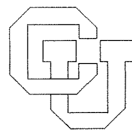
**An Almost-Linear Algorithm for Two-Processor Scheduling**

**Harold N. Gabow***

**CU-CS-196-80 October 1980**



**University of Colorado at Boulder**
**DEPARTMENT OF COMPUTER SCIENCE**

*This research was supported in part by National Science Foundation Grant MCS 78-18909
This is a revised version of CU-CS-169-80

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE ACKNOWLEDGMENTS SECTION.

# AN ALMOST-LINEAR ALGORITHM FOR TWO-PROCESSOR SCHEDULING†

by

Harold N. Gabow*
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado

CU-CS-196-80 October, 1980

Abstract

n unit-length jobs subject to precedence constraints must be scheduled on two processors in minimum finish time. Previous algorithms for this well-known problem begin by finding the transitive closure, and so use time $O(\min(en,n^{2.61}))$. An $O(e+n\alpha(n))$ algorithm is presented. The algorithm constructs a type of highest-level-first schedule, which is shown to be optimum.

* This research was supported in part by National Science Foundation Grant MCS 78-18909

†This is a revised version of CU-CS-169-80.

## 1. Introduction

In a well-known model for multiprocessor scheduling, n unit-length jobs are to be executed by m identical processors, a dag specifies precedence relations among the jobs, and a schedule with minimum finish time is sought [C]. For arbitrary m this problem is NP-complete [U]. For fixed $m \geq 3$ no polynomial-time algorithm is currently known. Here we investigate the tractable case, $m = 2$.

For this case several polynomial-time algorithms have been given. They all begin by finding the transitive closure of the dag. This uses time $O(\min(en, n^{2.61}))$, where e is the number of edges of the dag. (The first bound follows from doing n depth-first searches; the second follows from reducing transitive closure to matrix multiplication [AHU,P].)

The algorithm of Fujii, Kasami and Ninomiya [FKN] is based on matching techniques. Excluding transitive closure time, it requires the time to find a maximum matching, $O(n^{2.5})$ [K]. Coffman and Graham [CG] give an algorithm based on a lexicographic numbering scheme. Sethi [S] shows the numbering can be done in time $O(e+n\alpha(n))$.* (This algorithm can begin by finding the transitive closure or transitive reduction. However, both operations require the same time [AGU]). Garey and Johnson [GJ] give an algorithm for scheduling with precedence constraints and deadlines, which also solves our problem. It uses time $O(n^2)$ to compute "modified deadlines".

Time bounds for these algorithms often assume the transitive closure (or reduction) of the dag is given. In practice this is unlikely. On general dags the transitive closure step dominates, and the algorithms

---

*$\alpha(n)$ is an inverse of Ackermann's function and is very slow-growing. [T]

use time $O(\min(en, n^{2.61}))$. ($O(n^{2.61})$ for [FKN]).

Here we present an algorithm that does not use the transitive closure. The time on an arbitrary dag is almost linear, $O(e+n\alpha(n))$. The algorithm is a refinement of the highest-level-first scheduling rule; the schedule always executes nodes on the longest paths of the dag.

Section 2 gives definitions and motivation. Section 3 gives the algorithm. Section 4 briefly discusses extensions of the algorithm to more general scheduling models.

## 2. Preliminaries

This section gives the terminology for the problem. Also, it defines and motivates the basic idea of HLF schedules.

A scheduling problem is defined by a dag (directed acyclic graph). n and e denote the number of nodes and edges, respectively. If there is an edge from node x to node y, then x is an *immediate predecessor of y*, and we write *$x \rightarrow y$*; if there is a directed path (of 0 or more edges) from x to y, then x is a *predecessor* of y, y is a *successor* of x, and we write *$x \overset{*}{\rightarrow} y$*. A dag can be partitioned into *levels* i, $u \geq i \geq 1$: level i consists of all nodes x that start a path with i nodes, but not a path with i + 1 nodes. In this case *level(x)* = i. Figure 1 shows an example dag.

A *(2-processor) schedule executes (schedules)* nodes during *time units* i, $1 \leq i \leq \omega$, so that each node is executed in exactly 1 time unit, $\leq 2$ nodes are executed in the same time unit, and any predecessor of a node is executed in a lower time unit. $\omega$ is the schedule's *finish time*. We seek an *optimum schedule*, i.e., one with minimum finish time. Figure 2 shows a schedule for the example dag, as a *Gantt chart*, i.e., the $i^{th}$ column gives the nodes executed in time unit i. (The block structure, indicated by heavy lines, is explained in Section 3).

A *level* schedule "executes levels" in the order u,...,1 (where u is the highest level). More precisely, suppose levels u,...,i+1 have been executed, and level i contains s unexecuted nodes. Then level i is executed in the next $\lceil \frac{s}{2} \rceil$ time units: The first $\lfloor \frac{s}{2} \rfloor$ units each execute 2 nodes of i. If s is odd, the $\lceil \frac{s}{2} \rceil^{nd}$ unit executes the last node x of i, and possibly a node y of a lower level. The

schedule of Figure 2 is level.

