EXACT AND APPROXIMATE ALGORITHMS FOR SCHEDULING UET SYSTEMS ON TWO UNIFORM PROCESSORS

by

Harold N. Gabow

Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

CU-CS-225-82 July 12, 1982

This research was supported, in part, by
NSF grant MCS 78-18909.

ABSTRACT

A set of jobs related by precedence constraints is to be executed in minimum time on two processors, one requiring f time units per job and the other requiring $s \geq f$ time units. When $s = f + 1$ the desired schedule can be characterized as HLA, "highest-level-first with abstentions". It can be found in linear time when there is a bounded amount of idle; in general the time is exponential in the number of levels of the precedence graph. Approximately optimum schedules can be found in linear or nearly linear time. The schedule that is optimum for two identical processors has accuracy at most a factor $2 - \frac{f}{s}$ above optimum, for arbitrary f and s. The HLF "highest-level-first" algorithm has accuracy $\frac{5}{4}$ for $\frac{f}{s} = \frac{1}{2}$ and $\frac{6}{5}$ for $\frac{f}{s} = \frac{2}{3}$.

KEY WORDS

Scheduling, uniform processors, highest-level-first schedule, precedence constraints, directed graph.

INTRODUCTION

Much work has been done on the problem of finding a shortest multiprocessor schedule for a set of unit execution time jobs subject to precedence constraints (so-called *UET systems*) [C]. The most encouraging results are for two identical processors. This problem is $P2 \mid prec, p_j = 1 \mid Cmax$ in the notation of [GLLRK]. A number efficient algorithms have been given [FKN, CG, GJ1], including one with linear running time [G, GT]. The problem can be efficiently solved even in the presence of release times and deadlines [GJ1, GJ2]. On the other hand the problem is NP-complete when the two processors are identical but the job execution times are either one or two ($P2 \mid prec, p_j \in \{1, 2\} \mid Cmax$). [U].

This paper investigates the problem for two uniform processors. ($Q2 \mid prec, p_j = 1 \mid Cmax$. A *uniform* processor runs at constant speed.) Suppose a job can be executed in f time units on one processor and s time units on the other, $f \leq s$. The case $s = f + 1$ is most amenable to analysis. We show that the highest-level-first (HLF) characterization of an optimum schedule for two identical processors [G] generalizes to this case. Here the optimum schedule is HLA, "highest-level-first with abstentions". Such a schedule differs from HLF in that it contains idle time that, from a local viewpoint, is not forced. An HLA schedule can be constructed in linear time if the location of the idle time is known. This leads to an algorithm that finds the shortest schedule by trying all possibilities for the idle time. The algorithm runs in $O(2^L (n + m))$ time, where n and m are the number of nodes and edges of the precedence graph, respectively, and L is the number of levels. In particular the algorithm is linear if either L or the amount of idle time in the optimum schedule is bounded.

To handle general speeds f and s and also to obtain polynomial running times, we study algorithms that find a schedule that is guaranteed to be nearly optimum. One strategy is to use the optimum schedule for two identical processors. This gives a uniform processor schedule that is at most a factor $2 - \frac{f}{s}$ above optimum. As expected this factor approaches one as $\frac{f}{s}$ approaches one. Another strategy is to use an HLF schedule for the two uniform processors. (Such a schedule is defined only when $s = f + 1$). This gives better accuracy for two important cases: When $\frac{f}{s} = \frac{1}{2}$ the accuracy is $\frac{5}{4}$ and when $\frac{f}{s} = \frac{2}{3}$ the accuracy is $\frac{6}{5}$. (All bounds on accuracies are tight.)

Section 2 gives definitions and reviews the relevant results for two identical processors. Section 3 derives the HLA characterization of an optimum schedule. This gives an alternate proof that HLF schedules are optimum for two identical processors. Section 4 explores the use of the identical processor schedule to approximate the optimum uniform processor schedule. Section 5 explores the HLF (highest-level-first) approximation algorithm. Section 6 gives concluding remarks. Appendix A gives details of the construction of HLF schedules.

2. PRELIMINARIES

This section gives basic terminology and reviews some results in scheduling.

A scheduling problem is defined on a *dag* (directed acyclic graph). A node of the dag represents a job that requires one unit of processing time. An edge represents a precedence constraint. n and m 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 node with no predecessors is *initial*. A dag can be partitioned into *levels* i, $L \geq i \geq 1$: level i consists of all nodes x that start paths with i nodes but not paths with $i + 1$ nodes. We write $level(x) = i$. L denotes the highest level of the dag. Figure 2.1 shows a dag. (Throughout this paper dags are drawn with the convention that edges are directed from the higher node to the lower node.)

The nodes of the dag (jobs) are executed on two *uniform* processors, i.e., processors of constant speed. The speeds are given as positive integers f and s, where $f \leq s$. The *fast processor* P_f executes one node in f time units, while the *slow processor* P_s executes one node in s time units. (Strictly speaking parameters f and s are the inverses of speed. This should not cause confusion.) Without loss of generality f and s are relatively prime. Define $\Delta = s - f$. If $\Delta = 0$ the processors are *identical*.

Suppose that processors P_f and P_s start simultaneously and execute nodes without interruption. In fs time units P_f executes s nodes and P_s executes f nodes. These $f + s$ nodes are executed in a pattern that repeats, as illustrated in Figure 2.2. This pattern is the *execution period*. A time interval on P_f or P_s in the execution period, during which one node is executed, is a *slot*. The slots of a period are numbered from 1 to $f + s$, in order of increasing right endpoint. By

convention the last two slots are $f+s-1$ on P_s and $f+s$ on P_f.

For identical processors the execution period is trivial. Assume the processors are not identical, i.e., $f < s$. This implies that a slot on P_f overlaps at most two slots on P_s; also a slot on P_s overlaps at least two slots on P_f. Suppose a slot i on P_s overlaps more than two slots on P_f. Then i *subsumes* any slot on P_f that is entirely contained in i's time interval. In Figure 2.1(b) slot five subsumes slot four.

Exactly $\Delta-1$ slots on P_f are subsumed. For $f-1$ slots on P_f overlap the borders between slots on P_s. Aside from slots 1 and $f+s$ the remaining $s-f-1 = \Delta-1$ slots on P_f are subsumed.

When $\Delta = 1$ no nodes are subsumed and every slot overlaps at most two others. This is the fundamental reason why the results of Section 3 hold. The following properties of the slot numbering scheme are used in Section 3 (see Figure 2.2(a)).

Lemma 2.1. Assume that $\Delta = 1$.

(a) The slots are numbered from 1 to $2f+1$, with odd-numbered slots on P_f and even-numbered slots on P_{f+1}.

(b) The left endpoint of a slot is nondecreasing with slot number, as is the right endpoint.

Proof. (a) If slot $2i-1$ is on P_f, slot $2i$ is on P_{f+1} since no slot is subsumed. Slot $2i+1$ is on P_f since $f < f+1$.

(b) The right endpoint is nondecreasing by definition. The left endpoint is nondecreasing by an induction similar to (a). ∎

This property is used in Section 4. (See Figure 2.2(b).)

Lemma 2.2. Assume that $\Delta > 1$. For $1 \le i < \Delta$ the ith subsumed node on P_f is

the $\lfloor i\frac{f}{\Delta}\rfloor+i$th node scheduled on P_f. The ith subsuming node on P_s is the $\lfloor i\frac{f}{\Delta}\rfloor$th node scheduled on P_s.

Proof. The ith subsumed node ends at the first time P_f has executed $i+1$ more nodes than P_s. So if $j+i$ nodes have been executed on P_f, $(j+i)f < js$. This implies $j = \lfloor i\ \frac{f}{\Delta}\rfloor$ and the Lemma follows. ∎

A schedule is a specification of how each node is processed such that all precedence constraints are obeyed. More precisely a *schedule* is an assignment of each node x to an ordered pair $(p(x), t(x))$, where $p(x) \in \{f, s\}$ is the processor that *executes* x, and $t(x) \geq 0$ is the *start time* for x, and for any distinct nodes x and y, $t(x) + p(x) \leq t(y)$ if either $p(x) = p(y)$[1] or $x \to y$. $t(x) + p(x)$ is the *finish time* for node x, and processor $P_{p(x)}$ executes x from $t(x)$ to $t(x) + p(x)$. A processor is *idle* whenever it is not executing any node. ω, the largest finish time of a node, is the *length* or *makespan* of the schedule. An *optimum schedule* has length ω^*, the minimum length possible.

