December 1980

Report. No. STAN-CS-80-832

Scheduling Wide Graphs

by

Danny Dolev

Research sponsored by
National Science Foundation

Department of Computer Science

Stanford University
Stanford, CA 94305

SCHEDULING WIDE GRAPES

*Danny Dolev**

Computer Science Department
Stanford University

Abstract.

The problem of scheduling a partially ordered set of unit length tasks on m identical processors is known to be UP-complete. There are efficient algorithms for only few special cases of this problem. In this paper we explore the relation8 between the structure of the precedence graph (the partial order) and optimal schedules. We prove that in finding an optimal schedule for certain systems it suffices to consider at each step high roots which belong to at most 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 may lead to the development of linear scheduling algorithms for many practical cases and to better bounds for complex algorithms. In particular, in the case the precedence graph contains only inforest and outforest components, this result leads to efficient algorithms for obtaining an optimal schedule on two or three processors.

Key words: optimal schedules, identical processors, intrees and outtrees.

*This research was supported in part by Chaim Weizmann Postdoctoral Fellowship and in part by National Science Foundation grant MCS-79-09495.

1. Introduction.

The **goal of** deterministic scheduling is to obtain **efficient** algorithms for scheduling multiprocessor systems under the assumption that the tasks to be scheduled are known in advance and are all simultaneously available for **scheduling**. One **of** the fundamental problems of the deterministic scheduling theory is the scheduling, on a collection of identical **processors,** of partially ordered **unit-length** tasks.

A schedule for a set of tasks ordered according to a given precedence graph is a description of the time at which every task is to be started, and to which processor it is assigned. The schedule must not violate the given precedence relation. The schedule should not assign any task more than once. To specify such a schedule, we use a Ganttchart, which is a tableau consisting of a row for each processor, and **a** column for each time quantum, with the name **of** each task appearing exactly once.

Various **aspects** of scheduling **theory** have been extensively studied **in recent** years [GL79] and many of the scheduling problems are known to **be** UP-complete ([G J78],[Ul75]). H*ere* we study the scheduling of n unit tasks on m identical processors, with a partial order among the tasks specified by *a* precedence graph. Only recently **Garey** et. al. [GJ80] proved the UP-completeness of scheduling in the particular case of a precedence graph composed only of inforests and outforests. We say that a graph is an ***inforest*** if the out-degree of **each** vertex **is** at most one. An ***outforest*** is an upside down inforest, in other words, a graph **in** which the in-degree of each vertex is at most one.

Polynomial algorithms have been developed only for a *few* special cases of this scheduling problem. The first polynomial algorithm known is due **to Hu** [Hu61]. It is *a* very simple linear algorithm (O(n)) which **produces** an optimal schedule for any number of processors if the precedence graph is either an **inforest** or an outforest. The algorithm of Hu, to which we will **refer** later as the *Highest* ***Level*** First algorithm (HLF), is a ***level*** algorithm. This means that we choose higher tasks over lower ones, and among tasks *of* the **same** height we choose arbitrarily.

Another **efficient** level algorithm is due to Coffman and Graham [CG72]. Their algorithm is polynomial ($O(n^2)$) and produces an optimal schedule **for** *an* arbitrary precedence graph, but only for two processors. It is a **level** algorithm because higher tasks *are* chosen first. However the choice among vortices of **the**

same height is predetermined by the algorithm which therefore is not an HLF one.

Another special case for which there exists an efficient algorithm is the case where the partial order among the tasks is given by an interval graph; that is, to each task corresponds an interval of the real line and the precedence between tasks is determined according to the precedence of disjoint intervals. Papadimitriou and Yannakakis [PY79] have recently shown an $O(n^2)$ algorithm for obtaining an optimal schedule in this case.

Garey et. al. [GJ80] also proved that for a precedence graph which contains only inforest and outforest, there exists an $O(n^{m^2+m-5} \log n)$ algorithm for optimal scheduling, if m is flxed. In the special case of three processors, $(m = 3)$, they have shown an $O(n^2 \log n)$ algorithm.

The complexity of the scheduling problem is the result of the need to look at many possibilities when creating each step of an optimal schedule. We surmount this obstacle by using a different approach. By studying deeply the relations between the structure of the graph and the optimal schedules, we prove some general theorems which enable us to decrease the number of possibilities to be checked in many precedence graphs. The general problem remains, of course, UP-complete but in some special cases the solution becomes much simpler. Our method may lead to the development of efficient scheduling algorithms for many practical cases and to better bounds for complex algorithms.

The statement of our main result is the following: Every precedence graph has some basic height, called the *median*, such that the choice of a set of tasks for an optimal schedule can be done exclusively among the free tasks (roots of the precedence graph) which are above this height. Choosing tasks lower than that height can be done by HLF.

As an example for the use of the main result we give O(n) algorithms for finding an optimal schedule for 2 or 3 processors in the case of a precedence graph containing only inforest and outforest components. In the case of 2 processors Goyal [Go76] proves a similar result. The basic idea behind the algorithm for three processors is to sometimes remove vertices from the top of the precedence graph and sometimes from the bottom. The removed vertices are inserted as columns into two tableaux which are combined at the end into an optimal schedule for the graph.

In section 2 we describe three basic principles of optimal schedules, which enable us in later sections to manipulate with optimal schedules to prove desired properties. In section 2 we define the notion of median, which is the special height characterizing the results we prove later. We also prove some basic properties of the median. In section 3 we define the ***Elite*** of a graph and prove our main result, called the ***Elite Theorem.*** In se&ion 4 we use the Eiite Theorem to prove some properties of graphs with **inforest** and outforcst components. Using these properties we develop, in section 5, linear algorithms for scheduling on two and three processors.

2. **Principles** of **optimal scheduling.**

Let G be a precedence graph for a set of n tasks. The variable n will always refer to the number of tasks and the variable m to the **number** of processors. By a precedence graph we mean that G = (V, E) is a directed acyclic graph (a *DAG),* where V is the **set** of n tasks and E is a set of ordered pairs of tasks. We also assume that if a task x has to be executed before a task y then there exists a ***path*** (directed path) from x to y in G. Note that we assume neither that G is closed under transitivity nor that it does not contain redundant edges.

We use the following notions to express an ordering between tasks; if (z, y) is an edge in E then y is a child *of* x and x is a ***parent*** of y; if there exists **a** path from x to y, then y is a *descendant* of x and x is an *ancestor* of y. By $H(G)$ we mean the *height* of G, which is the **length** of the longest path in C. For a vertex $x \in$ G (i.e. $x \in V$) we denote by $H(x)$ **the** length of the longest path which starts at x. Wc define the ***length*** of a path which contains only one vertex to be *zero.* We use the notion of ***level*** to denote vertices of the same height in G. Let V' be a subset of V; *we* say that G' is the **subgraph** of G *induced* by V' if it is the maximal **subgraph** of G which contains exactly the **set** of vertices V'. If G^1 and G^2 are the subgraphs of G induced by V^1 and V^2 respectively we denote by $G^1 | G^2$ the graph induced by $V^1 \bigcup V^2$. Whenever we use the term ***subgraph*** in this paper, we mean the maximal **subgraph** induced by its set of vertices. We use only this meaning because **when** WC look at a **subset** of the tasks we are interested in the complete **ordering** among them (which is **determined** by the whole graph) and not at a subordcring; that is, we want to avoid the deletion of constraints (edges) between tasks.

The graph G = $(V,$ E) is composed of $\{G_1, \ldots, G_n\}$ if these subgraphs (called *components* of G) are a decomposition of G into its connected components; that

is, each subgraph is a connected graph and there are no edges in G which connect them, therefore, $V = \bigcup_i V_i$ and $E = \bigcup_i E_i$. $Root(G)$ is the set of all the roots of the directed graph G that is, the set of vertices with zero in-degree. Note that the $root(G)$ is not necessarily the *Top-level* of G, which contains the vertices of maximal height in G.

We concentrate only on the case of tasks of equal length. S(G) denotes a schedule for G which is an ordering of the tasks (vertices) into a tableau of columns, each of size at most m, in a way that does not contradict the partial order induced by G; that is, if x appears in column i and y appears in column $j \geq i$ then there is no path from y to x in G. The G is omitted when it is clear to which graph the schedule corresponds. $(S)_i$ denotes the i-th column of S, and $(S)_1$ is the set of tasks with which S starts. The *length* $t(S)$ of a schedule is the number of its occupied columns (obviously $l(S) \geq H(G)$). The schedule S is an ***optimal schedule*** for G if $t(S) \leq t(S')$ for every schedule S' for G.

For example assume we have a set of fifteen tasks subjected to the precedence graph G presented in Figure 1. We name the tasks by integer numbers. Throughout the paper we always assume that the graphs are directed downwards.

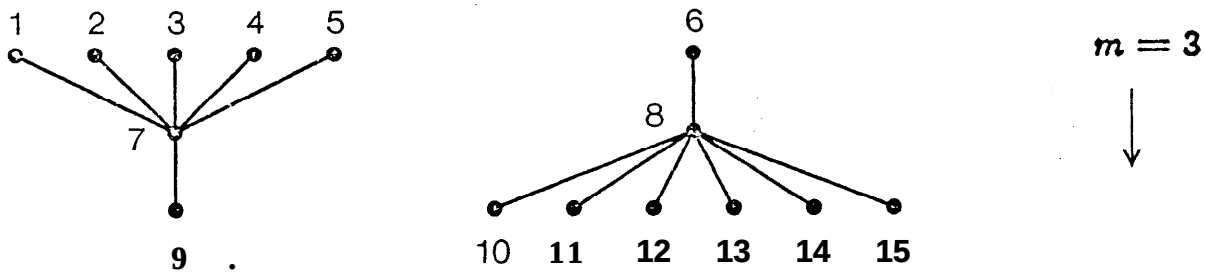


Figure 1.

The following tableau is a schedule for G on three processors:

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

This schedule is a level schedule but it is not an optimal schedule because there exists a shorter one which takes only five time units.

A schedule is a *level* schedule if higher level tasks are chosen before lower level ones. Level schedule induces the following property on its tableau: for every task x, if x is in column i and there exists, in a previous column j ($i > j$), a task y which is lower than x in the precedence graph ($H(z) > H(y)$), then there is an ancestor of x in the j-th column. A level algorithm is a *Highest Level First* algorithm (HLF) if the choice among vertices of the same height can be arbitrary. This is a level algorithm with the additional property that if y is of the same height as x then there exists another schedule which has the same first j columns except for the fact that x is exchanged with y in the j-th column. Thus, if by saying that HLF produces an optimal schedule we mean that any level schedule is optimal. Sometimes we emphasize this fact by saying 'any HLF algorithm produces an optimal schedule."

Definition 1: Let G be a *DAG* and S(G) be an optimal schedule for G.

(1) For $x \in G$, let k be the column in which x appears in S. The schedule S is **minimal** with respect to {w.r.t.} x if k is the minimal column in which x appears in all the optimal schedules for G.

(2) Let $\pi = x_1 \cdot \ldots x_r$ be a path in G and $(n_1, \ldots, n_r)$ be the columns of S in which $x_1, \ldots, x_r$ appear, respectively. The schedule **S is minimal w.r.t. π if**

   (i) S is minimal w.r.t. x_1.

   (ii) for every j, $1 < j \leq r$, n_j is the minimal column in which x_j appears in all the optimal schedules for G which are minimal w.r.t. the path $x_1 \cdot \ldots x_{j-1}$.