If s is odd, then i is a <u>1-level</u>, and (x,y) is a <u>jump from x to y</u> (or <u>from level(x) to level(y)</u>). For uniformity, if node y does not exist (i.e., the $\lceil \frac{s}{2} \rceil^{nd}$ unit executes only x), y is taken to be a dummy node 0. 0 is on a fictitious level 0; as many jumps to 0 may be made as necessary. Finally, note because of jumps, s may be less than the number of nodes originally on level i.

An optimum schedule that is level always exists. This follows from the Coffman-Graham algorithm [CG]. Alternatively, one can transform any optimum schedule to be level. For example, suppose the highest level of a dag has nodes $x_i$, i=1, 2; further, $x_i$ is scheduled with $y_i$, a node not on the highest level, i=1, 2. Then $x_1$ and $x_2$ can be scheduled together. Nodes $y_1$, $y_2$, and others, can each be scheduled with a successor of $x_1$ (or $x_2$) on their level. The basic principle is that if level(x) > level(y) and $x \not\to y$ then x has a successor z on level(y), $z \neq y$. In this way we get an optimum schedule that begins by executing $x_1$ and $x_2$. Repeating this transformation gives an optimum, level schedule. Details of this proof are in [G1].

In an arbitrary level schedule, let the 1-levels be $f_1 > f_2 \ldots > f_k$, and let level $f_i$ jump to level $t_i$ ($t_i$ = 0 if no real node is jumped from level $f_i$.) Then $(t_1, t_2, \ldots, t_k)$ is the schedule's <u>jump sequence</u>. Note the levels $f_i$ can be deduced from the jump sequence and the original dag (f is a 1-level if the number of nodes on f, minus the number of occurrences of f in the jump sequence, is odd.) Note also the jump sequence determines $\omega$. (The number of 0's in the jump sequence is the number of time units that have an idle processor, i.e., execute only 1 node.)

A highest-level-first (HLF) schedule is a level schedule whose jump sequence is as large as possible; here we use lexicographic order to compare jump sequences. An HLF schedule always jumps to the highest level possible; when there is a choice of nodes to jump on that level, the node that allows subsequent jumps to be highest is jumped. Figure 2 gives an HLF schedule.

Any HLF schedule is optimum. This is proved in Section 3, in the analysis of the algorithm. However, to motivate the algorithm, we indicate here why HLF schedules are optimum.

First, we can prove this fact directly, by transforming an optimum level schedule to be HLF. This elaborates on the transformation to level schedules discussed above. Details are in [G1].

Second, we can make a simple plausibility argument. Suppose f is a 1-level, and node y can be jumped from f. Then y can be jumped from any level that is $\leq$ f and $>$ level(y). So if there is a choice of nodes y to jump from f, the one on the highest level should be jumped; this allows the greatest number of levels $<$ f to jump other nodes y. For example, if there are two nodes $y_1$, $y_2$, where level($y_1$) $>$ level($y_2$), then $y_1$ should be jumped from f. This way some 1-level g, g $>$ level($y_2$), can jump $y_2$. If $y_2$ were jumped from f, then when level($y_1$) $>$ g, g cannot jump $y_1$, so the resulting schedule may not be optimum.

This reasoning also shows that when there is a choice of nodes to jump on the highest level, the choice should be made so subsequent jumps can be highest. In other words, the schedule should be HLF.

## 3. The Algorithm

This section presents an algorithm that finds an optimum, HLF schedule, in time $O(e+n\alpha(n))$.

The main problem in constructing an HLF schedule arises when there is a choice of nodes to jump. For instance in Figure 1, level 3 can jump to nodes 4, 5, 7 or perhaps even 8. In general terms, suppose the highest level some 1-level can jump to is t. If a number of nodes on t can be jumped, which one should be chosen? From the HLF definition, the choice should be made to allow subsequent jumps to be highest.

So consider a subsequent jump, from a level f. We begin by assuming $f > t$. If f can jump to t, another node on level t must be chosen. If f can jump to a level $> t$, clearly it does not matter which nodes on t have been jumped. Finally, suppose f can only jump to a level $< t$. Since the schedule is HLF, any node on t that f might have jumped must have been jumped from a higher level. Clearly the choice of node to jump from such higher levels does not matter, since all choices must be jumped before f.

In summary, the choice of node to jump on t does not effect the jump from any level $f > t$. Further, the nodes on t divide into two types, called "non-free" and "free." The non-free nodes must be jumped. For instance in the above discussion, a node on t, jumped before a jump that goes below t, is non-free. The free nodes may be jumped, but they need not be; an alternate choice exists. In Figure 1 on level 2, node 8 is non-free and the others are free.