The following definitions generalize highest-level-first schedules. These schedules are optimum for identical processors ($\Delta = 0$) [G]. The definitions are given for $\Delta = 1$, where we shall show they lead to optimum schedules. However they are also relevant for arbitrary Δ.

An "i-schedule", for $1 \leq i \leq f + s$, starts in slot i. More precisely let slot i be on processor P. ($i' = i + 1$, except that when $i = f + s$, $i' = 2$). In an *i-schedule*, $P(P')$ starts at slot $i(i')$ or later. The *length* of an i-schedule is measured starting at the beginning of slot i. So an ordinary schedule is a 1-schedule.

A level schedule "executes levels" in the order L, $L-1$, ..., 1. (Recall that L is the highest level.) More precisely let i be a slot, $1 \leq i \leq 2f + 1$. Let H be the

[1] If the processors are identical, the values f and s are assumed to designate a processor and so are distinct in the equation $p(x) = p(y)$.

set of all level L nodes. A *level i-schedule* executes the nodes of H in the first $|H|$ slots $i, i+1, \ldots, j$, where $j = (|H| + i - 2) mod (2f + 1) + 1$. There are three cases for j:

(i) $j = 2f + 1$ is the last slot of a period.

(ii) $j < 2f + 1$ and there is no node in slot $j + 1$.

(iii) $j < 2f + 1$ and there is a node y in slot $j + 1$.

In cases (i) - (ii) the rest of the schedule is a level 1-schedule for $G - H$. In case (iii) the rest of the schedule is a level j'-schedule for $G - H - y$, where $j' = j + 2$ if $j < 2f$ and $j' = 1$ if $j = 2f$ (see Figure 2.3). A *level-schedule* is a level 1-schedule. Figure 2.4 shows two level schedules.

In a level schedule consider a level l, $L \geq l \geq 1$, where case (iii) applies. Let x be the node in slot j. So $level(x) = l$, $level(y) < l$, and $x \overset{*}{\rightarrow} y$ since slots j and $j + 1$ overlap. The ordered pair (x, y) is a *jump from x to y*. Alternatively the jump goes *from level l to level (y)*, or *level l jumps y*. If case (ii) applies to level l, l has an *idle jump*. *Level l has a jump* in cases (ii) and (iii) but not case (i). In Figure 2.4 (and in all other figures of this paper) nodes that are jumped are dotted. The idle times in Figure 2.4 are idle jumps.

The *execution of level l*, for $L \geq l \geq 1$, is defined inductively. The execution of level L consists of slots $i, i + 1, \ldots, j$ plus slot $j + 1$ if L jumps a node. The execution of level l, $L > l \geq 1$, is defined by induction. Observe that nodes of l that are jumped are not included in the execution of l.

In a level schedule let the levels with jumps be $f_1 > \cdots > f_k$, where level f_i jumps to level t_i. By convention $t_i = 0$ for an idle jump. The *jump sequence* of the schedule is the ordered k-tuple $(t_1, t_2, \ldots, t_k)$. In Figure 2.4 the jump sequences are (1, 0, 0, 0) in (a) and (0, 1) in (b). Observe that the dag and the jump sequence together determine the slots where nodes are executed. In par-

ticular the length ω can be deduced.

Jump sequences are compared using lexicographic order. Hence $(t_1, \ldots, t_k) > (s_1, \ldots, s_r)$ if for some j, $1 \le j \le \min(k, r)$, $t_i = s_i$ for $1 \le i < j$ and $t_j > s_j$. (Lexicographic order allows the possibility that $(t_1, \ldots, t_k) > (s_1, \ldots, s_r)$ if $t_i = s_i$ for $1 \le i \le r$ and $k > r$. This cannot occur with jump sequences: If $t_i = s_i$ for $1 \le i \le r$ then $k = r$ and $(t_1, \ldots, t_k) = (s_1, \ldots, s_r)$.)

A highest-level-first (HLF) schedule is a level schedule whose jump sequence is as large as possible. HLF schedules are optimum for several types of identical processor scheduling [H, G]. Figure 2.4 shows they are not optimum for uniform processors: (a) is HLF but (b) is optimum. Observe that the optimum schedule (b) jumps to the highest level *except* when it has idle jumps. This is characteristic of an "HLA" schedule, defined as follows.

In a level schedule, a level is *abstentious* if it ends in slot $j < 2f$ and has an idle jump. A level that has a jump but is not abstentious is *nonabstentious*. So a nonabstentious level wither ends in slot $j < 2f$ and jumps a node or ends in slot $2f$. (When the level ends in slot $2f$, the schedule includes a complete slot $2f + 1$, whether or not the level jumps a node.)

Let A be a set of levels, $A \subset \{1, \ldots, L\}$, and let i be a slot, $1 \le i \le 2f + 1$. An *HLA i-schedule* is a level i-schedule with abstentious levels A, whose jump sequence is as large as possible. An *HLA (highest-level-first with abstentions) schedule* is an HLA 1-schedule. Figure 2.4 shows HLA schedules, with abstentions $\{1, 2\}$ in (a) and $\{4\}$ in (b).

For identical processors an HLF schedule is an HLA schedule with no abstentions. (Any idle jump is in the slot that is the analog of $2f$, the slot that never abstains.) Such a schedule can be constructed in linear time.

Similar algorithms can be used for uniform processors. In one instance we shall see the set of abstentious levels A is known in advance. (This is a nontrivial

assumption. A given set of levels A may not even have an HLA schedule, since the precedence constraints may force idle jumps at levels not in A).

Theorem 2.1. For processor speeds f, $f + 1$ and a given set of abstentious levels A, an HLA schedule, if it exists, can be found in $O(n + m)$ time and space.

Proof. The algorithm is a straightforward adaptation of the identical processor algorithm of [G]. This algorithm works in two passes. Pass I processes the levels t in descending order $t = L, L-1, \cdots 1$. It finds all levels that jump to level t. Pass II finds the specific nodes jumped.

The same strategy words for uniform processors. The important point is that after level t is processed in Pass I it is easy to decide whether or not t jumps a node. For the number of nodes on level t that are jumped is known. This determines the slot that executes the last node of t. Level t jumps a node if it does not end in slot $2f + 1$ and it is not in A. Both of these conditions are easy to test.

The time and space are $O(n + m)$ [G, GT]. ∎

The key fact in analyzing HLF schedules is that they decompose into blocks. The blocks are defined from a set of boundary levels l_i, $1 \le i \le B + 1$, where $l_1 = 1$, $l_{B+1} = L + 1$. The exact definition of l_i can be found in [G]. For $1 \le i \le B$, *block* X_i consists of all nodes scheduled after l_{i+1} up to and including l_i, except for the node (if any) jumped from l_i. Every node is in a block, except for the nodes jumped from boundary levels l_i. The blocks partition the time units of the HLF schedule.

Theorem 2.2. Assume that $\Delta = 0$.

(a) In an HLF schedule any block X is executed in $\lceil \frac{|X|}{2} \rceil$ time units.

(b) For any block X_i, $1 < i \leq B$, $X_i \overset{*}{\rightarrow} X_{i-1}$, i.e., any node of X_i precedes any node of X_{i-1}.

Proof. See [G]. ∎

Theorem 2.2 shows that an HLF schedule is optimum, since its length is $\sum_{i=1}^{B} \lceil \frac{|X_i|}{2} \rceil$. HLA schedules have a similar block structure, but it is not needed in this paper (although a block structure is used in Section 5).

3. THE EXACT ALGORITHM

This section proves that when $\Delta = 1$ there is an optimum HLA schedule. This leads to an algorithm that finds an optimum schedule in $O(n + m)$ time when there is no idle and in $O(2^L(n + m))$ time in general.

The proof is organized in a series of Lemmas. Lemmas 3.1-3 show that an optimum schedule that is level always exists. Lemma 3.4-6 establish some "highest-level-first" types of properties. Theorem 3.1 shows that an optimum schedule that is HLA always exists.

The first result will allow us to consider only schedules that execute nodes in slots.

Lemma 3.1 For any dag G and any slot i, $1 \leq i \leq 2f + 1$, there is an optimum i-schedule that executes a node in the first slot.

Proof. Let S be an optimum i-schedule. S can execute nodes at arbitrary times, not necessarily in slots. Consider two cases. First suppose that S executes a portion of some node x in slot i (i.e., S executes x on the same processor as i and begins x before the end of i). Since no node can be completed before the end of i (even if $i = 2f$), x is initial in G. So S with x moved to slot i is a valid. As an i-schedule it is no longer than S. Hence it is the desired optimum schedule.

