fig. 4.3. The Elite of this graph is empty $(E(G_2') = \varnothing)$ and thus, as we proved in the Elite theorem, HLF produces an optimal schedule for this graph. Therefore, the tableau T_2' of Table 4.1 describes an optimal schedule for G_2', and T_2 describes an optimal schedule for the whole graph G_2.

LEMMA 4.1. *Let G' be a closed subgraph of G. If the length of any* HLF*-schedule for G and M is bounded by λ, then the length of any* HLF*-schedule for G' and M is also bounded by λ.* □

The following results follow from the Elite theorem and Theorem 4.1, and are useful in showing that HLF produces an optimal schedule for certain classes of precedence graphs and profiles.

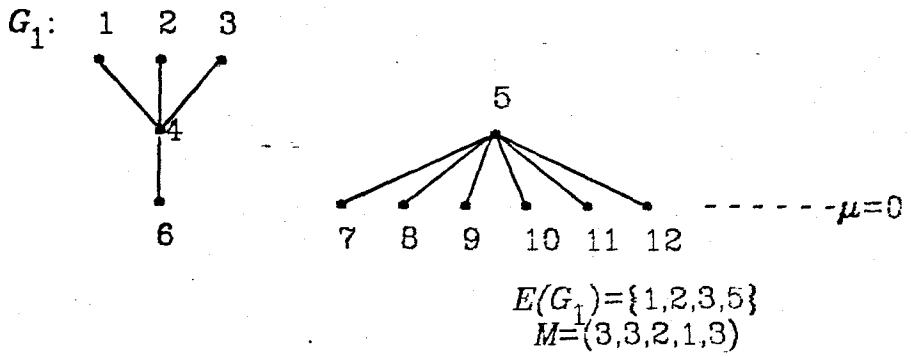


FIG. 4.1

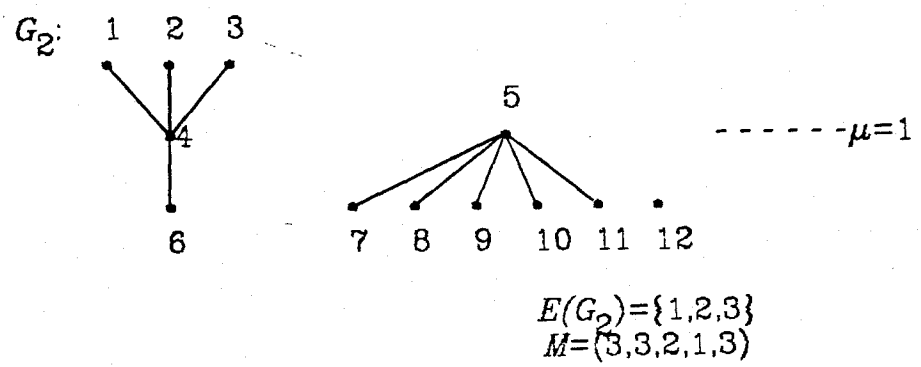


FIG. 4.2

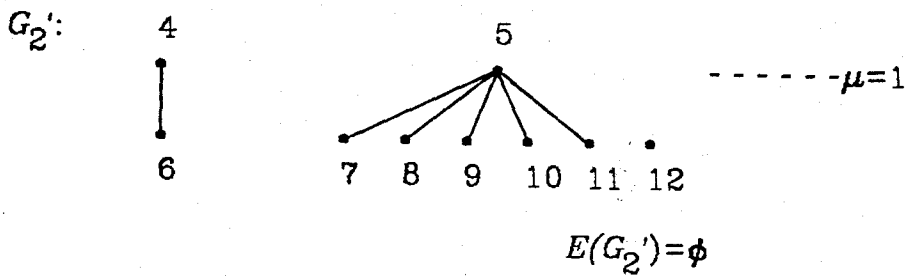


FIG. 4.3

TABLE 4.1.
The profile schedules for the graph G_2' and G_2.

	slot	2	3	4	5
T_2':	P_1	4	6	8	9
	P_2	5	7		10
	P_3	12			11

	slot	1	2	3	4	5
T_2:	P_1	1	4	6	8	9
	P_2	2	5	7		10
	P_3	3	12			11

LEMMA 4.2. *Let λ be the length of an optimal M-schedule for G. If λ bounds the length of every* HLF*-schedule for $H(G)$ and M, then* HLF *produces an optimal M-schedule for G. In particular, if* HLF *is optimal for $H(G)$, then* HLF *is optimal for G.*

Proof. The proof is by induction on the number of tasks of positive height in G. The case of no task of positive height is trivial. Assume that the lemma holds for every precedence graph of up to k tasks of positive height and let G be one with $k+1$ such tasks. Let T be any set of m_1 highest initial tasks of G. Denote by $G=G-T$ the remaining subgraph, and by M' the remaining profile. It is enough to show that;

a) There exists an optimal M-schedule for G starting with T.

b) HLF produces an optimal M'-schedule for G'.