Some further notations we use: The set $\{S^i\}_{i=1}^{l}$ is a decomposition of S '(denoted by $S = S^1 | S^2 | \cdots | S^l$) if $S^1, \ldots, S^l$ are consecutive subtableaux of S where the concatenation of these consecutive tableaux is the tableau S. To each such subschedule S^i of S, there corresponds a subgraph G^i of G which is the subgraph of G induced by the set of tasks appearing in S^i. We use the same notation to express the attachment of subschedules. Let G^1 and G^2 be subgraphs of G, and S^1 and S^2 their corresponding schedules. If there is no path from a vertex in G^2 to a vertex in G^1 then $S^1 | S^2$ denotes a schedule for $G^1 | G^2$ composed of the columns of S^1 followed by the columns of S^2. Note that the

notation $x|y$ has a meaning only if its two operands (x and y) are parts of some compound structure.

Theorem 1 presents the basic conservation property of optimal schedules, which we call the *principle of optimality*. This property is one of the three fundamental properties of optimal scheduling. We will make frequent use of these properties, sometimes without referring explicitly to them.

Theorem *1. (the principle of optimality)*
Let S be an optimal schedule for G and $S = S^1|S^2|\cdots|S^l$. Let $\{\hat{S}^i\}$ be any set of optimal scheduler for the **corresponding subgraphs induced by $\{S^i\}$. Then $\hat{S} = \hat{S}^1|\hat{S}^2|\ldots|\hat{S}^l$ is an optimal schedule for G.**

Proof. The case $l = 1$ is obvious. Assume $l = 2$, $S = S^1|S^2$ and let G^1, G^2 be the corresponding subgraphs. From the definition of a subschedule and that of the induced subgraph it is clear that for any pair of vertices, y in G^2 and x in G^1 the edge (y, x) is not in G. Let $\hat{S}^1$ and $\hat{S}^2$ be optimal schedules for G^1 and G^2, respectively. From the optimality we conclude $t(\hat{S}^1) \leq t(S^1)$ and $t(\hat{S}^2) \leq t(S^2)$; therefore $t(\hat{S}^1) + t(\hat{S}^2) \leq t(S^1) + t(S^2)$ = t(S). The new tableau $\hat{S} = \hat{S}^1|\hat{S}^2$ does not contradict the partial order represented by G simply because there exists no path from G^2 to G^1. Therefore the optimality of S implies the optimality of $\hat{S}$.

The inductive case follows immediately from the case $l = 2$. ∎

Theorem 1 enables us to make local changes in a given initial optimal schedule without affecting the optimality. It also enables us to construct from a given optimal schedule another one which satisfies specified conditions. We also USC the principle of optimality to produce an optimal schedule by the following means: Assume that we have some magic algorithm which, though it cannot specify an optimal scheduling for an input graph, can point out a subset of the roots of the graph with which some optimal schedule starts. The principle of optimality enables us to produce an optimal schedule using the magic algorithm. We use the output of the magic algorithm as a column in a tableau and remove it from the graph, and then reapply the magic algorithm. The principle of optimality implies that this recursive application of the magic algorithm produces a tableau which represents an optimal algorithm for the graph.

Optimal schedules have two more basic properties. Denote by G^R the *reversed graph* of G; that is, the graph we get by reversing all the directed edges in G. By S^R we denote the *reverse tableau* of S; that is the tableau we get by reading S from the end to the beginning. We call the following property the *reversing* principle.

> If S is optimal for G then S^R (the reversed tableau) is optimal for G^R.

The last basic property which we point out is the *nonvacancy* principle which holds only in the case where all the tasks are of equal length,

> One gains nothing by leaving empty places in the tableau which describes the optimal schedule.

The nonvacancy property together with the principle of optimality imply that whenever a graph has less than $m+1$ roots, then the set of all its roots starts an optimal schedule. Therefore we can put them as a column in the scheduling tableau and look for an optimal schedule for the rest of the graph.

In this paper we study graphs which are 'wide," in the sense of having sometimes more than m components. We use the notion of *median to* define the minimal level which does not have as many components; that is, the first level which contains less than m components of G.

Definition 2: Let G be a directed acyclic graph *(DAG)* and $\{G_i\}$ be its components. The *median* of G with respect to a given m is the minimal non-negative integer μ for which the set $\{G_i \mid H(G_i) > \mu\}$ contains strictly less than m components.

For example if the precedence graph is the one in Figure 2 and the number of processors is three, then the median is 2 because 2 is the minimal level which contains less than 3 components of the graph. If the precedence graph is the one in Figure 1, then the median is 0.

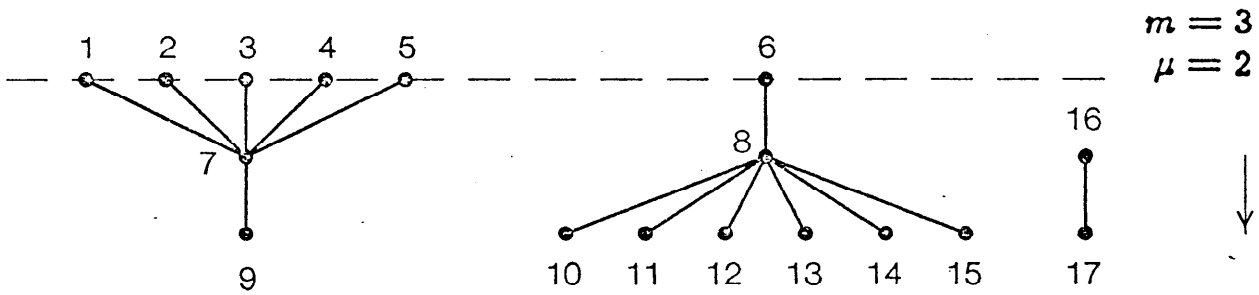


Figure 2.

We denote the median of G with respect to m by $\mu(G)$. In this paper we **only** use the notion **of the** median for a fixed number of processors ***m;*** therefore we will omit the parameter ***m*** from our notation for the median. We say that a component G_i of G is under *the median* if $H(G_i) \leq \mu(G)$, and ***above the median*** if the component **is** strictly higher than the median. We also use the same notions for vertices. If $\mu \geq$ W(G), we say that the graph is *flat,* and **if** $\mu = 0$ we say that the graph is *narrow.* Note that if $\mu \neq 0$ then there exists a root **of** height $\mu - 1$.

It is convenient to think of the median as defined in terms of a complete decomposition of the precedence graph into its components. Because if we would not define the decomposition to be specific for every graph then we would have to specify the decomposition at each time we use the median. Note that although we omitted that possibility, all the results in the paper still hold for any decomposition of the graph into edge-disjoint components. The delicate point is to define what inequalities between medians mean, but this can be done in a consistent way.

During the construction of a schedule, the median is a dynamic line; **whenever** we choose a task above it we may increase it and each time we choose **a** task under it we might decrease it. Some properties of the median are presented **in** Theorem 2 .

Theorem **2.** (The *median theorem)*

Let G $= (V, E)$ be **a** DAG and $G' = (V', E')$ be **a subgraph** of G.

(1.) **If** V' contains **at least all the vertices of** V **which are** not **roots** of G, then

$$\mu(G') \geq \mu(G) - 1.$$

(2.) **If** V' contains **at least all the vertices of** V **which are of height** h, **and all .**

their descendants, then

$$\mu(G') \geq \min(\mu(G), h + 1).$$

Proof. The median has the property that there exist at most m — 1 components of the graph with height not less than the median and at least *m* components of height not less than the median minus one.

Case 1: We have to show that G' has at least m components of height not less than $\mu(G) - 2$. But this fact is clear because G has at least m components of height not less than $m - 1$ and the deletion of roots and only roots can decrease the height of any component at most by one. On the other hand the number of components can be increased (unless G is of height zero, in which case the property trivially holds). Therefore G' has at least *m* components of height $\mu(G) - 2$ and higher which implies the desired relation.

Case 2: If $\mu(G) = 0$ the case trivially holds. So, assume $\mu(G) > 0$ and let $h' = \min(\mu(G), h+1)$. Every component of height not less than $h' - 1$ has at least one vertex of this height. We know that all these vertices and all their descendants appear also in the subgraph G'. Therefore, each such vertex remains at the same height as in G. If two vertices of height h' belong to different components in G then they obviously belong to different components in G', but it can be that two vertices of the same component of G belong to different components in G'. Thus, G' has at least the same number of components of height not less than $h' - 1$, as G does. G has at least *m* components of height $h' - 1$, and therefore we conclude the desired inequality. ∎

Figure 3 describes case *(2)* of the theorem. We assume that the graph G is the whole "cloud" and the subgraph G' is the part under the dashed line. $\mu(G')$ can have a value between $\min(h + 1, \mu(G))$ and H(G) + 1.

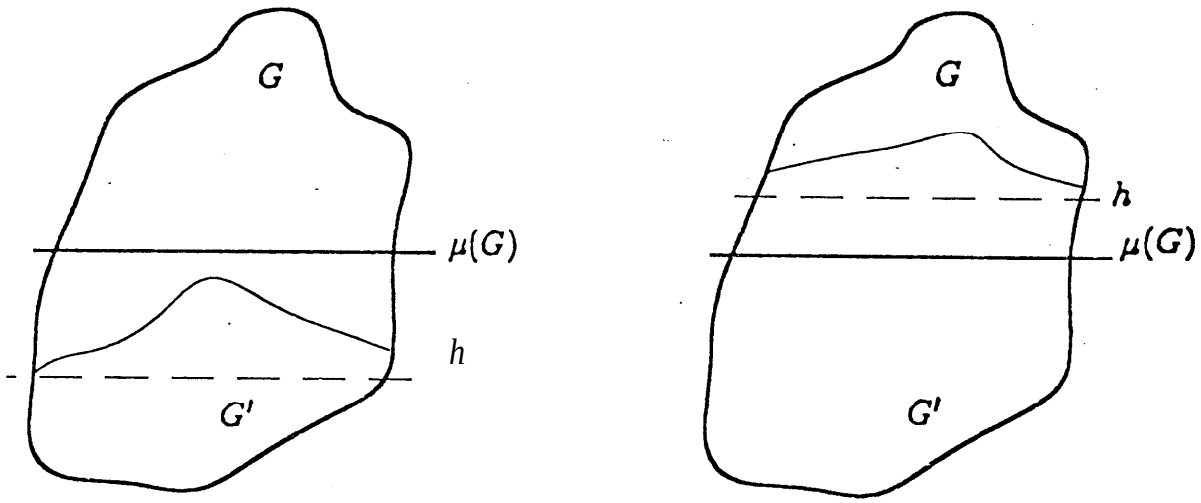


Figure 3.

The following corollaries arc useful in many cases. They present some of the situations we will **face** in the construction of optimal schcdulcs.

Corollary *3.* Given a precedence graph G, remove **Prom it** some **set** of **vertices, all** of **them** not **lower than** $\mu(G)$. **Let** G' bc the **resulting subgraph. Then**

$$\mu(G') \geq \mu(G).$$

Proof. The proof follows easily from Thcorcm 2 by taking $h = \mu(G) - 1$. ∎

Our next corollary claims that if a component of the graph is once under the median than after any choice of vertices according to height it will remain under the median of the resulting graph. Let G = $G_H | G_L$ where G_L is the subgraph which is composed of a11 the components which arc under the median; that is, $H(G_L) \leq \mu(G)$. G_H is the rest of the graph.