Now consider the jump from level t itself. The jump can be made from any free node x of level t. Thus the free node x that

allows the highest jump to be made should <u>not</u> be jumped. In Figure 1, node 4 should not be jumped, since it can jump to (node 2 of) level 1, while the other free nodes of level 2 cannot. (Note node 3 must be jumped before level 2). This single rule is the only one that governs the choice of nodes to jump!

Now we can describe a simple two-pass procedure to compute an HLF schedule. Pass I computes the jump from level f (if it exists), for f = u, ..., 1: It finds the highest level t that f can jump to. If level t has several nodes that can be jumped, it guesses one arbitrarily. Of course, the guess may be incorrect. However, Pass I keeps track of the non-free and free nodes. It always computes the best jump from a free node x of f. Pass II makes substitutions for the free nodes x that were incorrectly jumped.

This approach has a slight drawback in terms of efficiency. The difficulty is in finding the highest level t to jump to; t changes arbitrarily with successive jumps. If the priority queues of [E] are used to find t, an $O(e+n \log\log n)$ algorithm results.

For greater efficiency we restructure the computation. Pass I computes the jumps to level t, for t = u, ..., 1: For each node y on level t, it finds the highest level f that can jump to y. It guesses that f jumps to y. As above, Pass I keeps track of the non-free and free nodes, and always computes jumps from free nodes. Pass II fixes bad guesses.

This second approach has the advantage of a simpler "highest level" computation. The first approach computes the highest level t to jump to; a given t may be highest at various, arbitrary times. The second approach computes the highest level f to jump from;

a given f is highest only once. (After its jump has been found, level f is no longer a candidate.) This allows the use of set merging techniques, giving an $O(e+n\alpha(n))$ algorithm.

Now we give a detailed description of the algorithm, beginning with the data structures. The schedule is specified in arrays FROM and TO. For $u \geq f \geq 1$, (FROM(f), TO(f)) is the jump from level f. There are two special cases: if TO(f) = -1, there is no jump from f i.e., f is not a 1-level; if TO(f) = 0, node FROM(f) is scheduled with an idle processor. Clearly these arrays give enough information to deduce the entire schedule (in linear time), if desired.

Pass I guesses TO values; these guesses are modified in Pass II. For conceptual clarity, Pass I stores guesses in the array T; Pass II copies T to TO, and then modifies TO. The T array is of course unnecessary in an actual implementation.

Pass I uses two main data structure. First, it partitions levels into sets. When level t is being processed, a level $f > t$ is called <u>open</u> if the jumps to f make it a 1-level, but T(f) = 0 (i.e., no nontrivial jump has been found); also, level t itself is open. Each open level f has a set,

$OSET(f) = \{g \mid u \geq g \geq f$ and f is the highest open level $\leq g\}$.

OSETs are manipulated by operations FIND(g) (which returns f where $g \in OSET(f)$) and UNION(f,g) (which does a destructive merge of OSET(f) into OSET(g)) [AHU].

The second data structure helps assign jumps. When Pass I processes level t, it finds when each node y on level t is ready to be jumped, i.e., it computes

R(y) = the highest open level that can jump to y.

It also finds which nodes can be jumped from a given level, i.e., it computes, for each open level f, a list

$$RLIST(f) = \{y|R(y) = f\}.$$

(These interpretations of R and RLIST are valid immediately before line 6.)

Pass I also computes a "substitute jump" node on level t, SUB(t). This node is not jumped in Pass I; however, it is ready as early as possible (i.e., R(SUB(t)) is as large as possible). Any level $f \leq R(SUB(t))$ can jump SUB(t) instead of T(f). So Pass II can use SUB(t) to insure that FROM(t) is not jumped, thus fixing bad guesses. This motivates the following definition.

Definition 1 A node y on level t is *free* if $y = T(f)$ implies $f \leq R(SUB(t))$; i.e., either y is not jumped in Pass I, or y is jumped from R(SUB(t)) or below.

This definition is consistent with the earlier intuitive description of "free," i.e., a free node need not be jumped. Note, at least intuitively, any level has free nodes, since not every node of a level can be jumped. Also note the special case where $SUB(t) = 0$. Since the algorithm sets $R(0) = 0$, the only free nodes on such a level t are those that are not jumped.

The algorithm works as follows. Pass I processes levels t in decreasing order, $t = u,\ldots,1$. For each t, R and RLIST values are computed (lines 2-5). Then RLISTs are used to assign jumps, i.e., T-values (lines 6-9). The node with highest R-value that need not be jumped does not get jumped; instead it is made SUB(t) (line 10). The method for finding SUB(t) relies on merging RLISTs so nodes with higher R-values are at the end (line 9). After all levels t are processed, Pass II processes levels f in increasing order, $f = 1,\ldots,u$. For each f, a proper node FROM(f) is found. If FROM(f) happens to be jumped by Pass I, the jump is switched to go to SUB(f) instead of FROM(f).