Proof of a). If $|E(G)| \leqq m_1$, then by the Elite theorem there exists an optimal M-schedule for G starting with T. Otherwise $T \subseteq E(G)$, and by assumption it starts a schedule for $H(G)$ and M of length at most λ. Theorem 4.1 implies that it also starts an optimal schedule for G and M.

Proof of b). The subgraph G' contains fewer than $k+1$ tasks of positive height because T contains some. Since T starts an optimal M-schedule for G the length of an optimal schedule for G' is $\lambda - 1$. λ bounds the length of every HLF-schedule for $H(G)$ and M. Therefore $\lambda - 1$ bounds the length of every HLF-schedule for $H(G) - T$ and M'. By property M8 of the median, $H(G') \subseteq A(G) - T$, and thus, by Lemma 4.1, the length of every HLF-schedule for $H(G')$ and M' is bounded by $\lambda - 1$. Using the inductive assumption we conclude that HLF produces an optimal schedule for G' and M'. □

5. HLF for inforest and outforest. In this section we study inforests and outforests, and analyze whether HLF produces an optimal schedule for such precedence graphs. An *inforest* (respectively *outforest*) is a graph in which each task has at most one immediate successor (respectively one immediate predecessor). HLF is optimal for specific types of profiles.

A profile M is *nondecreasing* (resp. *nonincreasing*) if $m_i \leqq m_{i+1}$ (resp. $m_i \geqq m_{i+1}$), for $1 \leqq i \leqq d-1$; that is, the number of available processors does not decrease (resp. not increase) along the profile.

The results obtained in this section hold for more general types of profiles. We will see that if the profile is nondecreasing or nonincreasing but its amplitude of variation is bounded by one, then the complexity of the algorithms does not change.

We say that M is a *zigzag* profile if the following two conditions hold:

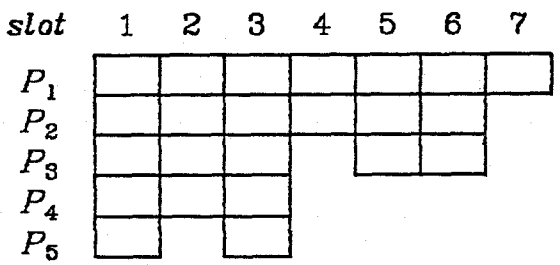
i) $m_i + 1 \geqq m_j$, for all ij such that $1 \leqq i \leqq j \leqq d$.

ii) $m_j + 1 \geqq m_i$, for all ij such that $1 \leqq i \leqq j \leqq d$.

If condition i) holds, then M is called a *nonincreasing zigzag profile*; if ii) holds, then it is called *nondecreasing zigzag profile.*

The profile of Table 2.1 is neither a nonincreasing nor a nondecreasing zigzag profile, since $m_4 - m_5 = -2$ and $m_4 - m_3 = -2$, respectively. In Table 5.1 we give an example of a nonincreasing zigzag profile; the profile is $M = (5, 4, 5, 2, 3, 3, 1)$. It is a nonincreasing but not a nondecreasing zigzag profile, since $m_7 - m_3 = -4$.

TABLE 5.1
A nonincreasing zigzag profile.



Three HLF results for forests have appeared in the literature. The first and basic one is by Hu [Hu61] who showed that HLF produces an optimal schedule for inforests and straight profiles. Bruno [Br81] proved that HLF is optimal for outforests and straight profiles. The third result is by Garey et al. [GJ81]; they proved that HLF works for inforests and nonincreasing profiles in which $m_1 - m_d = 1$. We prove that HLF is

optimal if the precedence graph is an inforest (resp. outforest) and the profile is nondecreasing zigzag (resp. nonincreasing zigzag). Note that scheduling an inforest (resp. outforest) on a nonincreasing (resp. nondecreasing) profile is NP-hard if the breadth of the profile is arbitrary [GJ81], [Ma81], [Wa81] and polynomial if the breadth of the profile is constant [GJ81], [Wa81], [DW82c].

THEOREM 5.1. *Let G be an outforest and M be a nonincreasing zigzag profile of breadth m. Then* HLF *produces an optimal M-schedule for G.*

Proof. It suffices to prove the following: let H be an outforest, M be a nonincreasing profile of breadth m; then for any set of m_1 highest initial tasks of H, there exists an optimal M-schedule starting with this set. Note that if H contains less than m_1 initial tasks, then the set of all initial tasks are the only set of m_1 highest initial tasks.