Next suppose that S does not use any portion of slot i. Let x be the first node to start executing in S. Clearly x is initial. Further, x ends no earlier than slot i does (even when $i = 2f$). So S, with x moved to slot i, is a valid i-schedule no longer than S. Hence it is the desired schedule. ∎

Now we give two convenient definitions. If S is a schedule then S/x denotes the schedule obtained by removing x and all nodes scheduled before it from S. For instance if x is the first node of S, S/x is a schedule for $G-x$. If i

is the slot after x's then S/x is an i-schedule.

Let S be a schedule that executes node x first. If S/x is a level i-schedule for $G-x$ (and any i) then S is an *almost-level* schedule.

The next Lemma is the heart of the transformation of an optimum schedule to a level schedule. Recall that L is the number of levels in the dag.

Lemma 3.2. If a dag G has an optimum i-schedule that is almost-level then it has an optimum i-schedule with a level L node in the first slot i.

Proof. The proof is by induction on L. The base case $L = 1$ is trivial. Assume the Lemma holds for graphs with less than L levels. Prove it for graphs with L levels as follows.

Consider an optimum i-schedule S that is almost-level. Without loss of generality, assume that x, the node in the first slot, has $level(x) < L$. Let y be the first node scheduled after x (see Figure 3.1). Since S is almost-level $level(y) = L$. Let S' be schedule S with nodes x and y interchanged. If S' is a valid schedule the Lemma is proved. So assume S' is not valid, i.e., some precedence constraint is violated. S' does not violate a constraint involving y, since y is initial in G and also y ends no later in S' than it does in S. This implies that S' violates a constraint $x \overset{*}{\rightarrow} z$, where node z is scheduled in the slot overlapping y.

Since $x \overset{*}{\rightarrow} z$, level$(z) < L-1$. So $G - \{x, y\}$ is an $L-1$ level graph and S/y is an almost-level schedule for it. By induction $G - \{x, y\}$ has an optimum schedule T whose first node w is on level $L-1$ and is in the first slot of S/y. Since $level(x) < L$, $x \overset{*}{\not\rightarrow} w$. So a valid schedule for G results from scheduling y and x as in S', followed by T (see Figure 3.1). This schedule starts with a level L node as desired. ∎

Now we show that an optimum schedule can be transformed to a level schedule.

Lemma 3.3. For any dag G and any slot i, $1 \leq i \leq 2f + 1$, there is a level i-schedule that is optimum.

Proof. The proof is by induction on the number of nodes in G. The base case of no nodes is vacuous. Assume the Lemma holds for any graph with fewer nodes than G. Prove it for G as follows.

Observe that G has an optimum i-schedule S that starts with a level L node in slot i (see Figure 3.2). For there is an optimum i-schedule with some node x in slot i, by Lemma 3.1. Without loss of generality assume $level(x) < L$. By induction $G-x$ has a level $(i + 1)$-schedule that is optimum. This schedule, preceded by x in slot i, is an almost-level i-schedule for G that is optimum. (The schedule is valid because x does not precede the node in slot $i + 1$, since $level(x) < L$). Now Lemma 3.2 shows the desired schedule S exists.

Let x be the first node of S. Consider two cases, depending on whether or not x is the only node of level L.

Suppose x is not the only node of L. By induction $G-x$ has a level $(i + 1)$-schedule that is optimum. This schedule, preceded by x in slot i, gives a level i-schedule for G that is optimum (x does not precede the node in slot $i + 1$ since both nodes are on level L.)

Next suppose x is the only node of L. If in S the processor opposite x is idle during the entire duration of slot i, then x is followed by a 1-schedule for $G-x$. By induction this schedule can be assumed level, as desired.

Otherwise the processor opposite x executes a portion of some node y during slot i. Clearly y can be assumed to be executed on slot $i + 1$. By induction $G-\{x, y\}$ has a level $(i + 2)$-schedule that is optimum. This schedule, preceded

by x in slot i and y in slot $i+1$, is the desired schedule. (This schedule is valid because y does not precede the node in slot $i+2$, since the latter is on level $L-1$ and $level(y) \leq L-1$.) ■

The next two Lemmas show that initial nodes have "highest-level-first" properties.

Lemma 3.4. Let x by an initial node of a dag G. Then for any slot $i, 1 \leq i \leq 2f+1$, there is an optimum level i-schedule where all nodes scheduled before x are on $level(x)$ or higher.

Proof. Let S by an optimum level i-schedule where x is in the earliest slot possible. Suppose a node y with $level(y) < level(x)$ is scheduled before x. We derive a contradiction by showing that x can be scheduled earlier.

Choose y as the last node before x with $level(y) < level(x)$. Node y is jumped in S since x is scheduled at $level(x)$ or before.

Let T be the result of interchanging nodes x and y in schedule S (see Figure 3.3). We will show that T is a valid schedule unless in S, $level(x)$ jumps from x to a successor of y. T can only violate a precedence constraint involving x or y. Node x is initial and is scheduled earlier in T than in S. So T satisfies all constraints involving x. Hence a violated constraint has the form $y \overset{*}{\rightarrow} z$ for some node z with $level(z) < level(y) < level(x)$. In S, node z is not scheduled before x, by the choice of y. z can be scheduled in the slot after x's slot only if $level(x)$ jumps from x to z. (Otherwise if $level(x)$ does not jump *from* x, the slot after x contains a node on $level(x)$ or higher.) So T has the desired property.

Now consider two cases. First suppose T is a valid schedule. T is a level schedule up to and including node x, which is jumped. Let x be in slot j in T. Let U be an optimum level$(j+1)$-schedule for the nodes in T/x (U exists by

Lemma 2.3). Let V be the schedule T with T/x replaced by U. V is a valid schedule. (Node x does not precede the node u in slot $j+1$, since $level(u) \geq level(x)$.) V is a level schedule. Since V schedules x earlier than S it gives the desired contradiction.

Next suppose T is not valid, i.e., $level(x)$ jumps from x to a successor z of y. Let y be in slot k in T. (So z is in slot $k+1$.) Let U be an optimum level$(k+1)$-schedule for the nodes in T/y. Let V be the schedule T with T/y replaced by U. V is a valid schedule. (Node y does not precede the node u in slot $k+1$, since $level(u) = level(x)-1 \geq level(y)$.) If V is level, it gives the desired contradiction.

If V is not level then $k = 1$. (In this case, $level(x)$ has a jump in S since it ends in slot 1; it does not have a jump in T since it ends in slot $2f+1$.) Let w be the node in the slot before y in V. Let W be V with V/w replaced by an optimum level 1-schedule. W gives the desired contradiction. ∎

Lemma 3.5. Let X and Y be dags where node x is initial in X, node y is initial in Y, and $X-x = Y-y$. Suppose $level(x) > level(y)$. Then for any slot i, $1 \leq i \leq 2f+1$, an optimum i-schedule for Y is no longer than one for X.

Proof. Using Lemma 3.4, let S be an optimum level i-schedule for X such that all nodes before x are on $level(x)$ or higher. Let T be the schedule S with node x replaced by node y (see Figure 3.4). If T is a valid schedule for Y the Lemma is true. So suppose T is not valid. Since y is initial in Y, T can only violate a precedence constraint $y \overset{*}{\rightarrow} z$ for some node z with $level(z) < level(y) < level(x)$. The choice of S shows that z is not in a slot before x. So in T, z is not before y, whence it is in the slot after y. Call this slot j. It is easy to see that in S, $level(x)$ jumps to z. So $T/y = S/x$ is a j-schedule for a dag with $level(x)-1$ levels.

Let U be an optimum level j-schedule for T/y. Let V be T with T/y replaced by U. V is a valid schedule for Y. (Node y does not precede the node in slot j since the latter is on $level(x)-1$.) Since V has the same length as S it implies the Lemma. ∎

The next Lemma investigates the situation where there is a choice of nodes to jump on the same level. Property (2) is illustrated in Figure 3.5.

Lemma 3.6. Let X and Y be dags where node x is initial in X, node y is initial in Y, and $X-x = Y-y$. Suppose $level(x) = level(y) = l$. Let i be a slot, $1 \le i \le 2f + 1$, and let A be a set of abstaining levels. Let S be an HLA i-schedule for A on dag X. Then there is a similar schedule T on Y with these properties:

(1) S and T have the same jump sequence before l;

(2) Let X' be the subgraph of X that remains after the execution of level l(in S), and similarly for Y'. Either $X' = Y'$ or l jumps a node y' in S and x' in T, such that $X'-x' = Y'-y'$.