Corollary 4. Remove from G **any set** of highest roots. **Let** G', G'_L and G'_H **bc the resulting subgraphs. Then**

$$H(G'_L) \leq \mu(G').$$

Proof. By definition $H(G_L) \leq \mu(G)$. On the other hand, from Theorem 2 we know that

$$\mu(G') \geq \mu(G) - 1.$$

It is enough to prove that if there exists a root of G'_L of height $\mu(G)$ then $\mu(G') \geq \mu(G)$. Assume that this is the case and let $x \in G'_L$ be that vertex. $H(x) = \mu(G)$ and $H(G_L) \leq \mu(G)$ thus x is also a root of G_L. We removed only highest roots from G and if x was not removed then no root of G which is lower than x has been removed. Therefore G' was constructed only by removing vertices, all not lower than $\mu(G)-1$. This implies, by Theorem 2, that $\mu(G') \geq$ p(G). ∎

3. The Elite Theorem.

Our main result is the "Elite Theorem" which assures us that to find an optimal schedule it is sufficient to check at each time only roots above the median. If there are less than m roots above the median then we take the rest of the roots according to their height, choosing arbitrarily among vertices of the same height. Therefore, we can decrease the 'width" of the graph that has to be considered at each step of the construction of an optima1 schedule. However, the resulting problem is still UP-complete , as can easily be concluded from the construction used by Ullman ([Ul75]) in the proof of UP-completeness of the general problem.

Definition 3: A root x of a graph G belongs to the *Elite,* $\mathcal{E}(G)$, **of** G **if** it is above the median; that is, if $H(x) > \mu$, where μ is the median of G.

In order to prove our main result, we must first prove another theorem relating the notion of median to optimal schedules, The theorem states that given an optimal schedule starting with a root lower than the median, if there is another root of the graph not in the initial set of that optimal schedule, then there is another optimal schedule which starts with the same vertices but with the lower root replaced by the higher one.

Theorem 5. ***The Basic Theorem***

Let *G* be a *DAG* and S(G) be an optimal schedule for *G* where $(S)_1 = (g, x_2, \ldots, x_m)$. Let w $\notin (S)_1$ be any **root such that H(w) $\geq$ H(g). If either**

(i) $H(g) \leq \mu(G)$, **or**

(ii) $H(g) = \mu(G)$ **and** $H(g) < H(x_i)$ **for every** $2 \leq i \leq m$,

then there exists an optimal schedule $S'(G)$ such that

$$(S')_1 = (w, x_2, \ldots, x_m).$$

The utility of this theorem comes from the fact that the tasks $\{x_2, \ldots, x_m\}$ appear also in the first column of the new optimal schedule; the only difference **in the** initial set is that g is replaced by w. Thcreforc, we can USC the **theorem** to go step by step from one initial set to another.

First we sketch the proof of the Basic Theorem. We prove **that** among **all** the optimal schedules for the graph ***G***, there is **one** which starts with the same set as the schedule S having the following structure:

g^0		g^1			$g^{\nu-1}$		y	
x_2		w^1			$w^{\nu-1}$		w^ν	
x_3	S_1^1	z_3^1	S_2^1	$\ldots$	$z_3^{\nu-1}$	S_ν^1	z_3^ν	S_ν^3
$\vdots$		$\vdots$			$\vdots$		$\vdots$	
x_m		z_m^1			$z_m^{\nu-1}$		z_m^ν	

Figure 4.

For every ι, $1 < \iota \leq \nu$, **the** above structure has the following **four properties:**

(A) S_ι^1 **does** not contain any child of $g^{\iota-1}$.

(*B*) $\{z_3^\iota, \ldots, z_m^\iota\}$ does not contain any child of $g^{\iota-1}$ **and** y is not, also, **a child** of $g^{\nu-1}$.

(C) S_i^1 does not contain any parent of w^i.

(D) For $1 \leq i \leq \mu - 1$, $\{x_3^i, \ldots, x_m^i\}$ does not contain any parent of w^{i+1} and $\{x_2, \ldots, x_m\}$ does not contain *any* parent of w^1.

If we succeed in proving that there is an optimal schedule with the above four properties then we can easily complete the proof of the Basic Theorem. Indeed, the above properties ensures that we can exchange the series $< g^0, \ldots, x_m >$ with the series $< w^1, \ldots, w^\nu >$, one by one, and obtain an optimal schedule which satisfies the Basic Theorem.

The formal proof of the Basic Theorem appears in the Appendix. A graph G is a conflict-free graph if it satisfies the Basic Theorem and *if* all its subgraphs also satisfy the Basic Theorem. Note that the Basic Theorem states that every graph is a conflict-free graph, as we will prove later.

After such--a long preparation we are finally ready to present the Elite Theorem. The statement of the Elite Theorem is the following: to find an optimal schedule it is sufficient to look at the roots of the precedence graph that are above the median. It also states that if there is no root above the median then HLF produces an optimal schedule.

Before giving the precise statement of the Elite Theorem we will demonstrate it on a few examples. First assume that the precedence graph is the one described in Figure 5.

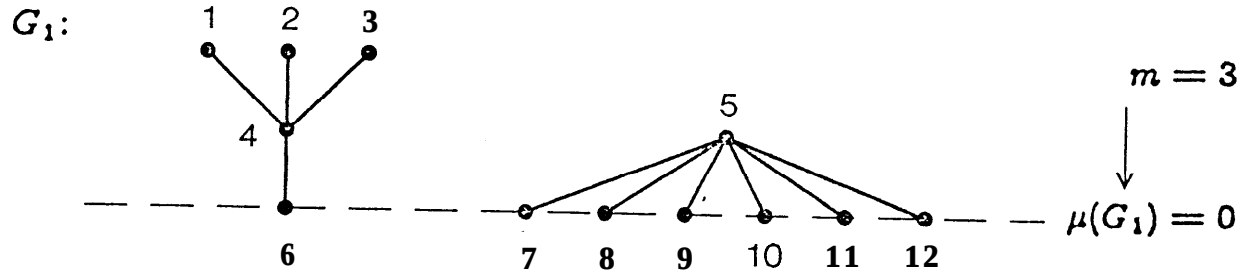


Figure 5.

In Figure 5, $\mu(G_1) = 0$ and therefore to find an optimal schedule we have to start by choosing three of the four tasks that are above the median. These four tasks are the Elite of the graph. If the precedence graph is the one given in

Figure 6 the situation is much simpler.

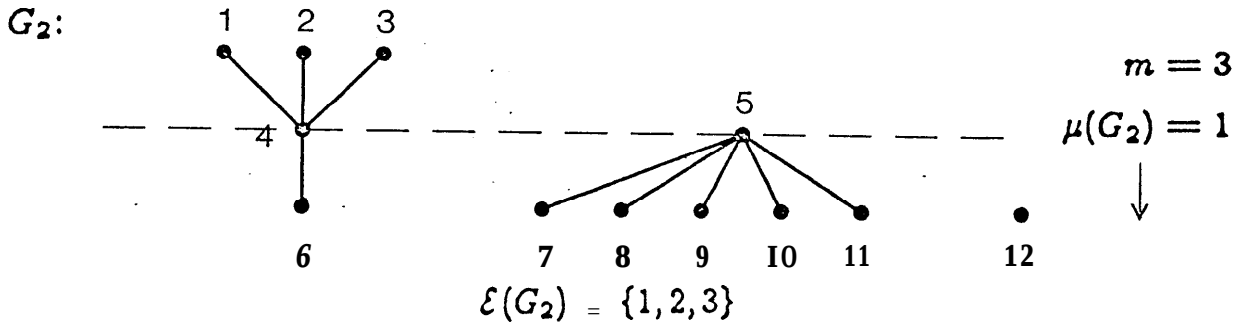


Figure 6.

In this case we **know** that there exists an optimal schcdulc starting with $\mathcal{E}(G_2)$, and if we remove this set from the graph we obtain the graph described in Figure 7. The Elite of this graph is empty ($\mathcal{E}(G_2') = \emptyset$) and thus (as we will prove in the Elite Thcorcm) any HLF algorithm produces an optimal schedule for this graph.

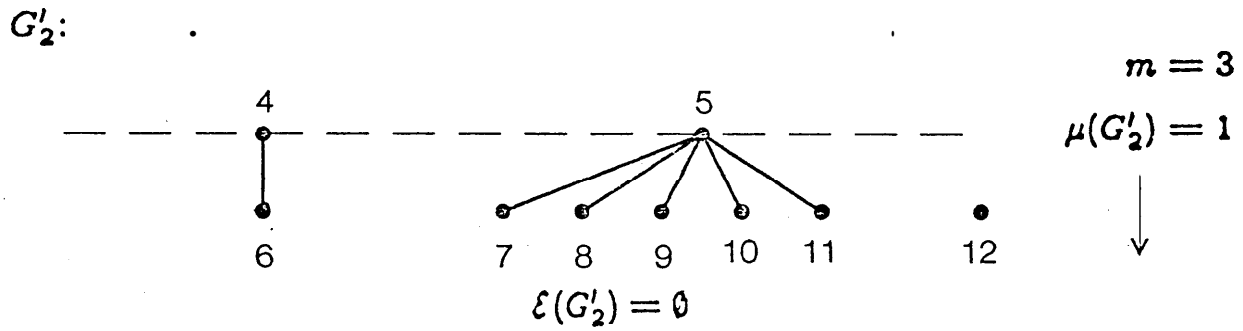


Figure 7.

Thcrcforc the tableau T_2' describes an optimal schcdulc for G_2' and T_2 describes an optimal schedule for the whole graph G_2.

T_2':

P_1	4	6	9
P_2	5	7	10
P_3	12	8	11

T_2:

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

Theorem 6. *The Elite Theorem*

Let G be a precedence graph for ordering a set of n unit tasks on m identical processors and $\mathcal{E}(G)$ its elite, then

(i) **If E(G) contains more than m roots, then there exists an optimal schedule for G which starts with a** subset of $\mathcal{E}(G)$.

(ii) **If $\mathcal{E}(G)$ contains m vertices or less, then** there **exists an optimal scheduling for G starting with a set of m highest roots of G. If such a set does not exist we then take all highest roots** of G.

(iii) **If E(G) = 0 then any HLF algorithm produces an optimal rehedule for G.**

Proof. We prove **(i) first.** Assume that $|\mathcal{E}(G)| \geq m$. The following claim is **a** straightforward **subcase** of the Basic Theorem.

Claim: Let S be an optimal schedule for G and **let** $x \in \mathcal{E}(G)$. If $y \in (S)_1 \setminus \mathcal{E}(G)$ is of minimal height in $(S)_1$ then there exists another optimal schedule, S', **for G** such that $(S')_1 = ((S)_1 \setminus \{y\}) \cup \{x\}$.

Using the above Claim it is clear that if the set $(S)_1 \setminus$ E(G) **is** not empty, we can decrease it until it is.

The proof of (ii) is similar. Assume that $\mathcal{E}(G) \neq 0$, then **as** in the previous case we can find an optimal **schedule** such that $\mathcal{E}(G) \subseteq (S)_1$. All the roots of G which are outside the Elite of G ***are*** under the median and therefore, again by the Basic Theorem, we can exchange them one by one until we get the **desired** set of highest roots **of** G. Note that the maximality of the **set** means that if there are less than m roots then the set contains all of them, otherwise it contains m vertices.

We prove **(iii)** by induction on n. First we point out that if the length of **a** schedule of n tasks on m processors is equal to $\lceil n/m \rceil$ then obviously it is an optimal schedule. We prove now by induction on n that **for** every precedence graph G if &(G) = 0 then any HLF **produces** a schedule of length $\lceil n/m \rceil$ **and** therefore optimal.