We prove the theorem by applying the Elite theorem and using the inequality

$$(*) \qquad |E(G)| \leqq m-1 \leqq m_1.$$

The inequality $m-1 \leqq m_1$ holds because M is a nonincreasing zigzag profile of breadth m. By the definition of $H(G)$, it has fewer than m components. Each component of $H(G)$ is an outtree having one initial task, and this task is the only task from that outtree in the Elite of G. This implies that the Elite of G has fewer than m tasks and $(*)$ holds. □

We now prove a similar result for inforests. Let G be a precedence graph; denote by G_z the subgraph of G obtained by removing all tasks of height zero. Observe that an optimal schedule for G_z is at least by one shorter than an optimal schedule for G, since the last slot of any schedule can only have tasks of height zero.

THEOREM 5.2. *Let G be an inforest and M be a nondecreasing zigzag profile of breadth m. Then* HLF *produces an optimal M-schedule for G.*

Proof. Assume to the contrary that the theorem does not hold, and let G be an inforest with a minimal number of tasks, such that HLF is not optimal (for some nondecreasing zigzag profile of breadth m). If $L(G) \neq \emptyset$, then $H(G)$ contains less tasks than G. Thus, HLF is optimal for $H(G)$, and by Lemma 4.2 it is also optimal for G. This proves that the minimality of G requires that $L(G) = \emptyset$, and that G contains at most $m-1$ components.

Let M be any nondecreasing zigzag profile of breadth m and let λ be the length of an optimal schedule for G and M. Assume that M has length λ. Notice that $m_\lambda \geqq m-1$, i.e., all tasks of G of height zero fit into the last slot of M. The minimality of G implies that HLF is optimal for G_z and M. But every HLF-schedule for G and M can be obtained from an HLF-schedule for G_z and M by scheduling all tasks of height zero in the last slot of M (which was empty) and making the resulting schedule greedy. The HLF-schedules for G and M are of length λ and therefore optimal. This is a contradiction. □

Theorems 5.1 and 5.2 can be further improved using Lemma 4.2. Lemma 4.2 proves that it is enough to require that only $H(G)$ will be an outforest (resp. inforest) for Theorem 5.1 (resp. Theorem 5.2) to hold.

The following examples show that Theorems 5.1 and 5.2 are tight. Table 5.2 presents an HLF-schedule for the inforest G_1 of Fig. 5.1 that fits the profile $M_1 = (3, 3, 1, 1, 1)$ but is not optimal. Notice that M_1 is not a nondecreasing zigzag profile. Table 5.3 presents a nonoptimal HLF-schedule for the outforest G_2 of Fig. 5.2 that fits the profile $M_2 = (1, 2, 3, 3, 3, 3, 3)$. Notice that M_2 is not a nonincreasing zigzag profile. If one starts with task 3, then it is easy to find an optimal schedule, which is shorter than the schedule of Table 5.3.

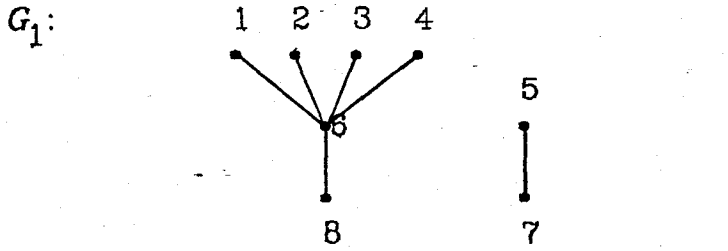


FIG. 5.1

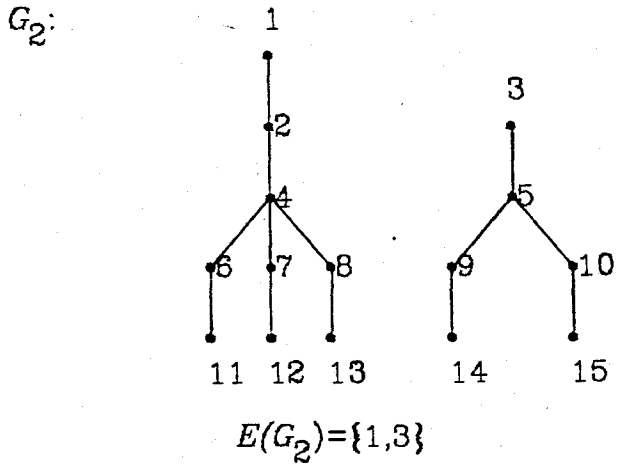


FIG. 5.2

TABLE 5.2
A HLF*-schedule for the graph of* G_1
of Fig. 5.1 *and the profile* $M = (3, 3, 1, 1, 1)$.