Now we give the algorithm in pseudo-Algol

```
procedure H; comment Given a dag, H returns the jumps of an HLF
        schedule;

begin

Initialization:

0.    do a breadth-first search of the dag to define levels u,...,1;
         set SUB(t) = 0, T(t) = 0, OSET(t) = {t}, RLIST(t) = φ, for u ≥ t ≥ 1;
         set OSET(0) = {0}, R(0) = 0;
Pass I:

1.  for t ← u to 1 do begin

2.     for each node y on level t do begin

3.        r ← min {u+1, ℓ|an immediate predecessor x of y is "executed
             at level  ℓ," i.e., x = T(ℓ), or x is on level ℓ and is not
             a T-value};

4.        if r ≤ u, T(r) = 0, and some free node on level r does not
             immediately precede y comment the test for "free"  is in Definition 1;

             then R(y) ← r

             else R(y) ← FIND(r-1);

5.        add y to RLIST (R(y));

          end;

6.     while RLIST(f) ≠ φ for some f > t do begin

7.        remove the first node y from RLIST(f); T(f) ← y;

8.        g ← FIND(f-1); UNION(f,g);

9.        add RLIST(f) to the end of RLIST(g);

          end;

10.    if RLIST(t) ≠ φ then begin
          y ← the last node in RLIST(t); RLIST(t) ← φ;
          if R(y) > t then SUB(t) ← y;
          end;

11.    if level t is not a 1-level (i.e., the number of nodes that are
          not T values is even) then begin T(t) ← -1; UNION(t,t-1) end;

  end Pass I;
```

Pass II:

```
    let TO(g) = T(g) for 1 ≤ g ≤ u comment TO and T can be the same array;
12. for f ← 1 to u do begin
13.     if TO(f) ≥ 0 then begin
14.         let FROM(f) be a free node on level f, that does not immediately
               precede TO(f) if TO(f) > 0;
15.         if FROM(f) = TO(g) for some g then TO(g) ← SUB(f);
    end end end H.
```

Table I gives the values calculated by the algorithm for the dag of Figure 1. In Table I (a), a node y has an asterisk, $y^*$, if it is a SUB node, i.e., y = SUB(level(y)); y is in parentheses, (y), if it is non-free. Figure 2 shows the complete schedule computed by the algorithm.

The H schedule is the one computed by the algorithm, i.e., the level schedule with jumps (FROM(f), TO(f)), $u \geq f \geq 1$. We analyze the H schedule as follows: Lemmas 1-4 give the basic properties of Pass I; Lemma 5 shows how Pass II modifies the schedule, and Corollaries 1-4 give analogous properties of the H schedule. These properties include the fact that H is a valid schedule (Corollary 3), and H has an HLF-like property (Corollary 4). The latter is used to prove H is optimum (Lemmas 6-8). Finally, Lemma 10 shows that H is actually an HLF schedule.

The analysis treats 0 as a dummy node on a fictitious level 0. So for instance the assertion "level(T($\ell$)) > f" means T($\ell$) is a real node, above f.

To start, note the OSETs are maintained by lines 8 and 11, in accordance with their definition above.

The intuitive discussion at the start of Section 3 notes that a node z on t, jumped before a jump that goes below t, is non-free. So if z is free, no subsequent jump goes below t. We prove this property (for z = T(f)) as follows.

Lemma 1: Let f be a 1-level where T(f) is free; let $\ell$ be a 1-level where $f \geq \ell >$ level(T(f)). Then level (T($\ell$)) $\geq$ level(T(f)); if equality holds, T($\ell$) is free.

Proof: Let $t = level(T(f))$. Suppose in line 10, RLIST(t) contains a node y. Lines 5-9 show than no level $\ell$, $t < \ell \le R(y)$, is open.

In the hypotheses of the Lemma, $t < \ell \le f \le R(SUB(t))$. So the above remark shows $\ell$ is not open, i.e., $level(T(\ell)) \ge level(T(f))$. Definition 1 shows if equality holds, $T(\ell)$ is free. □

The next Lemma is used to show FROM nodes exist.

Lemma 2: Let f be a 1-level, $f \le R(y)$. Then f contains a free node x, $x \not\to y$.

Proof: Let r be the value computed in line 3 for y. So $f \le R(y) \le r$. If $f = r$, then $R(y) = r$, and the Lemma holds by line 4. Otherwise $f < r$. Since f is a 1-level, it contains a node x that is not a T-value. x is clearly free; $x \not\to y$ since $f < r$. □

The next Lemma is used to show H respects precedence.

Lemma 3: If $x \to y$ then either $level(x) \ge R(y)$ or $x = T(f)$ for some $f > R(y)$ (i.e., Pass I executes x at or before R(y)).

Proof: Line 3 sets r so any immediate predecessor x of y is executed at or before r, i.e., level $(x) \ge r$ or $x = T(f)$ for some $f \ge r$. If $R(y) < r$, this gives the Lemma. If $R(y) = r$, line 4 shows $f > r$; again the Lemma holds. □

The next Lemma essentially shows the HLF property for Pass I. To motivate its statement, let $\ell$ be a 1-level, and $z = T(\ell)$. The HLF property implies that no node y above level(z) can be jumped from $\ell$. Thus if $level(y) > level(z)$, either y must be scheduled before $\ell$, or all free nodes of $\ell$ precede y. This is essentially Lemma 4 (a).