Proof For succinctness in this argument the term "good schedule" stands for "HLA i-schedule for A".

Property (1) asserts that any two good schedules for X and Y have the same jump sequence before l. We will show that if S is a good schedule for X, its jump sequence before l is at most that of a good schedule for Y. By symmetry Y has a similar property and (1) follows.

So let S be a good schedule for X. Let T be S with node x replaced by y. We first show that T is a valid schedule unless level l jumps from x to a successor of y.

T can only violate a precedence constraint $y \overset{*}{\rightarrow} z$ for a node z with

$level(z) < level(y) = l$. Suppose z is scheduled before y (in T). So z is before x in S. But since S is HLA and x is initial, this implies $level(z) \geq level(x) = l$, a contradiction. So z is scheduled in the slot after x. This means level l jumps from x to z, as desired.

Before the jump from l, T is a valid level i-schedule with abstentions A. (Note T is level since $level(x) = level(y)$.) So the jump sequence before l of a good schedule for Y is at least that of T. This is the same as that of S (again recall $level(x) = level(y)$). Property (1) now follows.

To show property (2) consider two cases. First suppose that in S no level above l jumps below l. Let T be any good schedule for Y. By (1) T has the property supposed for S. Hence X' consists of the nodes of X below l minus y', the node (if any) jumped from l. Similarly Y' consists of the nodes of Y below l minus x', the node (if any) jumped from l. So as in (2), either $X' = Y'$ or $X'-x' = Y'-y'$.

The second case is when in S some level $f > l$ jumps below l. By (1) f jumps below l in any good schedule for Y. Since S is HLA, x is jumped before f. Similarly y is jumped before f in any good schedule for Y. Now the argument for property (1) shows that a good schedule has the same jump sequence in X as in Y, and that S, with x replaced by y, is a good schedule for Y. Choosing this schedule for T makes $X' = Y'$, so (2) holds. ∎

Now we show the main result.

Theorem 3.1. Assume that $\Delta = 1$. For any dag G and any starting slot $i, 1 \leq i \leq 2f + 1$, there is a set of levels A such that any HLA schedule (for A) is optimum.

Proof. The proof is by induction on the number of levels L. If $L = 1$ the Theorem is trivial. Assume the Theorem holds for dags with fewer than L levels.

We prove it for dags with L levels as follows.

Let S be an optimum level i-schedule. Let z be the last node scheduled in the execution of level L. By induction assume that S/z is an HLA schedule. Furthermore choose S so that subject to the above conditions, its jump sequence is as high as possible. Let A be the abstentious levels for S, i.e., $A = \{|l|$ level l ends in slot j, $1 \leq j < 2f$, but l does not jump a node, i.e., the slot after j is slot $1\}$.

We will show that S is an HLA schedule (for A). Clearly this implies that any HLA schedule for A is optimum.

Suppose that S does not jump from level L (i.e., either $L \in A$ or L ends at slot $2f + 1$). The choice of S implies it is HLA for A. The Theorem follows. So in the rest of the argument assume that S does a jump from L.

Now we argue inductively. We find an HLA schedule T for A, and levels l_j, $L = l_1 > l_2 > \cdots > l_k = 0$, such that the following two properties hold (see Figure 3.6):

(1) For $1 \leq j \leq k$, S and T have the save jump sequence above l_j.

(2) For $1 \leq j < k$, S jumps a node y_j from l_j, T jumps a node x_j from l_j, and $X_j - x_j = Y_j - y_j$, where X_j is the induced subgraph (of G) of all nodes scheduled after the execution of l_j in S, and similarly Y_j is the subgraph of nodes scheduled after l_j in T.

Property (1) for $j = k$ gives the desired conclusion.

Start by letting T be an arbitrary HLA schedule for A. For the base case $l_1 = L$, property (1) is vacuous. For (2) recall that S jumps a node y_1 from L. So $X_1 - x_1$ is the subgraph of all nodes below level L except for x_1 and y_1. The same description holds for $Y_1 - y_1$. (2) follows.

For the inductive step assume (1)-(2) for level l_j, where $j < k$. We find a level l_{j+1} satisfying (1)-(2) as follows. From (2), l_j jumps node y_j in S and x_j in

T.

Since T is an HLA schedule for A, $level(x_j) \geq level(y_j)$. In fact equality holds. For otherwise $level(x_j) > level(y_j)$. Let h be the slot after x_j (or equivalently after y_j). Let U be schedule T, with T/x_j replaced by an optimum, HLA h-schedule (for Y_j. Such a schedule exists by induction, although it may not abstain at the levels in A.) Since $level(x_j) < l_j$, U is a valid level i-schedule. Further Lemma 2.5 implies U is optimum for G. But U shows that S does not have the highest jump sequence possible. This contradiction shows $level(x_j) = level(y_j)$ as desired.

Now Lemma 3.6 can be applied. It shows that S and T have the same jump sequence above $level(x_j)$. Further T can be chosen so that (2) of Lemma 2.6 holds. Using the notation of Lemma 2.6, either $X' = Y'$ or $X'-x' = Y'-y'$.

If $X' = Y'$, clearly S and T are identical after $level(x_j)$. So take $l_{j+1} = 0$ and $k = j + 1$ to complete the induction.

If $X'-x' = Y'-y'$, take $l_{j+1} = level(x_j)$ and k will be a value $> j + 1$. So (1) and (2) hold for l_{j+1} (for (2), let $y_{j+1} = y'$ and $x_{j+1} = x'$, so $X_{j+1} = X'$ and $Y_{j+1} = Y'$).

This completes the induction. The Theorem follows. ∎

The key to finding the optimum HLA schedule is in deducing the abstentious level. We have not succeeded in doing this. Instead we give two simple applications of the Theorem.

Corollary 3.1. Assume that $\Delta = 1$ and the given dag has a schedule with no idle time. The optimum schedule can be found in $O(n + m)$ time and space.

Proof. Apply Theorem 3.1 and Theorem 2.1 with $A = \phi$. ∎

Corollary 3.2. Assume that $\Delta = 1$. An optimum schedule can be found in $O(2^L(n + m))$ time and $O(n + m)$ space (where L is the number of levels in the

dag).

Proof. There are 2^L possible sets of abstentious levels A. The algorithm finds an HLA schedule for each set, using Theorem 2.1, and selects the shortest. ∎

The proof of Theorem 3.1 is easily adapted to show that an HLF schedule is optimum for identical processors. This derivation is independent from the one in [G].

Corollary 3.3. For two identical processors any HLF schedule is optimum.

Proof. For identical processors an execution period is one time unit long with slots numbered one and two. A node that is jumped goes in slot two. The proofs of the Lemmas and the Theorem are simpler because slot two does not overlap the following slot. In Lemma 3.6 and Theorem 3.1 the set of abstentions A is ϕ. ∎

The HLA characterization holds only for processor speeds with $\Delta = 1$. When $\Delta > 1$ some dags do not even admit an optimum level schedule.

Theorem 3.2. Assume that $\Delta = s - f > 1$. There is a dag where no level schedule is optimum.

Proof. The argument depends on the relative sizes of $2f$ and s. The case of equality does not occur since it implies $f = 1$, $s = 2$ and $\Delta = 1$. The other two cases follow.

Case I. $2f > s$.

Consider the dag of Figure 3.7. It consists of two components S and F. S is a chain of $s-1$ nodes. F has two nodes a, a' that precede a chain of $f-1$ nodes.

An optimum schedule has length fs, with no idle time. P_f executes node a followed by the nodes of S. P_s executes node a' followed by the remaining nodes of F. This schedule is not level since a and a' are on level f whereas the

highest level is $s-1 > f$.

To prove the Theorem it suffices to show that this optimum schedule is unique. First consider a dag consisting of two chains. In a schedule with no idle (if one exists), each processor executes nodes from one chain for fs time units. At this point the processors can switch chains, but not before.

Now consider a schedule with no idle time for Figure 3.7. There are three cases, depending on the nodes executed at time 0. If a and a' are executed at time 0 the preceding observation implies that the schedule is the optimum schedule already given. If P_s executes node b at time 0 the observation implies that P_f executes the entire F component and P_s executes the S component. But F has only $f + 1 < s$ nodes, leading to idle time on P_f.