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

TABLE 5.3
A HLF*-schedule for the graph* G_2 *of Fig.* 5.2 *and the profile* $M_2 = (1, 2, 3, 3, 3, 3, 3)$.

slot	1	2	3	4	5	6	7
P_1	1	2	4	6	9	12	15
P_2		3	5	7	10	13	
P_3				8	11	14	

6. Opposing forests. We say that a graph is an *opposing forest graph* if all its components are intrees or outtrees, that is, it is composed of an inforest and an outforest. It was shown that HLF is optimal for scheduling an inforest [Hu61] or an outforest [Br81] on a straight profile with arbitrary breadth. On the other hand scheduling an opposing forest on straight profiles with arbitrary breadth is NP-hard [GJ81], [Ma81], [Wa81].

If the profile is straight and its breadth m is fixed, then scheduling an opposing forest is polynomial [GJ81], [Wa81], [DW82c]. These algorithms have rather complex time bounds (m appears in the exponent). We use the results of this section, which are derived from the Elite theorem, to obtain a linear algorithm for the special case of straight profiles of breadth three and opposing forests. This improves the $O(n \log n)$ time bound of an algorithm presented in [GJ81] for this case. Our approach also leads to an $O(n \log n)$ algorithm for scheduling an opposing forest on a zigzag profile of breadth three (either two or three processors in each time slot).

Goyal [Go76] proved that HLF is optimal for series-parallel graphs [LT79] and straight profiles of breadth two. Since opposing forests are series-parallel graphs, HLF is also optimal for opposing forests and straight profiles of breadth two. In the case of scheduling opposing forests we can generalize Goyal's result to any profile of breadth two.

THEOREM 6.1. *Let G be an opposing forest*[3] *and M be a profile of breadth two. Then* HLF *produces an optimal M-schedule for G.*

Proof. In case of breadth two $H(G)$ contains at most one component. Therefore, $H(G)$ is either an intree or an outtree. Note that any profile of breadth two is a zigzag profile. By Theorem 5.1, Theorem 5.2, and Lemma 4.2 we conclude that HLF produces an optimal schedule for G. □

Note that Theorem 6.1 holds also for graphs that are not opposing forests whose highest component is either an intree or an outtree. It is easy to see that for arbitrary graphs and zigzag profiles of breadth two the Coffman-Graham algorithm [CG72] produces an optimal schedule. This algorithm corresponds to a restricted version of HLF. For profiles of breadth three choosing tasks according to highest height does not necessarily lead to optimal schedules.

For example, HLF does not produce optimal schedules for the graph of Fig. 6.1 and the straight profile of breadth three. Task 8 must be scheduled in an earlier slot than task 7. But always preferring the outtree tasks does not lead to an optimal schedule either. We need a criterion that tells us when to prefer outtree tasks over intree tasks. Such a criterion will be provided by a theorem proven below: If there is an initial outtree task x of the same height as the whole opposing forest G, then any set of three highest tasks of G that contains x starts an optimal schedule for G.

Notice that the above criterion does not apply to the graph of Fig. 6.1, since task 8 is not a task of maximum height. In this case, we "flip" the graph. Let G^R denote the graph obtained by reversing all the edges of G. Note that for an opposing forest, either G or G^R contains an outforest task of height $h(G)$.

In the example (Fig. 6.1 and Table 6.1) we can remove the set {12, 17, 18}. Since we remove this triplet from the reversed graph, we schedule it in the last slot of the schedule. The remaining subgraph is shown in Fig. 6.2.

Now both the top and the bottom contain an outtree of maximum height. We can choose either of the sides to continue the algorithm. Assume we choose {9, 15, 16} and

[3] Actually the theorem holds for any series-parallel graph. This can be proven by a simple induction on the size of the graph using Lemma 4.2.

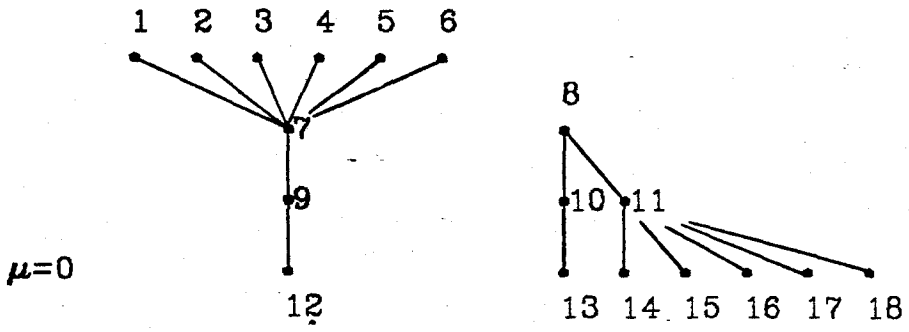


FIG. 6.1

TABLE 6.1
First step.