Lemma 4 (b) shows a supporting fact: all non-free nodes must indeed be jumped, i.e., no free node can be substituted for a non-free node.

Lemma 4: Let $\ell$ be a 1-level. Let y be a node executed after $\ell$ by Pass I, i.e., $level(y) < \ell$, and $y \neq T(f)$ for any $f \geq \ell$. Let $z = T(\ell)$, and suppose either

(a) level(z) < level(y)

or (b) level (z) = level(y), y is free but z is not.

Then all free nodes of $\ell$ precede y.

Proof: Without loss of generality, we can assume y has no predecessors executed after $\ell$. So in line 3 for y, $r \geq \ell$. Both (a) and (b) imply level $\ell$ is open when R(y) is computed. So line 4 shows $R(y) \geq \ell$, unless $r = \ell$ and all free nodes of $\ell$ precede y. We show $R(y) \geq \ell$ gives a contradiction, thus proving the Lemma.

Since $y \neq T(f)$ for $f \geq \ell$, lines 5-9 with $R(y) \geq \ell$ show $\ell$ is assigned a jump. Thus level(z) = level(y), so (b) holds. Further, if y is free, lines 5-10 show $R(SUB(t)) \geq R(y)$; thus $R(SUB(t)) \geq \ell$, and z is free. This contradicts (b). □

Now we examine how Pass II changes TO.

Lemma 5: For any level g, TO(g) is either T(g) or SUB(level(T(g))); in the latter case, T(g) is free. In both cases $g \leq R(TO(g))$.

Proof: At the start of Pass II, any value TO(g) is T(g). Line 15 may change TO(g) to SUB(f) where f = level(T(g)). This is done only if T(g) is free (by line 14). Further TO(g) is not changed again, since the new value is still on level(T(g)).

Finally, note $g \leq R(T(g))$ (by lines 5-9), and if T(g) is free, $g \leq R(SUB(level(T(g))))$ (by Definition 1). □

Corollary 1: For any level g, level(T(g)) = level(TO(g)); T(g) is free iff TO(g) is free; if T(g) is non-free, T(g) = TO(g). □

The properties of H follow:

Corollary 2: The H schedule is well-defined. More precisely, any node FROM(f) exists and is not jumped (i.e., FROM(f) $\neq$ TO(g) for any g).

Proof: When lines 13-15 are executed for a level f, TO(f) has its final value. By Lemma 5, $f \leq R(TO(f))$. So in line 14, node FROM(f) exists, by Lemma 2. Line 15 insures FROM(f) is not jumped. □

Corollary 3: The H schedule is valid

Proof: We show H respects precedence constraints. This means if $x \rightarrow y$, then x is executed before y. Since H is a level schedule, this is clear if y is not jumped. So suppose y = TO(f).

From Lemma 5, $f \leq R(TO(f)) = R(y)$. So Lemma 3 shows either level(x) $\geq$ f or x = T(g) for some g > f.

Suppose level(x) $\geq$ f. Then x is executed before y, unless x = FROM(f). The latter is impossible by line 14.

So suppose level(x) < f and x = T(g) for g > f. It suffices to show x = TO(g). Assume the contrary, i.e., T(g) $\neq$ TO(g). So T(g) is free (Corollary 1). Now Lemma 1, applied to g, shows level(T(f)) $\geq$ level(T(g)), i.e., level(y) $\geq$ level(x), a contradiction. □

Finally, we show a version of the HLF property for H. (Lemma 10 below shows this version actually implies the HLF property).

Corollary 4: Consider the H schedule. Let y be a node scheduled after a 1-level $\ell$. Let z be the node jumped by $\ell$, where either

(a) level(z) < level(y)

or (b) level(z) = level(y), y is free but z is not.

Then all free nodes of $\ell$ precede y.

Proof: We need only verify the hypotheses of Lemma 4.

Since y is scheduled after $\ell$, level(y) < $\ell$ and y $\neq$ TO(f) for f $\geq \ell$. This implies y $\neq$ T(f) for f $\geq \ell$. For assume y = T(f). So Pass II changes TO(f), whence y is free. Lemma 1 (and Corollary 1) show level(z) $\geq$ level(y). Thus (b) holds. Now Lemma 1 (and Corollary 1) show z is free. This contradicts (b).

Finally, note (a) and (b) imply their counterparts in Lemma 4, by Corollary 1. □

Now the derivation parallels [CG]: We divide the schedule into blocks on which H is obviously optimum. Blocks $X_i$ are defined by boundary levels $\ell_i$:

Definition 2: The levels $\ell_i$, $1 \leq i \leq v + 1$, are defined as follows: $\ell_1 = 1$. For i > 1, $\ell_i$ is the lowest 1-level such that $\ell_i > \ell_{i-1}$ and either

(a) $\ell_i$ jumps below $\ell_{i-1}$, i.e., level $(TO(\ell_i)) < \ell_{i-1}$,

or (b) $\ell_i$ jumps to a non-free node on $\ell_{i-1}$, i.e.,

$$level(TO(\ell_i)) = \ell_{i-1} \text{ and } R(SUB(\ell_{i-1})) < \ell_i.$$

Let $\ell_v$ be the last value defined using the above criteria, and set $\ell_{v+1} = u + 1$.