The last case is where P_f executes b at time 0. So P_s executes a at time 0. If P_f executes a' after b, $2f > s$ implies that P_s executes the rest of S while P_f executes the rest of F. This leads to idle time, as in the previous case. If P_f does not execute a' after b, $2f > s$ implies that P_s executes the entire F component while P_f executes the S component. But since S has only $s-1$ nodes this leads to idle time on P_f.

Case II. $2f < s$.

Consider the dag of Figure 3.8. An optimum schedule has length $\omega^* = \min(4f, f + s)$: First P_f executes node a while P_s is idle. Then P_s executes node b while P_f executes the remaining nodes. Clearly this schedule is not level.

The optimum schedule is essentially unique. If P_s executes no nodes then $\omega = 5f > \omega^*$. If P_s executes two nodes then $\omega \geq 2s > \omega^*$. If P_s executes one node, observe that any node besides b makes $\omega \geq s + 2f > \omega^*$. Otherwise P_s

executes b and the schedule is the optimum one described or a minor variant of it. ■

4. THE AI APPROXIMATION ALGORITHM

This section investigates the strategy of using an optimum identical processor schedule for two uniform processors. The worst-case accuracy bound is $2 - \frac{f}{s}$. As expected this approaches one as $\frac{f}{s}$ approaches one. The time and space to find the schedule are both $O(n + m)$.

An *AI (Approximately Identical) schedule* T is constructed as follows. First find an HLF schedule S for two identical processors. Then convert it to T: A time unit of S where one node is executed corresponds to f time units of T where the same node is executed on P_f. A time unit of S where two nodes are executed corresponds to s time units of T where the same two nodes are executed. P_s executes one node while P_f executes the other node with Δ units of idle.

Clearly an AI schedule can be found in $O(n + m)$ time. To derive the accuracy bound define these quantities from the AI schedule:

t = the number of slots on P_f scheduled with a node on P_s;

u = the number of slots on P_f scheduled with idle on P_s.

Clearly $\omega = st + fu$. The next two Lemmas derive bounds on ω^*.

Lemma 4.1. $(\frac{1}{f} + \frac{1}{s})\omega^* \geq 2t + u$.

Proof. The optimum schedule executes at most $(\frac{1}{f} + \frac{1}{s})\omega^*$ nodes. There are $2t + u$ nodes, so the Lemma follows. ∎

Lemma 4.2. $\omega^* \geq tf + uf$.

Proof. The Lemma is proved by decomposing the schedule into blocks X_j, $j = 1, \ldots, B$ (recall Theorem 2.2). Define the following quantities for a block

X.

$\omega^*(X)$ = the length of an optimum schedule for the nodes of X;

$t(X)$ = the contribution to t made by X in the AI schedule;

$u(X)$ = the contribution to u made by X in the AI schedule.

Observe that $\omega^* \geq \sum_{j=1}^{B} \omega^*(X_j)$ by Theorem 2.2(b). Also $t + u = \sum_{j=1}^{B} t(X_j) + u(X_j)$. So it suffices to show $\omega^*(X) \geq t(X)f + u(X)f$ for every block X.

Theorem 2.2(a) implies that in the AI schedule P_f executes exactly $\lceil \frac{|X|}{2} \rceil$ nodes of X. So $\lceil \frac{|X|}{2} \rceil = t(X) + u(X)$. Clearly $\omega^*(X) \geq \lceil \frac{|X|}{2} \rceil f$, since a schedule executes at least $\lceil \frac{|X|}{2} \rceil$ nodes on some processor. This gives the desired conclusion. ∎

Theorem 4.1 On any dag the AI schedule achieves the bound $\frac{\omega}{\omega^*} \leq 2 - \frac{f}{s}$.

Proof: Multiply the inequality of Lemma 4.1 by $s - f$, the inequality of Lemma 4.2 by $2 - \frac{s}{f}$, and add. This gives $(2 - \frac{f}{s})\omega^* \geq st + fu$. Since the right-hand side is ω the bound follows. ∎

The bound of the Theorem is tight.

Theorem 4.2. There is a dag where the AI schedule has $\frac{\omega}{\omega^*} = 2 - \frac{f}{s}$.

Proof. Consider the dag of Figure 4.1. It consists of a chain of s nodes plus f isolated nodes. The AI schedule executes a chain node with an isolated node until all isolated nodes are exhausted. Then it executes chain nodes with idle. So $\omega = fs + \Delta f$.

The optimum schedule executes the chain nodes on P_f and the isolated nodes on P_s, so $\omega^* = fs$. Thus $\frac{\omega}{\omega^*} = 1 + \frac{\Delta}{s} = 2 - \frac{f}{s}$. ∎

One way to improve the AI schedule is to compress idle on P_f. When two consecutive nodes on P_f are on the same level, the second node can be scheduled after the first with no intervening idle. Similarly when a node is jumped on P_s, the node following it on P_f can be scheduled right after its predecessor. The *compressed AI (CAI) schedule* is the result of applying these rules as much as possible to remove idle. Observe that if $\lceil \frac{f}{\Delta} \rceil$ nodes are compressed a whole new slot is available on P_f. We will not concern ourselves with how this slot is used since we are only interested in lower bounds and the desired bound does not depend on it.

The CAI schedule has this alternate description. It is a level schedule that never jumps from processor P_s but otherwise has the highest jump sequence possible. (Again we do not specify the schedule at slots that are subsumed and at slot $f + s$.)

Figure 4.1 illustrates how compression improves the AI schedule. For instance when $\Delta = 1$ the CAI schedule is optimum. However we now show that compression does not change the worst-case performance.

The lower bound dag has the overall structure shown in Figure 4.2. The basic component is a "module". There are m modules, each a copy of a fixed dag. The modules are arranged in series, i.e., every node of one module precedes every node of the next. (Figure 4.2 indicates this by heavy arrows between modules.) The last module is followed by a chain of c nodes. Every node of the last module precedes every node of the chain. Finally there are i isolated nodes.

We start with the simple case $\Delta = 1$.

Lemma 4.3. Assume that $\Delta = 1$ and $f \geq 4$. For any $\epsilon > 0$ there is a dag where the CAI schedule has $\frac{\omega}{\omega^*} > 2 - \frac{f}{s} - \epsilon$.

Proof. Consider the dag of Figure 4.2 with the module of Figure 4.3. The module consists of two chains of f nodes each plus an isolated node. Notice that ignoring the isolated nodes, each level of the dag has one, two or three nodes. Choose any positive integer r and let the parameters be $m = i = rf$, $c = rs$.

Figure 4.4 shows the CAI schedule. (Solid nodes are on levels with one, two or three nodes and are labelled accordingly. Dotted nodes are the jumped, isolated nodes). Each module is executed as follows: Levels with two nodes end with idle time on P_f. Levels with three nodes jump an isolated node. The following node (which is the first node of the next module or the first node of the chain) gets compressed. Note the restriction $f \geq 4$ implies that the last slot $f + s$ is not reached, so there is no problem about the definition of the CAI schedule.

The modules jump $m = i$ isolated nodes. So each chain node has an idle jump on P_s. Thus $\omega = m(f+1)s + (c+2)f - 2s = m(f+1)s + rfs - 2$.

Figure 4.5 shows the optimum schedule. Each module is executed as follows: P_s executes one chain while P_f executes the other chain plus the isolated node. Clearly this fills one execution period with no idle time.

In the rest of the schedule P_f executes the chain nodes while P_s executes the isolated nodes. Since $c = rs$ and $i = rf$ there is no idle time. Thus $\omega^* = mfs + rfs$.

It is easy to see that $\frac{\omega}{\omega^*} = 1 + \frac{ms-2}{fs(m+r)} = 1 + \frac{1-\frac{2}{rfs}}{s}$. This approaches $1 + \frac{1}{s} = 2 - \frac{f}{s}$ as r approaches ∞, as desired. ∎

Now we consider the general case.

Theorem 4.3. Assume that $\frac{f}{s} > \frac{4}{s}$ or $f = 4$, $s = 5$. For any $\epsilon > 0$ there a dag where the CAI schedule has $\frac{\omega}{\omega^*} > 2 - \frac{f}{s} - \epsilon$.

Proof. By Lemma 4.3 assume $\Delta > 1$. Thus there are subsumed nodes. Observe that the hypothesis implies $\frac{f}{\Delta} > 4$.

Consider the dag of Figure 4.2. Choose any positive integer r and let the parameters be $m = rf$, $c = rs\Delta$ and $i = rf(2\Delta - 1)$.