slot	1	2	3	4	5	6
P_1						12
P_2						17
P_3						18

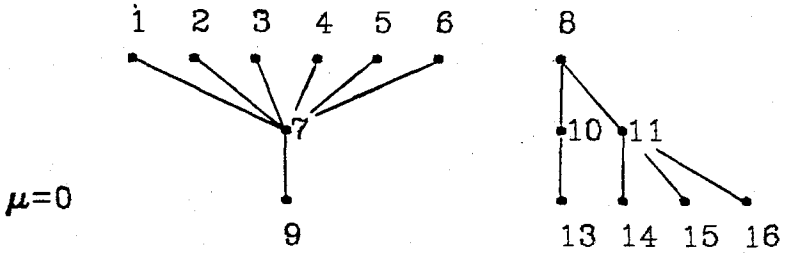


FIG. 6.2

TABLE 6.2
Second step.

slot	1	2	3	4	5	6
P_1					9	12
P_2					15	17
P_3					16	18

put them in the next slot from the bottom. We are left with the subgraph of Fig. 6.3. At this step we have to switch back to the top, because the reversed graph does not have an outtree of maximum height.

We remove {8, 1, 2} from the graph and insert them in the first slot of the schedule. We now obtain the graph in Fig. 6.4.

The Elite of this graph is empty and therefore by the Elite theorem we know that HLF produces an optimal schedule for the resulting graph. We complete the schedule according to HLF filling slots 2, 3 and 4 and obtain Table 6.4.

The above example demonstrates the Flip-Flop algorithm which will be presented in the next section. A high-level description of the algorithm is given at the beginning of the next section.

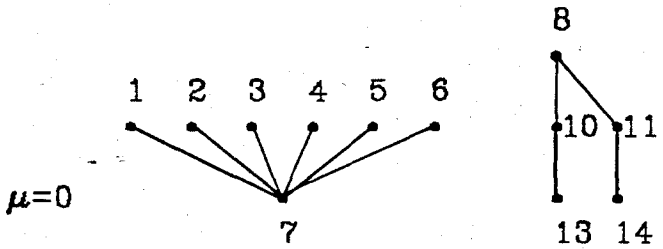


FIG. 6.3.

TABLE 6.3
Third step.

slot	1	2	3	4	5	6
P_1	8				9	12
P_2	1				15	17
P_3	2				16	18

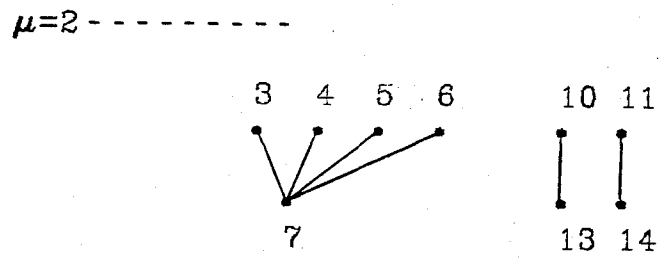


FIG. 6.4.

TABLE 6.4
Last step.

slot	1	2	3	4	5	6
P_1	8	3	6	7	9	12
P_2	1	4	10	13	15	17
P_3	2	5	11	14	16	18

The following lemma proves that among tasks of the same height, we can always schedule tasks of an outtree first.

LEMMA 6.1. *Let S be an optimal M-schedule for G. Assume that y is an initial task of an intree component and that x is an initial task of an outtree component, such that $h(x) \geqq h(y)$. If $y \in (S)_1$ and $x \notin (S)_1$ then there exists an optimal M-schedule, $\hat{S}$, for G that starts with x instead of y, that is,*

$$(\hat{S})_1 = ((S)_1 - \{y\}) \cup \{x\}.$$

Proof. Let π_x be a longest path in G that starts with x, and π_y the longest path that starts with y. The path π_x is not shorter than π_y, because $h(x) \geqq h(y)$. The order of the tasks on these paths is the same order in which they appear in the schedule S. Let k be the first slot in S at which the total number of tasks along π_x in slot k and in all previous slots, is equal to the corresponding number of tasks along π_y. There

: be such a slot, because S starts with π_y and not with π_x and the latter is not ter than the former. Moreover, no task of π_y exists in the slot $(S)_k$, otherwise, k ld not be the first.