First consider the case where G contains not more than m vertices and $\mathcal{E}(G) = 0$. In this case, if N(G) > 0 then it has **less** than m components and therefore $\mu(G) = 0$ which implies that $\mathcal{E}(G) \neq 0$. Thus, it must be that N(G) = 0. But if this is the case then HLF obviously takes only one unit **of** time which in

our case is equal to $\lceil n/m \rceil$.

Suppose the assumption holds for every precedence graph with less than n vertices $(n > m)$ and with an empty Elite. Let G be a prcccdcnce graph of n vertices with an empty Elite. The facts that $n > m$ and that $\mathcal{E}(G) = 0$ imply that G has at least m components and therefore it has at least m roots. Let A be any set of highest roots of G; that is, A is an initial set of some level algorithm. Let G' be the aubgraph we get after removing the set A from the graph G. G' has $n - m$ vertices. We distinguish between two cases:

Case $H(G') < H(G)$: Then obviously $H(G') = H(G) - 1$ and on the other hand by the first part of Theorem 2 we also know that $\mu(G') \geq \mu(G) - 1$. The fact that &(G) = 0 implies that $\mu(G) \geq H(G)$ and therefore $\mu(G') \geq H(G')$ which implies that $\mathcal{E}(G') = 0$.

Case $H(G') = H(G)$: Then by the second part of Theorem 2 (choosing $h = H(G) - 1$) we conclude that

$$\mu(G') \geq \min(\mu(G), H(G))$$

The fact that the Elite of G' is empty implies that $\mu(G) \geq H(G) = H(G')$ and therefore $\mu(G') \geq H(G')$ which implies that $\mathcal{E}(G') = 0$.

Thus, we have shown that the Elite of G' is empty in all cases and therefore, we can apply the inductive hypothesis for G' to conclude that any HLF takes time $\lceil (n - m)/m \rceil$. Therefore any HLF for G takes time $\lceil (n - m)/m \rceil + 1 = \lceil n/m \rceil$ which completes the proof of the Elite Theorem. ∎

We end this section with a useful corollary of the Elite Theorem.

Corollary 7. Let G be a *DAG* having the following properties:

(1) $|\mathcal{E}(G)| \leq m$.

(2) removing any set of up to m highest toots leaves a rubgtaph G' with $|\mathcal{E}(G')| \leq m$.

If removing *any* set of up **to m highest roots from the rubgtaph obtained in (2) again leaves a rubgtaph obeying the above** properties **then *any* HLF producer an optimal schedule *for* G.**

Proof. The corollary can be easily proved, by complete induction, using the Elite Theorem and the principles of optimal scheduling. ∎

A simple case in which Corollary 7 can be used is when the components above the median are trees.

4. Fan graphs.

We say that a graph is a *fan graph* if all its components are intrees and outtrees, that is, it is composed of inforcst and outforest. Hu [Hu61] proved that any HLF algorithm produces an optimal schedule for trees. The same fact also holds if G is outforest. But it does not hold for fan graphs, the scheduling of which is UP-complete [GJ80]. An algorithm, of order $O(n \log n)$, for scheduling fan graphs on three processors was very recently developed by Garey et. al. [GJ80]. We use the Elite Theorem to prove the existence of a linear algorithm with a small constant for scheduling of fan graphs in such a case. We also obtain easily that HLF produces an optimal schedule for fan graphs in the case of two processors, this result was also obtained by Goyal [Go76].

First we prove a general result concerning graphs having the property that all the highest components are either inforcst or outforest. We call an outforest that is composed of only one component an *outtree*, similarly, an *intree* is a one-component inforest.

Theorem 8. **Let G be a** precedence graph. **If the elite of G contains either only outforest** roots **or only inforest roots then HLF producer an optimal schedule G.**

Proof. Consider flrst the case where the elite contains only outforest roots. In this case every outtree component above the median has exactly one root. Therefore $|\mathcal{E}(G)| < m$. If we remove from G any set of highest roots then by Corollary 4 all the lower components will remain under the median. Removing outforest roots leaves an outforest and therefore the new subgraph we get contains only roots of an outforest above its median. Moreover, it also has less than m vertices above its median. This fact remains true after removing each highest root from the resulting subgraph. Therefore by Corollary 7, HLF produces an optimal schedule for G.

Consider now the other case. Note that removing of roots from an inforcst does not increase the number of components. Moreover, the number of roots is never increased. We distinguish between two cases. First, if $|\mathcal{E}(G)| \leq m$ then

from the above note we conclude that HLF produces **an** optimal schedule. The second and last case we have to consider is **the** case where $|E(G)| > m$. We intend to prove that if E(G) contains only **inforest** roots and $|E(G)| > m$ then **HLF** produces an optimal schedule for G. This can be proved **by** complete induction. Let us assume that the inductive hypothesis holds for all the subgraphs of G **and** show that it holds for G. Remove from G any set A of m highest **roots.** Let G' be the **resulting** subgraph. By **the** reasoning of the first case, $\mathcal{E}(G')$ contains only **inforest** roots. If $|\mathcal{E}(G')| > m$ then by the inductive hypothesis HLF produces an optimal schedule. Attaching the **set** A to an optimal schedule produces **an** optimal schedule for G. If $|\mathcal{E}(G')| \leq m$ then by the first case any HLF produces an optimal schedule for G'. And again from it **we** obtain an optimal **schedule for** G. Therefore in any case HLF produces an optimal schedule for G. I

The **case** of scheduling a fan graph on two **processors is an immediate** corollary **of** Theorem 8.

Corollary 9. Let G be a fan graph. Then **HLF produces an optimal schedule for G on two processors.**

Proof. In case of two processors the Elite of a graph can contain roots of **at** most one **component** of the graph. In our case **the** graph **is a** fan graph, therefore $\mathcal{E}(G)$ contains either roots of an **intree** (inforest of one component) or a root **of an** outtrce. In any case by Theorem 8 we conclude that HLF produces an optimal schedule for G. ∎

Note that Corollary 9 holds also *for* non-fan graphs in which the highest component is either an **outtree** or an **intree.**

The **case** of 3 processors is **more complicated** and we need some more general results for achieving an algorithm for optimal scheduling in this case. Let us **however** first describe intuitively the optimal scheduling algorithm we will use. Consider a fan graph G in which both the top and the bottom contain only outforcst and **inforest** roots. Choose a side such that the highest level contains a root of an **outtree.** From that side remove the root **of** the outtrce and any other two highest roots. Keep the **removed** roots in a tableau of this side (we **keep** two tableaux). Now, look at the graph resulting from the removal of these roots, and repeat the same process on that graph. At the end we arc left with two tableaux: in one of them vertices **removed** from the top and **in other from the**

bottom, Construct from them one tableau by attaching the reverse of the bottom tableau to the top tableau. The tableau thus obtained is an optimal schedule **for** the given precedence graph.

Lemma 10. Let G be any graph **which has an intree and an outtree component. Let** S(G) **be an** optimal schedule for G. Assume **that** y **is** a **root of an intree component and that** x **is** a root of an **outtree** component, such **that** $H(x) \geq H(y)$. **If** $y \in (S)_1$ **and** $x \notin (S)_1$ **then there exists an** optimal **schedule,** $\hat{S}$, **for G which starts with** x **instead** of **with** y; that **is,** $(\hat{S})_1 = ((S)_1 \setminus \{y\}) \cup \{x\}$.

Proof. Let π_x be the longest path in G which starts with x **and** π_y **the one that** starts with y. The path π_x is at least as long as π_y. The vertices on these paths appear in the schedule S in the order in which they appear in the paths. Let k be the first column in S at which the number of vertices along π_x which appear **up to that** column is equal to the corresponding vertices along π_y. There must be such a column **because** S starts with π_y and not with π_x and the latter is not shortef **than** the former. Moreover, no vertex of π_y exists in the column $(S)_k$, otherwise k would not be minimal.

π_x is a path in **an outtree,** therefore every vertex in π_x **has** at most one parent, which is the previous vertex along the path π_x. Similarly every vertex along π_y **has** at most one child, which is the next **vertex** along π_y. Therefore we can exchange the vertices of π_x and π_y which appear in the first k columns one **by** one, respectively. The **new** tableau we obtain is of the same length as S, and none of the orderings induced by G is violated, simply because every vertex **along** π_x moves upward, following its only parent, and every one along π_y moves downward following its only child. Note that in the k-th column there is no **vertex** of π_y **and** therefore the first to move can do so without contradicting the order induced by G.

Thus, we have obtained the optimal schedule which we were looking for because it starts with the set $((S)_1 \setminus \{y\}) \cup \{x\}$. ∎

Using the Basic Theorem we can strengthen Lemma 10, **for** the **case in which** the graph G is a fan graph.

Theorem 11. Let G be a fan graph, Let G_k **be a maximal height component of G. If** G_k **is an outtree then there** exists an **optimal schedule for G which starts**

with G_k.

Proof. Let S(G) be an optimal schedule for G and let x be the root of G_k (an outtree has only one root). Assume that $(S)_1$ does not contain x, we construct another optimal schedule which starts with x. If $(S)_1$ contains only roots of outforest then G has more than m components. Therefore, one of those in the first column must be under the median. This implies, by the Basic Theorem, that there exists an optimal schedule in which that root is exchange with x. The only case left is the case in which there exists a root of an intree in $(S)_1$, in this case we can apply Lemma 10 to get the optimal schedule. ∎

Thus, among components of the same height we will always choose an outtree if possible. In some sense, we can say that outtrecs have priority over intrees. The following lemma extends this result.

Lemma 12. For $m = 3$ let G be a fan graph and x be the root of an outtree in C. Denote by $\pi_x = x_1 \cdots x_r$ a maximal path in G which starts at x. Let S(G) be an optimal schedule which is minimal w.r.t. π_x. Denote by $(a_1, \ldots, a_r)$ the columns of $x_1, \ldots, x_r$ in S, respectively ($a_0 = 0$). For every j, $1 \le j \le r$, if y is a vertex in $(S)_i$, where $a_{j-1} < i < a_j$ then $H(y) > H(x_j)$.

Proof. Assume conversely that there exists a vertex $y \in (S)_i$, where the column i satisfies $a_{j-1} < i < a_j$ such that $H(y) \le H(x_j)$. Let $S = S^1 | S^2$, where S^2 is the subschedule that starts with $(S)_i$. Let G^2 be the subgraph induced by S^2. In G^2 the vertex x_j is a root of an outtree component. If y belongs to an intree component; that is, it is a root of an intree in G^2, then by Lemma 10 we conclude that there exists an optimal schedule, S' for G^2 which starts with x_j. Because of the principle of optimality that schedule contradicts the minimality of S w.r.t. π_x, as we can decrease a_j without affecting $a_1, \ldots, a_{j-1}$. The same contradiction arises in the case where $(S)_i$ does not contain m vertices.

The only case left is the case where y belongs to a intree and there exists m vertices in $(S^2)_1$. Note that in the previous case we did not use the fact that $m = 3$, but here we do have to use it. So, $(S^2)_1$ contains 3 vertices and y is a root of an outtree. But this implies that there are at least 3 components in G^2, one of x, one of y, and at least one of the other vertices in $(S)_1$. We have assumed that $H(y) \le H(x_j)$ and therefore at least one of the vertices in $(S^2)_1$ is under the median of G^2. By the Elite Theorem or by the Basic Theorem we can find another optimal sch- starting with x_j and again contradicting the minimality

of S w.r.t. π_z. This completes the proof of the lemma. ∎

Note that in the proof of Lemma 12 we use the fact that $m = 3$ only in comparisons with other outtree components, therefore, in general we can prove that for every m there exists an optimal schedule in which vertices along an outtree path precede inforest vertices of the same height.