Use the module of Figure 4.6. To describe this module define integers a_j, $i \le j \le \Delta$, as follows:

$$a_j = \lceil j\frac{f}{\Delta} \rceil - \lceil (j-1)\frac{f}{\Delta} \rceil - 2, \quad 1 \le j < \Delta;$$

$$a_\Delta = f - \lceil (\Delta - 1)\frac{f}{\Delta} \rceil - 1$$

Note that all $a_j \ge 1$ since $a_j \ge j\frac{f}{\Delta} - (j-1)\frac{f}{\Delta} - 3 > 4 - 3 = 1$. A module has Δ sections. For $j > 1$ the jth section consists of a level of one node followed by a_j levels of two nodes, followed by a level of three nodes. For $j = 1$ the level of one node is omitted. A node on a level with one or three nodes precedes all nodes below it in the module (Figure 4.6 indicates this by heavy arrows.) A node on a level with two nodes immediately precedes only one node on the next level.

Figure 4.7 shows the CAI schedule. Each module is executed as follows: Levels with two nodes end with idle time on P_f. Levels with one or three nodes jump an isolated node. The node on P_f after a jumped node gets compressed. So for $j > 1$ the jth section of a module begins with four consecutive slots on each of P_f and P_s. Since $\frac{f}{\Delta} > 4$ none of these slots is subsumed or subsuming. So there is no problem about the definition of the CAI schedule.

The modules jump $(2\Delta - 1)m = i$ isolated nodes. So each chain node has an idle jump. Thus $\omega = ms(\sum_{j=1}^{\Delta}(aj + 3) - 1) + cf - 2s + 2f = ms(f + \Delta) + cf - 2s + 2f$.

Figure 4.8 shows an optimum schedule. Each module is scheduled in one execution period as follows: Levels with two nodes jump the next level. Levels with three nodes are scheduled in the three slots preceding a subsumed slot (and also in the last three slots of the period). Levels with one node are scheduled in a subsumed slot. $\Delta-1$ isolated nodes are jumped in the slots on P_s that subsume a node.

The numbers a_j are defined so that this schedule for a module is valid: In an execution period, for $1 \le j < \Delta$ the number of slots on P_f strictly between the $(j-1)$st and jth subsumed slots is $\lfloor j\frac{f}{\Delta}\rfloor + j - (\lfloor (j-1)\frac{f}{\Delta}\rfloor + (j-1)) - 1 = a_j + 2$. This is also the number of nodes scheduled on P_f between the $(j-1)$st and jth levels with one node. Similarly the number of slots after the last ($(\Delta-1)$st) subsumed slot is $s - (\lfloor \Delta-1)\frac{f}{\Delta}\rfloor + \Delta-1) = a_\Delta + 2$. This is the number of nodes scheduled after the last level with one node. Analogous remarks hold for P_s, where the jth subsuming node is the $\lfloor j\frac{f}{\Delta}\rfloor$th node and $a_j + 1$ nodes are scheduled between levels with one node. These observations also show that the schedule obeys all precedence constraints.

The m modules jump $m(\Delta-1)$ isolated nodes, leaving $rf\Delta$ isolated nodes. These are scheduled with the chain nodes: $c = rs\Delta$ nodes of the chain are executed on P_f while the $rf\Delta$ isolated nodes are executed on P_s.

This schedule shows that $\omega^* = mfs + rs\Delta f$. Thus $\frac{\omega}{\omega^*} = 1 + \frac{cf - 2s + 2f}{fs(m + r\Delta)} = 1 + \frac{\Delta + \frac{(2f - 2s)}{rfs}}{s}$. As r approaches ∞ this approaches $1 + \frac{\Delta}{s} = 2 - \frac{f}{s}$. ∎

Theorem 4.3. shows that compression does not improve the AI schedule in the case of interest, where $\frac{f}{s}$ is close to one. We close this section by noting two other lower bounds on the CAI schedule's accuracy. For the purpose of comparison recall $2 - \frac{f}{s} = 1 + \frac{\Delta}{s}$.

(i) For any speeds f, s there is a dag where the CAI schedule has $\frac{\omega}{\omega^*} = \frac{s+f}{2f} = 1 + \frac{\Delta}{2f}$.

(ii) Assume that $\frac{f}{s} > \frac{3}{4}$. For any $\epsilon > 0$ there is a dag where the CAI schedule has $\frac{\omega}{\omega^*} = \frac{\lfloor \frac{s}{\Delta} \rfloor s + s}{\lfloor \frac{s}{\Delta} \rfloor f + s} - \epsilon = 1 + \frac{\lfloor \frac{s}{\Delta} \rfloor \Delta}{\lfloor \frac{s}{\Delta} \rfloor f + s} - \epsilon$.

(i) is proved by considering a dag that is two chains of $f + s$ nodes each. (ii) is proved by considering the dag of Figure 4.2-3 where the module of Figure 4.3 has height $\lfloor \frac{f}{\Delta} \rfloor$. Further details are left as an exercise.

5. THE HLF APPROXIMATION ALGORITHM

This section investigates the HLF scheduling rule as an approximation algorithm. When $\Delta = 1$ and HLF schedule can be found in $O(m + n \log \log n)$ time and $O(n + m)$ space. Two important cases of HLF scheduling are analyzed: When $\frac{s}{f} = \frac{1}{2}$ the worst-case accuracy bound is $\frac{5}{4}$. When $\frac{s}{f} = \frac{2}{3}$ the bound is $\frac{6}{5}$. Unless $\frac{s}{f} = \frac{3}{4}$, the AI schedule is more accurate than HLF.

The HLF scheduling algorithm differs from the HLA algorithm because the slots are not known in advance: The exact slots that comprise the execution of a level t depend on the number of idle jumps in levels above t. So the jumps from levels $l > t$ must be computed before level t's jump. This rules out the approach of the HLA algorithm.

Nonetheless a similar algorithm works for HLF scheduling. Pass I processes levels l in the order $l = L, L-1, \ldots, 1$. It finds the level (if any) that l jumps to. Pass II finds the specific jumps. A detailed presentation of the algorithm is in Appendix A.

Theorem 5.1. Assume that $\Delta = 1$. An HLF schedule can be found in $O(m+n \log \log n)$ time and $O(m+n)$ space.

Proof. See Appendix A. ■

HLF schedules decompose into blocks, as in the identical processor case described in Section 2.

Theorem 5.2. Assume that $\Delta = 1$.

(a) In an HLF schedule any block X is executed in consecutive time slots $i, i+1, \ldots, j$ with no intervening idle time. Here $1 \le i, j \le 2f+1$ and $i \ne 2$.

(b) For any block X_i, $1 < i \leq B$, $X_i \overset{*}{\rightarrow} X_{i-1}$.

Proof. In part (a), $i \neq 2$ since a level schedule never jumps a node in slot 1. The rest of the proof follows the identical processor case and is in Appendix A. ∎

The analysis of HLF schedules is done on blocks. Define the following quantities for any block X:

$\omega(X)$ = the length of the HLF schedule for X;

$\omega^*(X)$ = the length of an optimum schedule for X;

$\Delta(X) = \omega(X) - \omega^*(X)$.

If i is the first slot of X in the HLF schedule then it gives an i-schedule for X. $\omega(X)$ is defined as the length of this i-schedule, i.e., time 0 is at the start of slot i. So the length of the HLF schedule is $\omega = \sum_{j=1}^{B} \omega(X_j)$. $\omega^*(X)$ is defined as the length of an optimum 1-schedule for X. This allows both processors to start simultaneously, unlike $\omega(X)$. The length of an optimum schedule is $\omega^* \geq \sum_{j=1}^{B} \omega^*(X_j)$, by Theorem 5.2(b).

Now we analyze the HLF schedule for $\frac{f}{s} = \frac{1}{2}$. There are six possible shapes for a block, depending on the first and last slot. These are shown in Figure 5.1. Only one shape is nonoptimum.

Lemma 5.1. In a nonoptimum block X the first slot is three, the last slot is two and $\Delta(X) = 1$.

Proof: Let X be an arbitrary block with $3p + q$ nodes, $0 \leq q < 3$. Clearly $\omega^*(X) \geq 2p + q$. In Figure 5.1 all shapes for X have $\omega(X) = 2p + q$ except when the first slot is three and the last is two. In this case $\omega(X) = 2p + 3$ and $\omega^*(X) \geq 2p + 2$. ∎

To analyze the accuracy of the HLF schedule let i be the number of nonoptimum blocks. The bound follows from two inequalities.