For $1 \leq i \leq v$, block $X_i$ consists of all nodes scheduled after level $\ell_{i+1}$, up to and including $\ell_i$, except for the node jumped from $\ell_i$. Equivalently, $X_i = \{x | \ell_i \leq level(x) < \ell_{i+1}$, and x is not jumped from $\ell_{i+1}$ or above.$\}$ □

Figure 2 indicates the blocks for the example schedule. Now we show any schedule processes blocks in order.

Lemma 6: For $1 < i \leq v$ and any $x \in X_i$ on level $\ell_i$, $x \overset{*}{\rightarrow} X_{i-1}$.

Proof: First note that for any i, $1 \leq i \leq v$, any node $x \in X_i$ on level $\ell_i$ is free. For suppose x is non-free. Then x is jumped from some 1-level $\ell$. If x is on level $\ell_i$, Definition 2 implies $\ell \geq \ell_{i+1}$. Thus $x \notin X_i$.

Now to show the Lemma, take any $x \in X_i$ on level $\ell_i$, and any $y \in X_{i-1}$. x is free by the above remark; similarly if y is on $\ell_{i-1}$, it too is free. Now Corollary 4 shows $x \overset{*}{\rightarrow} y$. □

Lemma 7: For $1 < i \leq v$, $X_i \overset{*}{\rightarrow} X_{i-1}$.

Proof: Consider any $x \in X_i$. By Lemma 6, it suffices to show $x \overset{*}{\rightarrow} z$, for some node $z \in X_i$ on level $\ell_i$.

By Definition 2, level$(x) \geq \ell_i$. Clearly we can assume level$(x) > \ell_i$. So take z on level $\ell_i$ with $x \overset{*}{\rightarrow} z$. z must be executed after x, whence after level $\ell_{i+1}$. So $z \in X_i$, as desired. □

Finally the desired conclusion follows.

Lemma 8: The H schedule is optimum.

Proof: Let $\omega(X_i)$ ($\omega^*(X_i)$) denote the number of time units in the H schedule (optimum schedule) in which some node of block $X_i$ is executed. First note

$$(1) \quad \sum_{i=1}^{v} \omega^*(X_i) \geq \sum_{i=1}^{n} \omega(X_i).$$

For any 1-level $\ell$ of a block $X_i$, $\ell > \ell_i$, jumps a node of $X_i$. So every time unit of $\omega(X_i)$, except the last, executes 2 nodes of $X_i$. The last unit executes $\geq 1$ node of $X_i$. (It may execute 2 nodes, if $i = 1$). Thus $X_i$ has $\geq 2\omega(X_i) - 1$ nodes. This implies $\omega^*(X_i) \geq \omega(X_i)$, and (1) follows.

The length of the H schedule is the right-hand side of (1), by Definition 2. The length of the optimum schedule is at least the left-hand side (since Lemma 7 shows a time unit is counted in at most 1 term $\omega^*(X_i)$). So (1) implies H is optimum. □

Now we consider the timing of H. First we give some details needed for efficiency. For line 3, each node y has a list of immediate predecessors. Further, each node x indicates the level $\ell$ (if any) with $x = T(\ell)$. These data structures make the total time in line 3 $O(e+n)$.

For the test of line 4, each level r has a count of its free nodes. The count is set after level r is processed (line 11). It is easy to see this requires linear time. Line 4 compares this count with the number of free predecessors of y on level r. So the total time in the test of line 4 is $O(e+n)$.

For line 6, there is a list of levels f with RLIST(f) $\neq \phi$. For line 9, the RLISTs have end-pointers. It is easy to see the loop of lines 6-9, excluding set merging, uses a total of $O(n)$ time.

Lemma 9: H uses time $O(e+n\alpha(n))$ and space $O(e+n)$.

Proof: There are at most n FINDs in line 4, and also in line 8. Lines 8 and 11 do at most n UNIONs. So the time for set merging is $O(n\alpha(n))$. The remaining processing is $O(e+n)$, from the above discussion. □

Lemmas 8-9 give the final result.

Theorem 1: Procedure H finds an optimum schedule, in time $O(e+n\alpha(n))$, and space $O(e+n)$. □

As promised above, we conclude the discussion by showing the HLF property.

Lemma 10: H is an HLF schedule.