The following theorem presents the key idea of the algorithm we present later for optimal scheduling of fan graphs on three processors.

Theorem 13. For $m = 3$ and G a fan graph, let A be a set of up to three maximal height roots of G. If A contains a root of an outtree which is highest in G, then there exists an optimal schedule for G which starts with the set A.

Proof. If G is of one level the theorem is trivially true. Otherwise, let A be a set of three highest vertices satisfying the conditions of the theorem. Let $x \in$ A be the root of the highest outtrce. By Theorem 11, there exists an optimal schedule which starts with x. Denote by π_x the maximal path in G which starts with x. Let S(G) be an optimal schedule for G which is minimal w.r.t. π_x. If A contains less than 3 vertices there must be less than 3 roots in G and in such a case $A \subseteq (S)_1$. Therefore assume A $= \{x, y, z\}$ and $(S)_1$ = {z, v, w}. We want to prove that there exists an optimal schedule which starts with A. Assume that y is not v or that w is not z; otherwise $(S)_1$ is A and we have nothing else to prove. By the definition of A we know that $H(y) \geq_G H(v)$ and $H(z) \geq_G H(w)$. This inequality holds up to renaming of identical vertices, because it might be that y is v or z is w, but not both. Let π_y, π_z, π_v, and π_w be the maximal paths in G which start at y, z, u, and w, respectively. Note that it may be that some of these paths (or even all of them) coincide somewhere in G. According to the definitions it is clear that vertices along the paths appear in different columns of S in the order induced by the path. Let k be the first column in S at which the number of vertices along the path π_y equal to the corresponding number along π_v and the number along the path π_z is equal to the number along π_w. Let S = $S^1 | S^2$, where S^1 is the first k columns of S and S^2 is the rest of them. Denote by G^1 the graph induced by S^1. Figure 8 describes schematically the structure of the subgraphs and the paths.

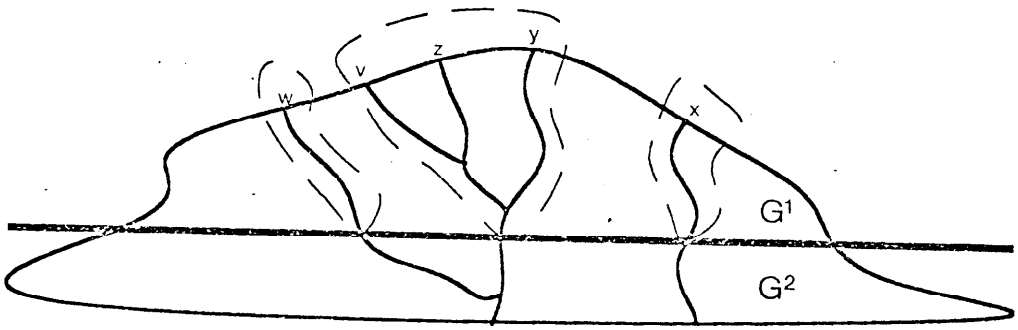


Figure 8.

The way WC defined k implies that in G^1,

$$H(y) \geq_{G^1} H(v) \text{ and } H(z) \geq_{G^1} H(w) \qquad (4-1).$$

The minimality of S w.r.t. π_z and the fact that z is highest root in G imply, by Lemma 12, that in G^1,

$$H(z) \geq_{G^1} H(y) \text{ and } H(z) \geq_{G^1} H(z) \qquad (4-2).$$

Therefore, z is highest root of an outtree in G^1. We assume that either y or z is not in $(S)_1$, this fact and (4 — 1) imply that in the first k columns at least one of the paths is disjoint from the others, otherwise k would not be the first column at which the numbers are equal. G is a fan graph and therefore the disjoint paths are in a disjoint component of G^1. So, the above vertices belong to at least two components of G^1 and none of them is the component to which the root z belongs. Therefore, G^1 has at least three components. Equations (4 — 1) and (4 — 2) imply that at least one of the two vertices, v and w, is under the median of G^1. More precisely, it can be show by checking the various possibilities that every vertex in $(S)_1$ which is not in A is under the median. The Elite Theorem proves that in this case there exists an optimal schedule for G^1 which starts with {z, y, z} as desired. ∎

Theorem 13 suffices for proving the correctness and linearity *of* the Flip-Flop algorithm. One can prove a slightly stronger version of Theorem 14: it is sufficient.

to ensure that the set **A** is **a set** of up to three maximal-height roots **of the graph** which contains an outtrce root. Thus, it is not **necessary** that the outtrce **root be** the highest: it can be only one of the three highest ones.

5. The Flip-Flop algorithm.

The precise flip-flop algorithm we **describe** here is more **efficient** than the intuitive algorithm we described in section 4. The method is this one: we remove roots from the top and from the bottom of the preccdcncc graph, but we stop this flip-flop **process** and continue by an HLF algorithm as soon as we are able to **do** so. Our ability to stop the **flip-flop** process ensures the linearity ***of*** the algorithm.

To decide when we can apply an HLF algorithm for optimal scheduling on 3 processors, we use three of the results we have proved previously. The first result is case (iii) of the Elite Theorem; according to that result, if the Elite of **the** precedence graph is empty (E(G) = $\emptyset$) then WC can obtain an optimal schedule for the graph by using an HLF algorithm. The second result is Theorem 8; which says that if we have above the median either only outtrce or only **intree** components then we can also obtain an optimal schedule using an HLF. The third result, that enables us to decide easily if we can apply an HLF to obtain an optimal schedule, is Corollary 7; it states that we can do so if the number **of** the roots above the median is less than four ($|\mathcal{E}(G)| \leq$ m).

We say that the HLF-condition holds for the precedence graph if one of the above conditions holds; therefore, if the HLF-condition holds we can then obtain an optimal schedule **by** using an IILF algorithm.

The only remaining case in which the HLF-condition does not hold **is when** the precedence graph has exactly two highest components above the median: **an outtree** and an **intree** with more than two roots strictly higher than any other component of the graph. During the execution of the Flip-Flop algorithm we use a test function $HLF?(G)$ to check if **the** HLF-condition holds.

Let G **be** a fan graph, denote by G_0 the **highest outtree** component **of** G and by G_1 the highest **intree** component ***of*** G. We call these two components the ***principal components*** of the graph. Let h_0, h_1 be the heights of the two **principal components, respectively,** and let h_3 be the height of a third highest component

of G. If one of these heights is not dcfincd, set the corresponding h to be zero. The two principal components G_O, G_I and the triple $< h_O, h_I, h_3 >$ enable us to test easily if the HLF-condition holds **for** G. Note that all the five notions defined above are the same for the graph and for its reverse with the only exception that **"O"** and **"I"** are exchanged because an **outtree** of the graph is an **intree** of its reverse and similarly for an intrce.

Function *H LF?(G):*

HLF?=FALSE;
If $\max(h_O, h_I) \leq h_3 + 1$ **then HLF?=TRUE fi;**
If $\min(h_O, h_I) \leq h_3 + 1$ **then HLF?=TRUE fi;**
If the three highest roots of G_I arc not higher than $h_3 + 1$;
then **HLF?=TRUE fi.**

The following lemma proves that the function $HLF?(G)$ computes exactly the HLF-condition.

Lemma *14.* $HLF?(G)$**=TRUE if and only if the HLF-condition holds for G.**

Proof. We prove that the subconditions of the function HLF? are equivalent to the conditions of the three results used to **define** the **HLF-condition.**

Claim (i): $\max(h_O, h_I) \leq h_3 + 1$ if and only if E(G) $= 0$.

Proof. From the definition of the triple $< ho, h_I, h_3 >$ we know that

$$\max(h_O, h_I) \geq h_3. \qquad (5-1)$$

Therefore the left hand side of Claim (i) holds only in the case where all the three highest components of the graph are not lower than h_3 and at the same time are not higher than $h_3 + 1$. This **implies** that

$$\mu(G) = h_3 + 1. \qquad (5-2)$$

The two equations (5-1) and *(5-2)* imply that no component is higher than the median of the graph which implies that its Elite is empty, E(G) $= 0$. The second direction is very similar to what we have just proved and therefore **will** be omitted.

Claim (ii): $\max(h_O, h_I) > h_3 + 1$ and $\min(h_O, h_I) \leq h_3 + 1$ if and only if the graph has either only intrce component or only outtree component above the median.

Proof. Assume that h_3 is an outtree, then by definition $h_O \geq h_3$. If the left hand side of Claim (ii) holds then we also know that

$$h_O > h_3, \quad h_O > h_I \text{ and } h_3 \geq h_I - 1. \qquad (5-3)$$

This implies that $h_I < \mu(G)$, otherwise we would have at least three components above the median which contradicts the definition of the median. By definition, h_I is the highest intrce component, thcrcforo WC only have outtree components above the median, which proves the claim. The case where h_3 is an intree is *exactly* the same. The other direction of Claim (ii) is similar and therefore omitted.

The only remaining possibility is the case

> $\min(h_O, h_I) > h_3 + 1$ and the three highest roots of G_I are not higher than $h_3 + 1$, and the outtree G_O is higher than $h_3 + 1$.

It is easy to show that this case is equivalent to the one where the first two results do not hold and the Elite of the graph contains less than m roots. This completes the proof of the lemma. ∎

Thc definition of the function $HLF?(G)$ and Lemma 14 imply the following simple result.

Corollary 15. If $HLF?(G)$=TRUE then the only two components above the median of G are G_I and G_O.

We are now ready to present the Flip-Flop algorithm. The Flip-Flop algorithm produces an optimal schedule if the precedence graph is a fan graph and the number of processors is three. The algorithm removes roots from the graph and inserts them into tableaux. The roots from the top are inserted in the tableau T_{top} and those from the bottom in T_{bot}. Each tableau has three rows (one for each processor). The variable $\hat{G}$, which we use in the algorithm represents the precedence graph remaining at thc current step of the algorithm. Initially $\hat{G}$ is equal to G and after each removal of vertices from the graph the variable $\hat{G}$ becomes the resulting subgraph.

The Flip-Flop Algorithm *(m = 3)*

[1] **While not** $HLF?(\dot{G})$ **and** the **outtree** component is the highest component **do:**

begin
remove from $\dot{G}$ **the** root of the highest **outtree** and any other two highest roots. Add these three roots as the next column in the tableau T_{top};
end;

[2] **While not** $HLF?(\dot{G}^R)$ **and** the **outtree** component is the highest component do:

begin
remove from $\dot{G}^R$ the root of **the** highest **outtree** and any other two highest roots. Add these three roots as the next column in **the tableau** T_{bot};
end;

[3] **If** $HLF?(\dot{G})$ **then** continue to fill T_{top} by an HLF algorithm applied to $\dot{G}$ **fl.**

[4] **If** $HLF?(\dot{G}^R)$ **then** continue to **fill** T_{bot} by an HLF algorithm applied to $\dot{G}^R$ **fl.**

[5] **If** $\dot{G}^R$ is not empty **then** return to step [1] **fl.**

[6] If $\dot{G}^R$ is empty **then** output the tableau $T_{top} \mid \mathbf{reverse}(T_{bot})$ as an optimal schedule **fl.**

Note that the expression "$T_{top} \mid \mathbf{reverse}(T_{bot})$" represents the tableau we get by attaching to the end **of** the tableau T_{top} the reversed tableau of T_{bot}.