Lemma 5.2. (a) $\omega^* \geq \omega - i$,

(b) $\frac{3}{2}\omega^* \geq \omega + i$.

Proof. (a) The number of time units the schedule can decrease is at most the number of nonoptimum blocks, by Lemma 5.1.

(b) Observe that there are at least $\omega + i$ nodes: From Figure 5.1 it is clear that each of the ω time units of the schedule can be associated with a distinct node in some block. In addition there are i nodes not in any block: Every nonoptimum block starts in slot three. The node jumped in slot two is not in any block.

(b) follows since the optimum schedule executes at most $\frac{3}{2}\omega^*$ nodes. ∎

Theorem 5.3. Assume that $\frac{f}{s} = \frac{1}{2}$. On any dag the HLF schedule achieves the bound $\frac{\omega}{\omega^*} \leq \frac{5}{4}$.

Proof. Add the two inequalities of Lemma 5.2. ∎

Now we show that the bound of the Theorem is tight.

Theorem 5.4. Assume that $\frac{f}{s} = \frac{1}{2}$. For any $\epsilon > 0$ there is a dag where the HLF schedule has $\frac{\omega}{\omega^*} > \frac{5}{4} - \epsilon$.

Proof: For any positive integer r consider the dag of Figure 5.2. It consists of r repetitions of a module with six nodes. (Node six is on level one but is drawn higher for clarity. Observe the similarity with Figure 2.1.)

The HLF schedule is shown in Figure 5.3. Each module is executed in five time units, so $\omega = 5r$. A better schedule is shown in Figure 5.4. Each module schedules node one of the next module. This shows $\omega^* \leq 4r + 1$. Thus $\frac{\omega}{\omega^*} = \frac{5r}{4r+1}$ which approaches $\frac{5}{4}$ as r approaches ∞. ■

We turn our attention to the HLF schedule for $\frac{f}{s} = \frac{2}{3}$. It has a different nature from previous approximate schedules. Regarding the lower bound dags, recall that in the previous algorithms the optimum schedule has $\Theta(1)$ idle time. (Indeed this is the case in all other examples for level-type scheduling algorithms [L, LS, G].) We have not found such examples for this algorithm--the dag that achieves the tight lower bound has $\Theta(\omega^*)$ idle time.

This property manifests itself in the proof of the upper bound. The previous schedules are analyzed by inequalities based on the fact that the number of nodes gives a lower bound on ω^*. Inequalities of this type cannot be tight in dags with $\Theta(\omega^*)$ idle time in the optimum schedule. Hence a different approach must be used. Our approach is based on an estimate, derived from the precedence constraints, of how much ω can decrease.

We first investigate how much a block can shrink. For generality consider two processors of speeds f and $s = f + 1$, and a block X in some HLF schedule. Define this quantity from the HLF schedule:

$T(X) = \frac{i}{f} + \frac{j}{s}$, where $i(j)$ is the number of time units in $\omega(X)$ when P_f (P_s) is not executing a node of X.

In the HLF schedule for X both processors are busy at all time except possibly during the first slot (if one processor is executing the node jumped before X) and during the last slot (if one processor is idle or is executing the node jumped from X). The quantity $T(X)$ is the amount of extra work that could have been

done in these two slots.

Lemma 5.3. $\Delta(X) \leq T(X)/(\frac{1}{f}+\frac{1}{s})$.

Proof. If n is the number of nodes in X then $(\frac{1}{f}+\frac{1}{s})\omega(X) = n + T(X)$ and $(\frac{1}{f}+\frac{1}{s})\omega^*(X) \geq n$. These inequalities imply the Lemma. ∎

Table I gives the contribution of each slot to $T(X)$ when it is the first or the last slot of X. For instance if the first slot of X is three then during the initial time unit of this slot P_3 is idle, contributing $\frac{1}{3}$ to $T(X)$. (If X consists of only one slot, its contribution to $T(X)$ is the sum of the first and last contributions.)

Lemma 5.4. Let X be a block.

(a) $\Delta(X) \leq 2$.

(b) If $\Delta(X) = 2$ then slot five is first in X and slot four is last.

(c) If $\Delta(X) > 0$ then slot three, four or five is first in X.

Proof. Table I shows that $T(X) \leq \frac{5}{3}$, with equality only when slot five is first and four is last. Applying Lemma 5.3 gives (a) and (b). For (c) note that any block with slot one first is optimum and slot two is never first. ∎

To obtain a global view of how the schedule can shrink, merge blocks into "segments". These are the portions of the schedule between idle jumps: A *segment* W consists of consecutive blocks, $W = \bigcup_{s \geq i \geq t} X_i$, where X_{s-1} and X_t, but no X_i, $s \geq i > t$, end with an idle jump. (At the boundaries, allow $s = B$ in the first segment and $t = 1$ in the last segment.)

Clearly the segments partition the blocks. Each segment starts in slot one of a period. Both processors run uninterrupted, jumping a node between con-

secutive blocks of the segment, until the last slot of the segment (where an idle jump may be made). To analyze the length of the schedule for a segment define these quantities:

$\omega(W)$ = the length of the HLF schedule for segment W;

$\omega^*(W)$ = the length of the optimum schedule for W.

Note that W does not contain the nodes jumped from the last level of each of its blocks. So it is possible that $\omega^*(W) < \omega(W)$.

Lemma 5.5 For any segment W, $\omega^*(W) \geq \frac{5}{6}\omega(W)$.

Proof. Let $W = \bigcup_{s \geq i \geq t} X_i$. Set $\Delta(W) = \sum_{s \geq i \geq t} \Delta(X_i)$. Write $\omega(W) = 6p + q$, $0 \leq q < 6$, so p is the number of complete execution periods in the segment. It suffices to show that $\Delta(W) \leq p$ since this implies $\omega^*(W) = \omega(W) + \Delta(W) \geq 5p + q$. This is done by assigning each unit of $\Delta(W)$ to a distinct, complete execution period of the segment, as follows.

Consider a block X in W with $\Delta(X) > 0$. By Lemma 5.4(a) $\Delta(X)$ is 1 or 2. If $\Delta(X) = 1$ this unit of $\Delta(W)$ is assigned to the period that X starts in. If $\Delta(X) = 2$ these two units of $\Delta(W)$ are assigned to the periods that X starts in and ends in.

Note that units of $\Delta(W)$ are assigned to complete execution periods. For a period is complete if it contains a node in slot four. Lemma 5.4(b)-(c) imply this is the case for all periods assigned to. It remains only to show that a period is assigned at most one unit of $\Delta(W)$.

Observe that two blocks X, Y with $\Delta(X)$, $\Delta(Y) > 0$ start in different execution periods. For suppose otherwise. Lemma 5.4(c) shows that X and Y both start in slots three, four or five. Since X and Y are separated by at least a jump node, the two starting slots are three and five. Hence one of the blocks, say X, consists of exactly one node in slot three. But this implies $\Delta(X) = 0$, a

contradiction.

Next observe that if $\Delta(X) = 2$ and X ends in period P, then X starts before P and no other block has slots in P. For Lemma 5.4(b) shows that X starts before P and contains slots one to four of P. Slot five of P contains the node (if any) jumped from X.

The two observations imply that each period is assigned at most one unit of $\Delta(W)$, as desired. ∎

Theorem 5.5. Assume that $\frac{f}{s} = \frac{2}{3}$. On any dag the HLF schedule achieves the bound $\frac{\omega}{\omega^*} \leq \frac{6}{5}$.

Proof. Let the segments of the HLF schedule be W_j, $j = S, S-1, \ldots, 1$. From Theorem 5.2(b), $W_j \overset{*}{\rightarrow} W_{j-1}$ for $S \geq j > 1$. Hence $\omega^* \geq \sum_{j=1}^{S} \omega^*(W_j)$. Clearly $\omega = \sum_{j=1}^{S} \omega(W_j)$. Now the Theorem follows from Lemma 5.5. ∎

Now we show that the bound of the Theorem is tight.

Theorem 5.6. Assume that $\frac{f}{s} = \frac{2}{3}$. For any $\epsilon > 0$ there is a dag where the HLF schedule has $\frac{\omega}{\omega^*} > \frac{6}{5} - \epsilon$.

Proof. For any positive integer r consider the dag of Figure 5.5. It consists of r repetition of a module with eight nodes. (As in Figure 5.2 the nodes 7 and 8 are on level 1 but are drawn higher for clarity.)