Proof: Let H have jump sequence $(t_1,\ldots,t_k)$. Let S be an arbitrary

schedule, with jump sequence $(s_1,\ldots,s_r)$. The Lemma requires $(t_1,\ldots,t_k) \geq (s_1,\ldots,s_r)$. We show, by induction on i, $1 \leq i \leq k$, that

(i) $(t_1,\ldots,t_i) \geq (s_1,\ldots,s_i)$;

(ii) if equality holds in (i), then for each of the first i jumps of S and H, S jumps a free node iff H does.

It is easy to see (i), with $i = k$, implies $k = r$. This gives the desired conclusion.

Supposing (i)-(ii) are true for i, we prove them for i+1, as follows. We may assume equality in (i). (Otherwise (i) holds with inequality for all indices $\geq i$). So the next 1-level is the same in both schedules, say $\ell$. Let the jump from $\ell$ be from node x in S, and to node z in H (so $z = TO(\ell)$). We show (i) and (ii) follow from Corollary 4.

First note that in S,

(1) all non-free nodes of $\ell$ are jumped from above $\ell$;

(2) all non-free nodes of level(z) are jumped from above $\ell$, if z is free.

For in H, (1) is obvious; (2) follows from Lemma 1 and Corollary 1. Now inductive assertion (ii) implies (1) and (2) for S.

Next note x is free. For x is not jumped in S, so (1) shows it is free.

Now we prove (i). Let y be any node with level(y) > level(z), and suppose H schedules y after $\ell$. x, a free node of $\ell$, precedes y, by Corollary 4. Thus y is scheduled after $\ell$ in S. The remaining nodes in level(y) are jumped from above $\ell$ in H. (i) (with equality) shows S jumps all of these nodes from above $\ell$. So in S, the jump from $\ell$ goes

goes to level(z) or lower. (i) follows.

For (ii), first suppose z is free. (2) implies S can only jump a free node of level(z), as desired.

Next suppose z is non-free. In H, all free nodes of level(z) are scheduled after $\ell$ (by Lemma 1 and Corollary 1). Thus x, a free node of $\ell$, precedes all free nodes of level(z), by Corollary 4. So S can only jump a non-free node of level(z). (ii) follows. □

This completes the formal justification for the HLF idea:

Theorem 2: Any HLF schedule is optimum.

Proof: All HLF schedules have the same jump sequence, hence the same length. So Lemma 10 gives the Theorem. □

## 4. Conclusions

We have shown that for two-processor systems, HLF schedules are optimum and easy to construct. It is natural to ask how these schedules fare on various extensions of the model.

For example, consider the case of $m > 2$ processors. If the dag is a forest, Hu's algorithm [H] finds an optimum schedule for arbitrary m; further, the schedule is HLF. Unfortunately, some dags admit no optimum, level schedule when $m > 2$. In fact, there are dags where <u>any</u> level schedule is a factor $2 - \frac{2}{m}$ greater than optimum [LS]. Among level schedules, however, the HLF strategy is best: $2 - \frac{2}{m}$ is an upper bound on the accuracy, and the time to find an HLF schedule is almost linear.

Other extensions of the basic model include tasks with arbitrary integer lengths, uniform processors (i.e., processors whose speeds differ by a constant factor) and scheduling with resources other than

processors. In each case the results are similar: the HLF strategy achieves the best possible accuracy bound for a level schedule, and the time is $O(e+n\alpha(n))$ or $O(e+n \log\log n)$. These results are presented in detail in [G2]. These problems and others illustrate the usefulness of the highest-level scheduling method.

## Acknowledgments

The author thanks the anonymous referees for helpful advice on clarifying the presentation, and Dr. Michael Garey for valuable editorial assistance.

References


[AGU] Aho, A. V., Garey, M. R. and Ullman, J. D., "The transitive reduction of a directed graph," SIAM J. Comput. 1, 1972, pp. 131-137.

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

[C] Coffman, E. G. Jr., Ed., Computer and Job-Shop Scheduling Theory, Wiley & Sons, New York, 1976.

[CG] Coffman, E. G. Jr. and Graham, R. L., "Optimal scheduling for two-processor systems," Acta Informatica 1, 3, 1972, pp. 200-213.

[E] van Emde Boas, P., Kaas, R., and Zijlstra, E., "Design and implementation of an efficient priority queue," Math. Systems Theory 10, 1977, pp. 99-127.

[FKN] Fujii, M., Kasami, T., and Ninomiya, K., "Optimal sequencing of two equivalent processors," SIAM J. Appl. Math. 17, 4, 1969, pp. 784-789. Erratum, SIAM J. Appl. Math. 20, 1971, p. 141.

[G1] Gabow, H. N., "An almost-linear algorithm for two-processor scheduling," CU-CS-169-80, Dept. of Comp. Sci., Univ. of Colorado, Boulder, Colo., Jan. 1980.

[G2] Gabow, H. N., "Highest-level-first algorithms for approximate scheduling," in preparation.

[GJ] Garey, M. R. and Johnson, D. S., "Scheduling tasks with nonuniform deadlines on two processors," J.ACM 23, 3, 1976, pp. 461-467.