Before we prove that the Flip-Flop algorithm produces an optimal schedule let us demonstrate it on the precedence graph given in Figure 9.

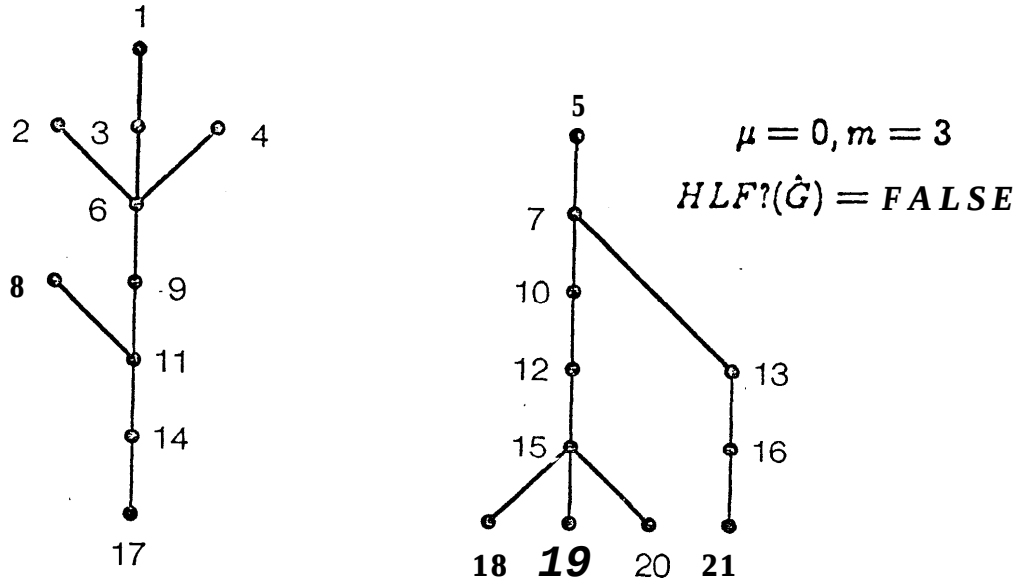


Figure 9.

The graph has two components, one outtrcc and one intrcc. We observe that the $HLF?(G)$ test fails and that the intrcc compoocnt is higher. Therefore we look at the bottom of the graph and find "17" as the root of the highest outtrce (of G^R). Thus we can take {17, 18, 19) and put them as the first column in the tableau T_{bot}. After removing this triple from the graph we arc left with the following subgraph :

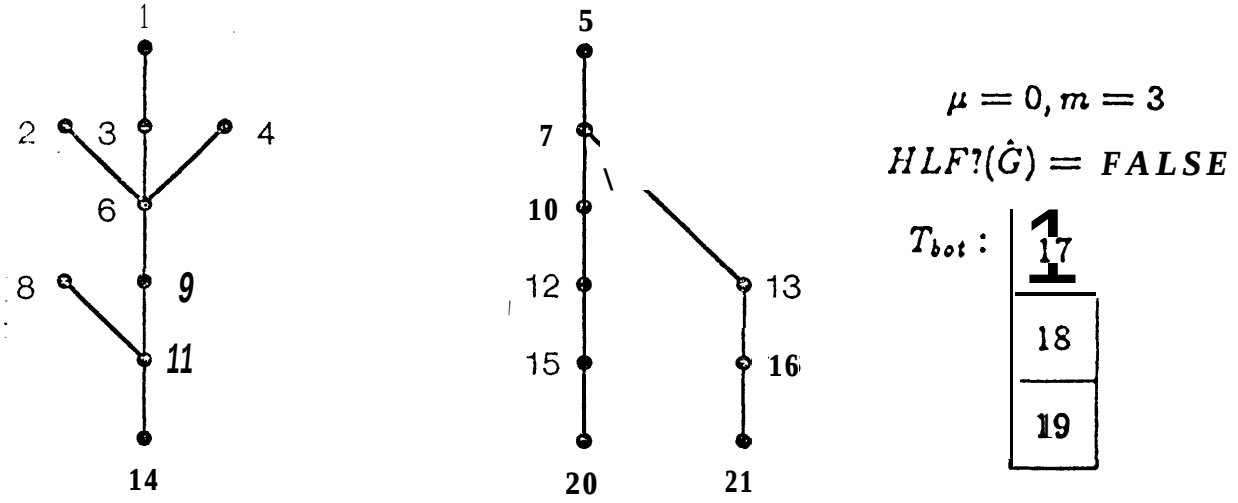
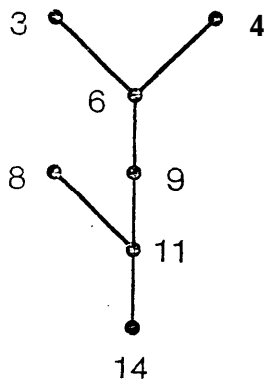
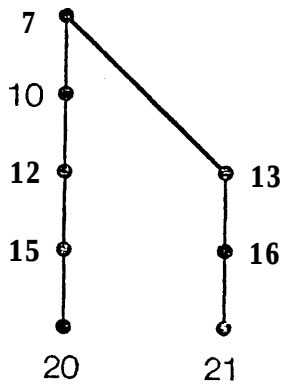


Figure 10.

Now, both the top and the bottom contain outtrcc roots of maximal height.

We can choose either of the sides to continue the algorithm. Note that according to the Flip-Flop algorithm we have to continue from the bottom. This is done to prevent too many "reverses", but for the demonstration we continue from the top. So, we can choose $\{5, 1, 2\}$ and put them as the first column in the top-tableau. We remove these roots from the graph and arc left with the subgraph which appears in Figure 11.





$\mu = 0, m = 3$

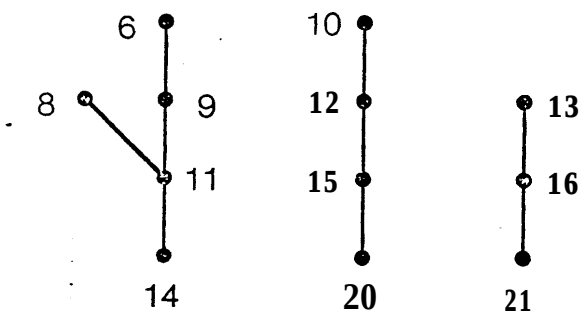
$HLF?(\dot{G}) = FALSE$

T_{top} :	5
	1
	2

T_{bot} :	17
	18
	19

Figure 11.

Again, we can choose either side of the resulting graph to proceed let use chodse the top. Then, we rcmovc $\{7, 3, 4\}$ from the graph and insert them as the next column in the toptablcau. We get now the graph in Figure 12.



$\mu = 4, m = 3$

$HLF?(\dot{G}) = TRUE$

T_{top} :	5	7
	1	3
	2	4

T_{bot} :	17
	18
	19

Figure 12.

The Elite of the graph that we have just got is empty and therefore by the Elite Theorem we know that HLF produces an optimal schedule for the resulting graph. Thus, $HLF?=TRUE$. So, we cm complete the tableau T_{top} according to

HLF and get,

T_{top} :

5	7	6	9	11	14
1	2	10	12	15	20
3	4	8	13	16	21

T_{bot} :

17
18
19

The concatenation *of* the two tableaux is an optimal schedule for the **graph C.** Note that the tableau we have got is a level tableau, but not every level tableau is an optimal schedule. If we apply the algorithm **to** the various **examples, given** in the paper, we would find that in some cases **we get *a* schedule which is not *a*** level schedule.

Theorem 16. The Flip-Flop algorithm producer *an* optimal **schedule for *G* on three processors.**

Proof. First we have to show that the algorithm terminates. For **this, it** is enough to show that for any fan graph at least one of the cases in the algorithm holds. The **cases** are complementary, **except** for the condition of having **a highest outtree** component. But it is obviously true that if the graph does not **contain** highest outtrec component then its **reverse** contains one; thus, at any step of **the** algorithm at least one *of* the cases in the algorithm holds. To complete the proof of termination, observe that at every step of the algorithm we **decrease** the size of the graph and therefore after a finite number of **steps** the algorithm terminates.

The optimality of the final tableau can be deduced from **the** fundamental principles of optimal schcdulcs. The **reverse** principle and the principle **of op timality** imply that if at **every** step of the algorithm we **remove** columns **that** belong to optimal schcdulcs then the whole **tableau** that we **get** at step **[6]** is **an** optimal schedule for the precedence graph. Therefore it remains to show that **all the** columns belong to optimal schedules. Theorem 13 **ensures** that **the** columns we get in the steps **[1]** and **[2] belong** to optimal schedules. The Elite Theorem, Corollary 7 and Thcorcm 8 imply that the columns that we produce in steps **[3]** and **[4] also** belong to optimal schedules. Therefore, we have proved the optimality of the schedule produced by the Flip-Flop algorithm. ∎

Note that the algorithm can be further improved using the remark we made after Theorem 13. Due to this remark we immediately know that we can start an optimal algorithm with the triple $\{1,5,3\}$. Indeed, that triple is a set of three maximal height roots containing a root of an outtrcc. This improvement can be added to the algorithm but it does not seem that its contribution is significant.

To prove the linearity of the Flip-Flop algorithm we have to describe carefully the data structures that we use and the precise implementation of the algorithm. WC assume that the preccdcncc graph is given by the regular adjacency lists [AH74] with the only exception that the list is dual; that is, there arc also pointers that point back. The motivation behind this data structure, which is called *Dual Linked-List,* is to enable us to remove easily vertices from either the top or the bottom of the graph, level after level.

WC supplement every vertex in the Dual Linked-List with some more information. Every vertex contains a counter with the number of vertices pointing to it from above (parents), and a counter for the number of its chidren. To each vertex we attach a bit that denotes if it belongs to an intrcc or to an outtree component. Note that if the graph is a chain of vertices it can be defined either as an intrce or. as an outtrce; in our case it is not essential which type we choose to define it as, except that being an outtrce can sometimes improve the algorithm. The height of a vertex in the graph is different from its height in the reverse graph, which WC call the depth of the vertex. Therefore, we also attach to each vertex two numbers: one is its height in the graph G and the other is its depth. All the above attachments can be done in a linear time.

The next problem we have to face is how to manage the list of the roots of the graph and its reverse, The list has to enable us to easily find the roots according to their height and to let us easily remove roots. These properties are those of hashing schemes [Kn73]. Thus, we can keep the list of the roots of the graph in a hash table. To each level of the graph there correspond a row in the hash table. This row points to a linked list of all the roots of that height. For easy insertion in the hash table WC can keep at each vertex, in addition to its height, a pointer to the corresponding row in the hash table. Deletion from the hash table can be done row after row and within the rows according to the linked list. Thus, the first vertex in the hash table is the first vertex of the first noncmpty row.

Note that our special form of the hash-tables and data structure ensures that every insertion or deletion takes a constant number of steps. There are no collisions during the insertions because of the queues.

Everything we have described until now can help us to implement HLF in a linear number of steps. But in the Flip-Flop algorithm we change the height of some components by either removing from the top or from the bottom of the graph, therefore we cannot keep all the roots of the graph in the same hash table. The problem does not exist if all the roots in the hash table arc of the same component, because in that case the height of all of them is changed at the same time by the same amount. To be more precise, assume that the hash table contains the roots of an intrce, if we remove from that intrcc the leaf, which is the root of the reversed component then the height of all the roots in the hash table is decreased by exactly one. So, to bc consistent with the structure we can keep roots in the same hash table if either all of them belong to the same component or all of them arc going to be removed from the table without changing their height. In the complete implementation we will have four hash tables.