Since the component of x is an outtree, every task in π_x has at most one immediate decessor which is the previous task along the path π_x. Similarly, every task along has at most one immediate successor, which is the next task along π_y. Therefore, can exchange the tasks of π_x and π_y which appear in the first k slots, one by one, pectively. The new schedule obtained is of the same length as S, and none of the cedence constraints represented by G are violated, simply because every task along moves upward following its only immediate predecessor, and every one along π_y ves downward preceding its only immediate successor. Since in the kth slot there no task of π_y, we can exchange the member of π_x, which is in $(S)_k$, with the last mber of the path π_y that is in a slot of lower index than k. Thus, we have obtained e desired optimal schedule that starts with the set $\{(S)_1 - \{y\}\} \cup \{x\}$. □

In the following theorem we use the Elite theorem to show that the inforest tasks in be chosen according to height.

THEOREM 6.2. *Let O be an outtree and I be an inforest. Let x be the root of O and M be a zigzag profile of breadth three. Then* i) *or* ii) *holds.*

i) *For any set T_1 of $m_1 - 1$ highest initial tasks of I there exists an optimal M-schedule for $O \cup I$ starting with $T_1 \cup \{x\}$.*

ii) *For any set T_2 of m_1 highest initial tasks of I there exists an optimal M-schedule for $O \cup I$ starting with T_2.*

Proof. If $H(O \cup I)$ is an outtree (resp. inforest), then by Theorem 5.1 (resp. Theorem 5.2) HLF is optimal for $H(O \cup I)$ and M. Furthermore, Lemma 4.2 implies that HLF is optimal for the whole graph $O \cup I$ and M, and the theorem holds.

Assume that $O \cup I$ is a counterexample with the fewest number of vertices. By the above remarks we know that $H(O \cup I)$ consists of the outtree O and an intree which we call N. Theorem 4.1 guarantees that if i) or ii) holds for $H(O \cup I)$ and M then it also holds for $O \cup I$ and M. Thus $O \cup I$ is not a minimal counterexample unless $L(O \cup I) = \varnothing$, i.e. $I = N$.

Let y be the task of N of height zero, and let S be any optimal schedule for $O \cup N$ and M. Assume y appears in slot k of S and let G' be the subgraph of $O \cup N$ induced by the tasks which are in the first $k-1$ slots of S. Observe that G' contains less tasks than $O \cup N$. Thus i) or ii) must hold for G' and M. Since $h_{G'}(x) = h_N(x) - 1$, for any task x of N in G', a set of highest initial tasks from N in G' is a set of highest initial tasks of N. We conclude that i) or ii) holds for $O \cup N$ and M, which contradicts the minimality of $O \cup N$. □

The following theorem, which is a consequence of the previous two theorems, leads to the Flip-Flop algorithm. Note that for any opposing forest G, either G of G^R contains an outtree task of height $h(x)$.

THEOREM 6.3. *Let G be an opposing forest that contains an outtree of height $h(G)$. Let M be a zigzag profile of breadth three; let x be the initial task of an outtree with height $h(G)$; and let A be any set of m_1 highest initial tasks of G that contains x. Then there exists an optimal schedule that starts with A.*

Proof. If $H(G)$ is an outforest, then the theorem holds by Theorem 5.1 and Lemma 4.2. Theorem 4.1 implies that it is sufficient to prove the theorem for the case when $L(G)$ is empty. Thus G consists of an intree and an outtree and we can apply the previous theorem. If case i) holds, then we are done. Assume ii) holds and let R be any set of m_1 highest initial tasks of the intree. This starts an optimal schedule for G and M. Let z be a minimum height task of R. Then $A = R - \{z\} \cup \{x\}$ is an arbitrary

set of m_1 highest initial tasks of G which contains the root of the outtree. By Theorem 6.1 this set starts an optimal schedule for G and M. □

7. The Flip-Flop algorithm. We first give a high-level description of the Flip-Flop algorithm that produces an optimal schedule for an opposing forest, G, and a zigzag profile, M, of breadth three. The algorithm has two phases, a *Flip-Flop phase* and an HLF *phase*. In the Flip-Flop phase we deal with the case when G has an intree and an outtree above the median. We iteratively remove sets of initial tasks from G or from the reverse graph G^R to fill up slots in the schedule we are constructing. To do this we apply Theorem 6.3, i.e. we remove from G if there is an outtree in G of height $h(G)$ and from G^R if there is an outtree in G^R of height $h(G)$. Notice that one of the two cases must hold and that we always choose a set of highest initial tasks that contains the root of the highest outtree. The sets of inital tasks removed from G are put in the first, second, $\cdots$, time slot, and those removed from G^R are put in the last, second to last, $\cdots$, time slot.

We stop the Flip-Flop phase and enter the HLF phase as soon as the HLF condition holds (given below). In the HLF phase we schedule the remaining graph according to highest-level-first. Our ability to stop the Flip-Flop phase assures the linearity of the algorithm, because we do not have to keep track of too many components.