[H] Hu, T. C., "Parallel sequencing and assembly line problems," Op. Res. 9, 6, 1961, pp. 841-848.

[K] Kariv, O. "An $O(n^{2.5})$ algorithm for finding a maximum matching in a general graph," Ph.D. Diss., Weizmann Inst. Science, Rehovot, Israel, 1976.

[Kn] Knuth, D. E., The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley, Reading, Mass., 1973.

[LS] Lam, S. and Sethi, R., "Worst case analysis of two scheduling algorithms," SIAM J. Comput. 6, 1977, pp. 518-536.

[P] Pan, V. Y., "Field extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplications," Proc. 20th Annual Symp. of Found. of Comp. Sci., 1979, pp. 28-38.

[S] Sethi, R., "Scheduling graphs on two processors," SIAM J. Comput. 5, 1, 1976, pp. 73-82.

[T] Tarjan, R. E., "Efficiency of a good but not linear set union algorithm," J.ACM 22, 2 (April 1975), pp. 215-225.

[U] Ullman, J. D., "NP-complete scheduling problems," J. Comput. System Sci. 10, 1975, pp. 384-393.

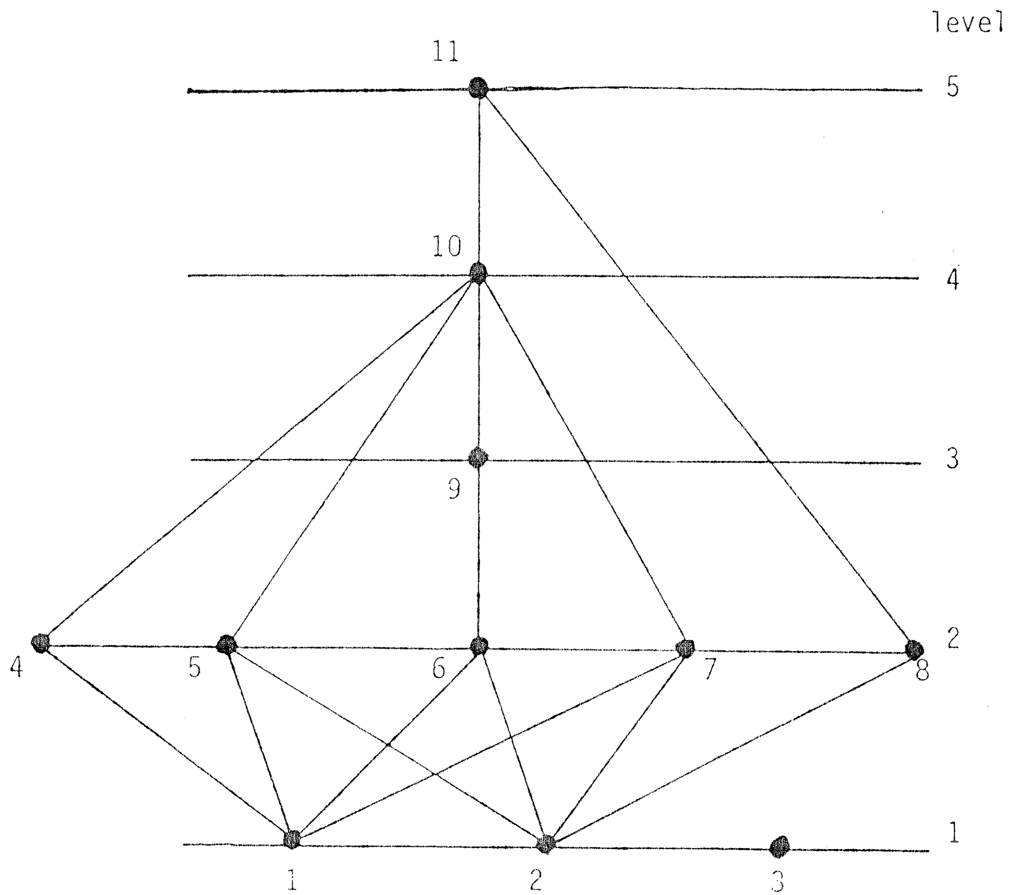



Figure 1

Example dag; all edges are directed downward.

| y | 11 | 10 | 9 | (8) | 7* | 6 | 5 | 4 | (3) | (2) | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R(y) | 6 | 4 | 3 | 4 | 3 | 2 | 3 | 3 | 6 | 2 | 1 |

| f | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|
| T(f) | 3 | 8 | 4 | 2 | 0 |

(a)

| f | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|
| FROM(f) | 11 | 10 | 9 | 4 | 1 |
| TO(f) | 3 | 8 | 7 | 2 | 0 |

(b)

Table I

(a) Pass I values. ( ) = non-free node; * = SUB.

(b) Pass II values.

| 11 | 10 | 9 | 6 | 4 | 1 |
|---|---|---|---|---|---|
| 3 | 8 | 7 | 5 | 2 | 0 |

Figure 2
Complete schedule, with blocks.