The four hash tables arc H_I, H_O, H and H^R. In the hash table H_I we keep all the roots of a highest intrce component of the precedence graph. The roots are inscrtcd, as described before, according to their height in the graph. The reverse of this component is an outtrce and therefore contains exactly one root. The pointer r_I points to that root. In the hash table H_O WC hold all the roots of the reverse of a highest outtrce component; that is, its leaves. These leaves are inserted according to their depth in the graph. The pointer r_O points to the root of this outtrce component. The rest of the roots of the graph are inserted according to their height into the hash table H, and the rest of the leaves in the hash table H^R, according to their depth. In the complete implementation of the algorithm we will show how these four hash tables solve the problem.

The complete implementation oj the Flip-Flop algorithm:

[Initialization]

Construct thc Dual Linked-List to represent the precedence graph and supplement every vertex with the information we rcquircd. Define G_I and G_O, as described above. Construct the four hash tables. If there is no outtree component then set h_I to zero, otherwise set it to be the height of the component G_I. Similarly set h_O. Set h_3 to be the height of the highest component of the graph, other than G_I and G_O. Set $\hat{G}$ to be G.

[1] The test $HLF?(\hat{G})$ can be implemented easily, as noted in the definition of the function $HLF?(G)$. If the test is FALSE then compare the heights, execute the body of step [1] only if $H(G_O) \geq H(G_I)$. Insert the root r_O and the first two roots in the hash table H_I as the next row in the tableau

T_{top}. Remove the two roots from H_1 and from the graph $\hat{G}$. Insert their chidren in H_1 according to their heights in the original precedence graph G. Note that their heights in $\hat{G}$ might be different from their heights in G, but the difference is fixed for all of them and *for* all the roots which are already in H_1. Update the value of h_1, its value can either remain the same or be decreased by one. Set G_1 to be the remaining intree. Now observe that the removing of r_0 might break G_0 in some outtrce components. In such case we cannot leave all the leaves of G_0 in H_0 because they do not belong to the same component any more. So, after removing r_0 from G_0 set G_0 and $\hat{G}$ to be the remaining graphs. Decrease h_0 by one and set r_0 to point to the highest child of the old r_0. Scan the components of the chidren other than the new r_0, update the heights and depths, find the leaves and transfer them from H_0 to H^R. Insert the leaves according to their depth in the new $\hat{G}$. Insert the chidrcn, other than r_0, in the hash table H_0 according to their new heights. If one of these chidren is higher than h_3 than set h_3 to be the maximal-height of them.

[2] Step two is done exactly as step one with the exception that we look at the reversed graph. Thus, by exchanging "0" with "1", "H" with "H^R", and "T_{top}" with "T_{bot}".

[3] If at the end the test $HLF?(\hat{G})$ becomes TRUE, then enter the HLF step. Scan G_0 to update the heights of its vcrticcs. Insert the root r_0 of the outtrce G_0 in the hash table H according to its new height. Update the heights of the vertices in G_1, the height of every vertex in this component is the difference between its original height in G and the height in G of the leave r_1. Merge the two hash tables H_1 and H into one; that is, transfer all the roots in H_1 to H according to their heights in $\hat{G}$. Remove from H three roots at a time, according to their order. Insert the removed roots as the next row in the tableau T_{top}. Remove these roots also from $\hat{G}$ and insert their chidren in H. Continue this process until $\hat{G}$ is empty. If at some point there are less than three roots in H then remove as many as there exist and complete the row with empty places.

[4] Step four is done exactly as in the previous stop, merge H_0 with H^R and put the roots in the tableau T_{bot}.

[5] Test if the remaining graph is empty; if not then return to step [1].

(6) If the graph is empty then take the tableau T_{top} and attach to it the reversed

tableau T_{bot}. We attach the reverse of T_{bot} because it describes the schedule from the end to the beginning. The resulting tableau is the desired optimal schedule.

Theorem 17. The Flip-Flop algorithm is a linear algorithm

Proof. As we pointed out during the detailed description of the implementation of the Flip-Flop algorithm, all the constructions of the data structures are linear. During the algorithm we insert every vertex in the hash table at most twice and remove it at most twice. We look at each vertex during the execution of the algorithm at most twice, before transferring it to the tables. The only suspicious step is the search that we do for updating the heights of the subcomponents of G_I and G_O, but these searches are done at most once for every branch of the graph. The last thing we have to show for proving that the implementation is linear is to show an efficient way to find the first three empty rows in each hash table and keep track of them. We can do this by having three pointers pointing, at the beginning, to the first three empty rows. Whenever we delete vertices from the hash table we update them as WC update h_o. Using the fact that the precedence graph is a fan graph, one can prove that during the insertions we increase the value of such a pointer at most twice.

Note that, as we proved in Thcorcm 16, at each step of the algorithm at least one of the cases holds. This completes the proof that the algorithm can be implemented in a linear number of steps. ∎

All the above reasoning also proves that the algorithm is not only linear, but that it is linear with a fairly small constant. Note that the implementation we described is also an efficient implementation of the HLF algorithm for any directed graph.

Appendix

For the formal proof of the Basic Theorem we need the following lemma which will be used recursively in the proof of the Basic Theorem. Recall that the graph G is *a conflict-free* graph if it satisfies the Basic Theorem and if all its subgraphs also satisfy the Basic Theorem.

Lemma A. Let $G = (V, E)$ be a conflict-free graph. Let A be any set of vertices all strictly lower than $\mu(G)$. Denote by h the maximal height of the vertices in A. Let w $\notin$ A be a root of G such that $H(w) \geq h$. If G has at least $m - 1$ components of height $\geq h - 1$ not containing vertices in A, then there exists an optimal schedule $S = S^1 | S^2 | S^3$ (therefore $G = G^1 | G^2 | G^3$), where $w \in S^2 =$ (S), such that *one* of the following eases holds.

(i) $S^2 = (y, w, z_3, \ldots, z_m)$ and $A \subseteq V^3$ (the vertices ot G^3); that is, w appears before any vertex of the set A;

or

(ii) $S^2 = (g, w, z_3, \ldots, z_m)$ and $g \in A$, having the properties:

(P1) every vertex x of G^1 satisfies $H(x) > \min(\mu(G), H(w))$ and $x \notin$ A;

(P2) for every root, x, of G^3 the following holds, $H(x) < h$, *or* x has a parent in G^2, or $x \in A$;

(P3) every component of G with height not less than h which does not contain vertices from A has exactly one representative in S^2. The vertex g is the only representative of all the other components;

(P4) $H(g) = h$;

(P5) $\mu(G^3) > h - 1$.

Note that A can be any set of vertices under the median; it may include roots and nonroots, or it may be empty. The only restriction is that there are enough higher components not containing vertices from A. One possibility is that all the elements of A arc taken from a single component. The root w can even be in the same component as vertices from A. The case where A is empty obviously leads to case (i) of the theorem.

Proof. If G has less than m components then $\mu(G) = 0$ and the theorem trivially holds. Therefore, let us assume that it has more than $(m - 1)$ components; that

is, $\mu(G) \neq 0$. We also assumed that w $\notin$ A.

We prove that if (i) does not hold then (ii) will hold. Let S be an optimal schedule which is minimal w.r.t. w (see Definition 1). Assume $w \in$ (S), and let x be the carliest vertex in S such that $H(x) \leq \min(H(w), \mu(G))$. Assume $x \in (S)_l$; if $l >$ u then (i) holds, therefore assume that $l \leq$ u.

Case $l <$ u : let $S = S^1 | S^2$, where $S^1 = (S)_1 | \ldots | (S)_{l-1}$. Then x is a root of G^2. By the construction we know that S^2 is an optimal schedule for G^2 and that Theorem 2 implies $\mu(G^2) \geq \mu(G)$. G^2 is *a* subgraph of a conflict-free graph; therefore it is itself conflict-free, so the Basic Theorem holds for it. The conditions of the Basic Theorem are satisfied by w and x because

$$H(x) \leq \mu(G) \leq \mu(G^2) \text{ and } H(x) \leq H(w),$$

and S^2 starts with x and not w. Therefore there exists another optimal schedule $\hat{S}^2$ for G^2 which starts with w. But in this case, by the principle of optimality (Theorem 1), the schedule $\hat{S} = \hat{S}^1 | \hat{S}^2$ is optimal for G and contradicts the minimality of S with respect to w.

Thus, we are left with the case $l =$ u. If the column (S), does not contain any vertex from the set A then we are again in case (i), otherwise let g be the highest vertex of A in that column. We can partition the optimal schedule into $S = S^1 | S^2 | S^3$ where S^2 = (S),, S^1 is the part of S before S^2, and S^3 is the part of S after the column S^2. We also know that all the vertices in S^1 are above the median and that w and g are in S^2. Without loss of generality, let $S^2 = (g, w, z_3, \ldots, z_m)$ where g $\in$ A. Let G^1, G^2 and G^3 be the corresponding subgraphs of G.

So far we actually proved that :

> if (i) does not hold for any optimal schedule then every optimal schedule S for G which is minimal w.r.t. w satisfies the following: for every vertex x in G', $H(x) \geq \min(H(w), \mu(G))$; (1)

which proves (P1).

Note that according to the above proof, for every optimal schedule minimal w.r.t. w (Definition 1), the column S^2 is uniquely determined to be the u-th column. Now, among all of these optimal schedules we choose those which have

the least number of vertices from A in the u-th column. The least number is not zero because WC assumed that case (i) does not hold and zero leads to case (i). Let g be the highest vertex of A which appears in the u-th column among all such optimal schedules. Assume, without loss of generality, that S is such an optimal schedule containing g in its u-th column. For further references we call the property of an optimal schedule to have the least number of vertices of A in its u-th column, the minimality w.r.t. A.

Denote by $\tilde{G}^3$ the graph induced by $S^2 | S^3$. By assumption, it is a conflict-free graph. From (P1) and Theorem 2 we know that

$$\mu(\tilde{G}^3) \geq \min(\mu(G), H(w) + 1),$$

we are considering the case $\mu(G) > 0$ and therefore $\tilde{G}^3$ has at least m components. $S^2 | S^3$ is an optimal schedule for $\tilde{G}^3$ and starts with g, where $H(g) \leq \mu(\tilde{G}^3)$. Let v be a root of G^3 which has no parent in G^2 and is not included in A. $\tilde{G}^3$ is conflict-free and therefore if $H(u) \geq H(g)$ then there exists an optimal schedule S' for $\tilde{G}^3$ in which g and u are exchanged in the first column. But this implies that $S^1 | S'$ is an optimal schedule for G which contradicts the minimaliy of S w.r.t. A. Therefore, we have proved

> for every v in G^3, $H(v) < H(g)$, or v has a parent in G^2, or v is in A. **(2)**

Now assume that G_a is a component of G such that $H(G_a) \geq H(g)$ and G_a does not contain vertices from A. By (2) we know that every vertex v of G_a with height H(u) $\geq H(g)$ is in S'. G_a must have a vertex of height $H(g)$; let v be such a vertex. Recall that we assumed $v \notin A$. By (P1) WC know that v is not in S^1, so therefore it must be in S^2. There are at least $m - 1$ such components, and therefore every one of them has exactly one vertex in S^2, and g is the only representative of all the other components. So, WC have proved

> *every* component of G that has height $\geq H(g)$ and does not contain vertices from A has exactly one representative in S^2 and g is the only one from all the other components of C. **(3)**

Note that if w belongs to a component which contains a vertex from A then (2) and (3) imply that (i) holds.