The Flip-Flop algorithm produces two sequences of sets. One consists of the sets which were removed from the top (from G) and the other consists of the set removed from the bottom (from G^R). If the two sequences do not overlap, then we get a valid schedule for G. In the case of a straight profile we run the Flip-Flop algorithm on a very long straight profile and then just omit the empty slots between the two sequences to get an optimal schedule. In this case the Flip-Flop algorithm is linear. For zigzag profiles another factor of $\log n$ is required to find a minimum length valid schedule.

DEFINITION. We say that the HLF-condition holds for the opposing forest G if there are either only outtree components or only intree components above the median.

LEMMA 7.1. *If the* HLF*-condition holds for an opposing forest G, then* HLF *produces an optimal schedule for G and any zigzag profile of breadth three.*

Proof. Follows from Lemma 4.2, Theorem 5.1 and Theorem 5.2. □

The only remaining case in which the HLF-condition does not hold is when the opposing forest has exactly two components above the median: an outtree, and an intree with more than one initial task. Note that an intree that contains only one initial task is a chain of tasks and a chain is also an outtree.

We are now ready to present the Flip-Flop algorithm. The variables i_{top} and i_{bot} will point to the next empty slot in the corresponding end of the profile M. Initially they point to the first and last slots of the profile M, respectively. The variable $\hat{G}$ represents the opposing forest remaining at the current step of the algorithm. Initially, $\hat{G}$ is equal to G, and after each removal of a set of tasks from the graph the variable $\hat{G}$ becomes the resulting subgraph.

THE FLIP-FLOP ALGORITHM
(*Flip-Flop phase*)
  REPEAT
    WHILE **not** HLF? ($\hat{G}$) **and** an outtree component is the highest component of $\hat{G}$ DO:
    BEGIN
      Remove the initial task of the highest outtree and any other $m_{i_{\text{top}}} - 1$ highest initial tasks from $\hat{G}$. Add these tasks as slot i_{top} to the schedule S. Increase i_{top} by one.

END
IF **not** HLF? (G) THEN
WHILE **not** HLF? ($\hat{G}^R$) **and** an outtree component is the highest component of $\hat{G}^R$ DO:
BEGIN
Remove the initial task of the highest outtree and any other $m_{i_{bot}} - 1$ highest initial task from $\hat{G}^R$. Add these tasks as slot i_{bot} to the schedule S. Decrease i_{bot} by one.
END
UNTIL HLF̂? (G) **or** HLF̂? (G^R)
(*HLF phase*)
IF HLF? ($\hat{G}$) THEN continue to fill the schedule S, updating i_{top} by applying the HLF strategy to $\hat{G}$ FI.
If HLF ($\hat{G}^R$) THEN continue to fill the schedule S, updating i_{bot} by applying the HLF strategy to $\hat{G}^R$ FI.
(*check for overlapping of the two sequences*)
IF $i_{top} < i_{bot}$ THEN the constructed schedule fits the profile ELSE return failure.

THEOREM 7.1. *The Flip-Flop algorithm produces a schedule for an opposing forest and a zigzag profile of breadth three. If there is no feasible schedule it returns failure.*

Proof. To prove the correctness of the Flip-Flop algorithm it is enough to prove at if there exists a feasible schedule, then the algorithm will find one. It suffices to ow that all sets of initial tasks that are removed from $\hat{G}$ (resp. $\hat{G}^R$) start some timal schedule for $\hat{G}$ (resp. $\hat{G}^R$) and the corresponding profiles. Theorem 6.3 assures s for the Flip-Flop phase and Lemma 7.1 for the HLF phase. □

THEOREM 7.2. *The Flip-Flop algorithm can be implemented in time $O(n)$.*

Proof. We just sketch how to do this. A more detailed description is given in W2b]. First we outline that with some simple data structures an inforest or an tforest (and therefore an opposing forest) can be scheduled in time $O(n)$. We keep e initial tasks of the forest in an array of lists. The ith list contains all initial tasks level i. To facilitate the removal of up to three highest initial tasks we remember e highest three levels that contain initial tasks. After each removal of an initial set e update the array of lists and redetermine the highest three levels. We conclude that LF is linear for any kind of forest. In our algorithm we sometimes remove initial sks from the reversed graph according to HLF. This can be handled by keeping dual ata structures for G^R. So far we reasoned that the HLF phase is linear.

In the Flip-Flop phase we remove from $\hat{G}$ as well as from $\hat{G}^R$. Thus the heights the tasks in $\hat{G}$ and $\hat{G}^R$ might change and we need some more insights in the gorithm to assure linearity. Observe that if the HLF condition does not hold then ere are at least three tasks in the Elite. Thus during the entire Flip-Flop phase we nly remove tasks from the highest two components of G which must be an intree nd an outtree. As soon as the outtree splits into several outtrees then at most one of em will remain above the median and then this outtree will be a highest one. omponents that drop below the median do not need to be considered any more uring the Flip-Flop phase.

It is easy to keep track of the highest outtree vertex in $H(\hat{G})$ and $H(\hat{G}^R)$. But e also need to remove sets of highest initial tasks from the intree of $H(\hat{G})$ and $H(\hat{G}^R)$, even though the heights of the tasks of the intree in $H(\hat{G})$ and $H(\hat{G}^R)$ are hanging during the Flip-Flop algorithm. If we remove the highest outtree vertex of $H(\hat{G})$ (resp. $H(\hat{G}^R)$), then the heights of all intree tasks in $H(\hat{G}^R)$ (resp. $H(\hat{G})$) drop

by one. Thus their relative heights in $H(\hat{G})$ and $H(\hat{G}^R)$ remain the same and we do not need to update the heights. This completes the sketch of the linear implimentation. □

In the following theorems we show how we can use the Flip-Flop algorithm for finding optimal schedules.

THEOREM 7.3. *The Flip-Flop algorithm implies a linear algorithm to find an optimal schedule for an opposing forest and a straight profile of breadth three.*

Proof. We choose the initial profile to be of length n and run the Flip-Flop algorithm. The resulting schedule might contain some empty slots in the middle, that is, after the algorithm terminates $i_{\text{top}} < i_{\text{bot}}$. By removing the empty slots we get an optimal schedule. □

The proof of Theorem 7.2 does not hold for general zigzag profile, because the difference in the slots size prevents us from just squeezing the schedule to obtain an optimal one.

THEOREM 7.4. *The Flip-Flop algorithm implies on $O(n \log n)$ algorithm to find an optimal schedule for an opposing forest and a zigzag profile of breadth three.*

Proof. We run the Flip-Flop algorithm for different schedule lengths. The legitimate lengths are between $n/3$ and n; therefore by binary search we can find the shortest schedule that fits the profile in time $O(n \log n)$. □

Observe that in Flip-Flop phase only tasks of $E(\hat{G})$ and $E(\hat{G}^R)$, respectively, are removed. Therefore, the algorithm can be easily extended to work for every graph G for which $H(G)$ is an opposing forest. The basic reason is that tasks that are under the median will not "pop up" to be above the median, if we remove tasks according to height from $L(G)$ and $L(G^R)$, respectively(see properties M7 and M8 of the median). The correctness proof for this extended algorithm remains the same, using Lemma 4.2. For the sake of simplicity we restrict ourselves to the case where the whole graph is an opposing forest. The time complexity for running the Flip-Flop algorithm on a graph G for which $H(G)$ and $H(G^R)$ are opposing forests will be $O(n+e)$, where e is the number of edges in G. Note that in opposing forests e is $O(n)$.

Acknowledgments. We would like to thank Barbara Simons for many useful suggestions and thorough proofreading of this paper. We are also very grateful to the referee for simplifying some of our proofs.

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.

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

[CD73] E. G. COFFMAN, JR AND P. J. DENNING, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

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

[Do80] D. DOLEV, *Scheduling wide graphs*, Technical Report STAN-CS-80-832, Dept. Computer Science, Stanford Univ., Stanford, CA, December 1980.

[DW82a] D. DOLEV AND M. K. WARMUTH, *Scheduling precedence graphs of bounded height*, J. Algorithms, 5 (1984), pp. 48–59.

[DW82b] ———, *Scheduling flat graphs*, IBM Research Report RJ3398, May 1982.

[DW82c] ———, *Profile scheduling of opposing forests and level orders*, SIAM J. Alg. Discr. Meth., 6 (1985), to appear.

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

[Ga82] ———, *An almost linear algorithm for two processor scheduling*, J. Assoc. Comput. Mach., 29 (1982), pp. 766–780.

[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.

[GJ81] M. R. GAREY, D. S. JOHNSON, R. E. TARJAN AND M. YANNAKAKIS, *Scheduling opposing forests*, Technical Report, Bell Laboratories, Murray Hill, NJ, 1981; SIAM J. Alg. Discr. Meth., 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. Discrete 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. Computer Science, Washington State Univ., Pullman, September 1976.

[Hu61] T. C. HU, *Parallel sequencing and assembly line problems*, Operations Res., 9 (1961), pp. 841–848.

[LR78] J. K. LENSTRA AND A. H. G. RINNOOY KAN, *Complexity of scheduling under precedence constraints*, Oper. Res., 26 (1978), pp. 22–35.

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

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

[PY79] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Scheduling interval-ordered tasks*, this Journal, 8 (1979), pp. 405–409.

[Ul75] J. D. ULLMAN, *NP-complete scheduling problems*, J. Comput. System Sci., 10 (1975), pp. 384–393.

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