From (3) we conclude that there are exactly m components of G which are of height $H(g)$ or more (including the component of g itself). This fact implies that if g is not of maximal height in A then there must be another root which belongs to a component which contains a vertex from A, and is higher than g and is in $\bar{G}^3$. $\bar{G}^3$ is conflictfree and therefore we can find another optimal schedule for $\bar{G}^3$ in which we exchange that root with g in the first column. Either such an optimal schedule contradicts the minimality of S w.r.t. A, or g is not the highest vertex in A appearing in the u-th column of the corresponding optimal schedule. This contradiction implies that $(P4)$ holds; that is, $H(g) = h$. $(P4)$ enables us to conclude from (2) and (3), respectively that $(P2)$ and $(P3)$, hold..

To finish the proof we just have prove $(P5)$. Let x be any root of $\bar{G}^3$ having the property $H(x) \geq \mu(\bar{G}^3)$. Then by $(P2)$, $(P3)$ and $(P4)$ it is clear that $x \notin G^3$ and therefore, by Theorem 2, $\mu(G^3) \geq \mu(\bar{G}^3) - 1$. On the other hand

$$\mu(\bar{G}^3) \geq \min(\mu(G), H(w) + 1) > H(g),$$

which implies that $\mu(G^3) > H(g) - 1$ = h-l, as desired. ∎

Using Lemma A we are ready to prove the basic theorem. Intuitively, the idea is to find a path in the graph G, starting at g and a set of vertices from the component to which w belongs. We construct an optimal schedule in which the vertices along the path appear in such a way that they can be exchanged one by one with the vertices from the component of w without violating the ordering induced by G.

Proof of the Basic Theorem:

Using Lemma A rccursivcly, we prove the theorem by complete induction *on* the prcccdcncc graph G. Let G be a DAG and assume that all its subgraphs satisfy Theorem 5; that is, they are conflict-free. We prove that this assumption implies that G itself is conflict-free.

In the following two cases the theorem holds trivially:

Case $\mu(G) = 0$: This case occurs when G has less than m components but then the theorem is trivially true.

Case $H(G) = 0$: This case occurs when G is a collection of vertices. In this case the theorem also trivially holds.

Now, let G bc a graph having only conflict-free subgraphs. Let S be an optimal schedule for G. Assume $(S)_1 = (g^0, x_2, \ldots, x_m)$, where $H(g^0) \leq \mu(G)$. Let w^1 be any root of G such that $H(w^1) \geq H(g^0)$ and $w^1 \notin (S)_1$. We want to prove that there exists another optimal schedule S' for G such that $(S')_1 = (w^1, x_2, \ldots, x_m)$.

Denote by G_0 the subgraph of G obtained by deleting $(S)_1$ from G. By assumption, G_0 is a conflictfree graph. The following claim determines the median of G_0.

Claim:

(a) If $H(g) < \mu(G)$ then $\mu(G_0) \geq \mu(G) - 1$.

(b) If $H(g) = \mu(G)$ then $\mu(G_0) \geq \mu(G)$.

Proof. From the first case of Theorem 2 we conclude (a) immediately. But if $H(g) = \mu(G)$ then by condition (ii) of the Basic Thcorcm we know that all the vertices in $(S)_1$ are not lower than the median; Corollary 3 implies (b).

If $\mu(G_0) = 0$ then, by the previous claim, either $H(g) = 0$ or $\mu(G) = 0$. This implies that in any case g is an isolated vertex of G. Therefore g can be exchanged with w without destroying the optimality of the new tableau we get.

Thus, let us assume that $\mu(G_0) > 0$; that is, G_0 has more than $m - 1$ components. To apply Lemma A to the subgraph G_0, let A_0 be the set of all chidrcn of g^0. By the above claim it is clear that for every $x \in A$, $H(x) < \mu(G_0)$ and $H(x) < H(w^1)$.

We use the following indexed notations for the variables in Lemma A. Denote

$$G, A, w \quad \text{by} \quad G_{i-1}, A_{i-1}, w^i$$

respectively; and

$$S^1, G^3, g, \{z_3, \ldots, z_m\} \quad \text{by} \quad S^1_i, G_i, g^i, \{z^i_3, \ldots, z^i_m\}.$$

With this notation the conditions for Lemma A can be written:

(C1) G_{i-1} is conflict-free.

(C2) $w^t \notin A_{t-1}$ and is a root of G_{t-1}.

(C3) **For** every $x \in A_{t-1}$, $H(x) < \mu(G_{t-1})$ and $H(x) \leq H(w^t)$.

(C4) There exist at least $m - 1$ components which arc higher than the vertices in A_{t-1} and which do not include any of these vertices.

These four conditions enable us to apply Lemma **A.** Using the above notation we can describe the **results** of Lemma A as follows:

There exists an optimal schedule $S_t = S_t^1 \mid S_t^2 \mid S_t^3$ for G_{t-1} such that one of the following **cases** hold:

(i) $S_t^2 = (y, w^t, z_3^t, \ldots, z_m^t)$ and $A, \subseteq V_t$ (V_t is the set of vertices of G,) **where G, is** the **subgraph** of G_{t-1} induced by S_t^3.

(ii) $S_t^2 = (y, w^t, z_3^t, \ldots, z_m^t)$ and $g^t \in A$, having the properties **:**

(R1) No **vertex** of A, appears in S_t^1.

(R2) No vertex of **A,** appears in $\{z_3^t, \ldots, z_m^t\}$.

(R3) No vertex from $\{z_3^t, \ldots, z_m^t\}$ belongs to the component to which w belongs.

(R4) $\mu(G_t) > H(g^t) - 1$.

Result **(R1)** holds by **(P1)** of Lemma A, (R2) is implied **by** *(P2)* **and (P3).** Result *(R3)* **is** also implied by *(P3)* and *(R4)* is just **(P5).**

We now show that we can apply Lemma A for $t = 1$ and also that whenever case (i) **does** not hold we can reapply the Lemma again on the resulting graph. **The conditions** (Cl) to *(C4)* hold for $t = 1$ for the following reasons. The inductive assumption implies that G_0 is a **conflict-free** graph; the assumptions that w **is** a root of G and that $H(w) \geq H(g^0)$ imply (C2). A_0 is **defined** to be the set of **the** chidren of g^0 and therefore the height of its maximal vertex is $H(g^0) - 1$. **By the** above claim, as we have pointed out before, (C3) holds. The case $H(G_0) = 0$ *was* excluded before; thcrcforc, we can assume that $\mu(G_0) > 0$. This implies that there are $m - 1$ components of G in which g^0 does not **appear** and therefore there are at least m - l components of G which do not contain chidren of g^0. By the above claim it is clear that the chidrcn of g^0 are strictly under the median of G_0, **and**

therefore at least m- 1 components which do not contain chidren of g^0 are higher than A_0 in G_0. We now prove that after every iteration, if case (ii) holds then we can make another iteration. So, assume that (ii) holds after the ι-th iteration; that is, the results (R1) to (R4) hold. By complete induction every subgraph of G is conflict-free and hence G,. Define A_ι to be the set of the chidren of g^ι, then the height of the maximal vertex in the set A_ι is $H(g^\iota) - 1$. Choose $w^{\iota+1}$ to be the highest root of the component to which w^ι belongs; that is, the highest root after the deletion of w^ι. This choice implies that $H(w^\iota) - 1 \leq H(w^{\iota+1})$. (R4) implies that $\mu(G_\iota) \leq H(g^\iota) - 1$ and therefore for every $x \in A_\iota$,

$$H(x) \leq H(g^\iota) - 1 < \mu(G_\iota)$$

and

$$H(x) \leq H(g^\iota) - 1 \leq H(w^\iota) - 1 \leq H(w^{\iota+1}),$$

which imply that (C3) holds. The choice of $w^{\iota+1}$ immediately implies (C2). In the subgraph $G^2_{\iota-1}$, g^ι was strictly under the median $\mu(G_{\iota-1})$ and therefore there are at least $m - 1$ components of G_ι which are higher than its chidren and obviously do not contain any of them. This proves (C4), and thus we have shown that if case (ii) holds we can make another iteration.

Whenever (i) does not hold we can continue the above iteration, but in each iteration the height of g decreases, $H(g^\iota) = H(g^{\iota-1}) - 1$, and therefore after a finite number of iterations WC should get either that (i) holds or that for some μ, $H(g^\mu) = 0$. This last case implies that $A_{\mu-1}$ is empty which ensures that Case (i) also holds in the μ-th iteration.

So, assume that after ν iterations Case (i) holds. By the principle of optimality (Theorem 1) we know that the compound schedule constructed from the optimal subschedules obtained during these ν iterations is optimal.

This optimal schedule has the structure which is presented in Figure 4. For every ι, $1 < \iota \leq \nu$, this structure has the following four properties:

(A) S^1_ι does not contain any child of $g^{\iota-1}$.

(B) $\{z^\iota_3, \ldots, z^\iota_m\}$ does not contain any child of $g^{\iota-1}$ and y is not a child of $g^{\nu-1}$.

(C) S^1_ι does not contain any parent of w^ι.

(D) For $1 \leq \iota \leq \mu - 1$, $\{z^\iota_3, \ldots, z^\iota_m\}$ does not contain any parent of $w^{\iota+1}$ and $\{z_2, \ldots, z_m\}$ does not contain any parent of w^1.

(R1) and *(R2)* imply *(A)* and (B), respectively; the way we defined the w^i in each iteration implies (C), and *(D)* holds because of *(R3).*

Properties *(A)* to *(D)* ensure that we can exchange the series $[g^0, \ldots, g^{\nu-1}]$ with the series $[w^1, \ldots, w^\nu]$ one by one, respectively, and obtain an optimal schedule for G which starts with $(w^1, x_2, \ldots, x_m)$ as dcsircd. ∎

References

[AH74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, 1974.

[CG72] E. G. Jr. Coffman, and R. L. Graham, *Optimal scheduling for two-processor8 systems,* Acta Informatica. 1(1972), pp. 200-213.

[G J78] M. R. Garey, and D. S. Johnson, *Computers and Intractability:* A *Guide to the Theory of NP-completeness,* Freeman, San Francisco 1978.

[G J80] M. R. Garey, D. S. Johnson, R. E. Tarjan and M. Yannakakir, *Scheduling Opposing Forests,* Technical Report, Bell Laboratories, To appear.

[GL79] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, *Optimization and apptozimation in deterministic sequencing and scheduling: a survey,* Bell Laboratories, TM-79121612, 1979.

[Go76] D. K. Goyal, *Scheduling processor bound systems,* Report No. CS-76-034, Computer Science Dept., Washington State University (1976).

[Hu61] N. C. Hu, *Parallel sequencing and assembly lint problems,* Operations Rcs. 9, 1961.

[Kn73] D. E. Knuth, *The Art of Computer Programing, volume 3: Sorting and* Searching, Addison-Wesley, Reading, *Mass.,* 1973.

[PY78] C. H. Papadimitriou, and M. Yannakakis, *Scheduling interval-ordered tasks,* Report No. TR-11-78, Center of Research in Computing Technology, Harvard University (1978).

[Ul75] J. D. Ullman, *UP-complete scheduling problems,* Journal of Computer and System Sciences. 10(1975), pp. 384-393.