The HLF schedule is shown in Figure 5.6. Each module is scheduled in six time units so $\omega = 12r$. A better schedule is shown Figure 5.7. Each module schedules node 1 of the next module. This shows $\omega^* \leq 10r + 1$. Thus $\frac{\omega}{\omega^*} = \frac{12r}{10r+1}$ which approaches $\frac{6}{5}$ as r approaches ∞. (Note as previously men-

tioned the optimum schedule has $\Theta(r)$ units of idle time on P_s.) ■

Now we make a simple comparison between the HLF and the AI schedules. When $\frac{f}{s} \geq \frac{4}{5}$ the AI schedule has accuracy $2 - \frac{f}{s} \leq \frac{6}{5}$. We will show that when $\frac{f}{s} \geq \frac{4}{5}$, $\frac{6}{5}$ is a lower bound on the accuracy of the HLF schedule. Hence the HLF schedule is less accurate than the AI schedule for $\frac{f}{s} > \frac{4}{5}$ and no more accurate for $\frac{f}{s} = \frac{4}{5}$. Recall also that the AI scheduling algorithm has a lower time bound. Hence it is the preferable method.

Strictly speaking the HLF schedule is defined only when $\Delta = 1$. However $\frac{f}{s} \geq \frac{4}{5}$ implies that there are no subsumed or subsuming slots among the first four slots on P_f and the first four slots on P_s (since $5f \geq 4s$). Any reasonable definition of HLF will behave just like the $\Delta = 1$ case in these slots. Since the proof below only uses these slots, we omit the restriction $\Delta = 1$.

Theorem 5.7. Assume that $\frac{f}{s} \geq \frac{4}{5}$. There is a dag where the HLF schedule has $\frac{\omega}{\omega^*} = \frac{6}{5}$.

Proof. Consider the dag of Figure 4.2 with the module of Figure 5.8. The module consists of three levels of two nodes each, where each node precedes the two nodes below it. As in Figure 4.2 these nodes precede all nodes of the next module. (This is indicated by heavy arrows.) There are also two isolated nodes in the module. They are exceptions to Figure 4.2 because they do *not* precede the nodes in the next module. (Hence they are on level one of the dag. Note they *are* preceded by the nodes of the previous module.) Choose parameters $m = f$, $c = 2s$, $i = 0$.

Figure 5.9 shows the HLF schedule. Each module is executed as follows: The first two levels jump the isolated nodes of the module and the third level has an idle jump on P_f. Each chain node has an idle jump. Hence $\omega = 4sm + fc = 6sf$.

Figure 5.10 shows the optimum schedule. Each module is executed as follows: Each pair of nodes on a level is executed with idle time on P_f. The isolated nodes of the module are not executed with it. After the modules, the chain nodes are executed on P_f while the isolated nodes of all modules are executed on P_s. Since $c = 2s$ and there are $2m = 2f$ isolated nodes, there is no idle time. So $\omega^* = 3sm + 2fs = 5sf$. Thus $\frac{\omega}{\omega^*} = \frac{6}{5}$ ■

When $\frac{f}{s} < \frac{4}{5}$ the situation is less clear. Our best lower bound for the HLF schedule is appreciably below the AI schedule's accuracy $2 - \frac{f}{s}$. On the other hand our bounds do not rule out the possibility of compression improving the AI schedule, in this range. The main case of interest is $\frac{f}{s} = \frac{3}{4}$ (the only unanalyzed ratio with $\Delta = 1$. For $\frac{f}{s} \geq \frac{3}{4}$ there is a dag where the HLF schedule has $\frac{\omega}{\omega^*} = \frac{7s}{3s + 4f}$ (this is $\frac{7}{6}$ for $\frac{f}{s} = \frac{3}{4}$). The dag is the skeleton of Figure 4.2 with the module of Figure 5.11, and $m = f$, $c = s$, $i = 0$. Further details are left as an exercise.

6. CONCLUSIONS

We have shown that an optimum HLA schedule exists for two uniform processors when $\Delta = 1$. This gives a linear-time algorithm for finding an optimum schedule on dags that have a bounded amount of idle time. It would be of some interest to extend this to an algorithm that finds an HLA optimum schedule in polynomial time, regardless of its structure. We have not succeeded in doing this.

Before summarizing the results for approximate schedules we introduce one more scheme. This is oriented toward processors of disparate speed. An *Approximately One Processor (AO) Schedule* is any schedule with no idle time on P_f. It is easy to see that $\frac{\omega}{\omega^*} \leq 1 + \frac{f}{s}$ for this schedule, since $(\frac{1}{f} + \frac{1}{s})\omega^* \geq \frac{\omega}{f}$.

Figure 6.1 plots the accuracy of the approximate schedules. The AI schedule can be found in linear time and has accuracy $2 - \frac{f}{s}$. It is more accurate than AO when $\frac{f}{s} \geq \frac{1}{2}$. It is more accurate than HLF when $\frac{f}{s} \geq \frac{4}{5}$ and perhaps for other cases. The HLF schedule is defined for $\Delta = 1$. It can be found in $O(m + n \log \log n)$ time. It is most accurate for $\frac{f}{s} = \frac{1}{2}$ and $\frac{2}{3}$.

7. REFERENCES

[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 and Sons, New York, 1976.

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

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

[E2] van Emde Boas, P., "Preserving order in a forest in less than logarithmic time and linear space", *Inf. Proc. Letters* 6, 3, 1977, pp. 80-82.

[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.

[G] Gabow, H. N., "An almost-linear algorithm for two-processor scheduling", *J. ACM*, to appear July 1982.

[GJ1] Garey, M. R. and Johnson, D. S., "Scheduling tasks with nonuniform deadlines on two processors", *J. ACM* 23, 3, 1976, pp. 461-467.

[GJ2] Garey, M. R. and Johnson, D. S., "Two-processor scheduling with start-times and deadlines", *SIAM J. Comput. 6*, 3, 1977, pp. 416-426.

[GLLRK] Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kan, A. H. G., "Optimization and approximation in deterministic sequencing and scheduling: A survey", *Annals of Discr. Math.* 5, 1979, pp. 287-326.

[GT] Gabow, H. N., and Tarjan, R. E., "A linear-time algorithm for set merging on trees", submitted for publication.

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

[L] Lloyd, E. L., "Critical path scheduling with resource and processor constraints", *J. ACM*, July 1982, to appear.

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

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

APPENDIX A. The HLF Algorithm.

This section describes an algorithm that constructs an HLF schedule for processors of speeds f and $f + 1$. The time is $O(m + n \log \log n)$ and the space is $O(m + n)$. The presentation follows that of [G].

The key idea is the notion of "free" node. Consider a level l. Let $(t_1, \ldots, t_k)$ be the HLF jump sequence and let $(t_1, \ldots, t_q)$ be the prefix corresponding to the jumps from levels above l. Loosely speaking the nodes on level l partition into two types: those that *must* be jumped to achieve the jump sequence $(t_1, \ldots, t_q)$ and those that *need not* be jumped. The former are "nonfree" and the latter are "free". The jump from level l (if it exists) can be *from* any free node of l. So to compute an HLF schedule we must determine the free nodes of l and the highest possible jump from a free node of l.

The algorithm consists of two passes. Pass I computes the jump (if any) from each level l, for $l = L, \ldots, 1$. It finds the highest level t that l can jump to. If there is more than one node on level t that can be jumped Pass I guesses one arbitrarily. The guesses allow the free and nonfree nodes to be computed. A guess may be incorrect, in that a free node must be used as the *from* node of a jump may itself be jumped. Pass II fixes the bad guesses.

Bad guesses are fixed using "substitute" nodes. There is a substitute node on each level t. Any jump to a free node of t can be rerouted to go to the substitute. The existence of a substitute follows from the definition of free node. Pass I computes substitute nodes. Pass II fixes bad guesses by rerouting jumps to go to substitute nodes.

Now we give a detailed description of the algorithm, beginning with the data structures. The schedule is specified in arrays FROM and TO. For $L \geq l \geq 1$, $(FROM(l), TO(l))$ is the jump from level l. (So $FROM(l)$ and $TO(l)$ are nodes with $level\ (FROM(l)) = l$, $level\ (TO(l)) < l$). There are two special cases: If

$TO(l)$ = -1, l is not a jumping level; if $TO(l)$ = 0 node $FROM(l)$ is scheduled with an idle jump. Clearly these arrays give enough information to deduce the entire schedule (in linear time), if desired.

The *FROM* and *TO* arrays can be used to store both the jumps that Pass I guesses and the final jumps that Pass II computes. In an actual implementation this should be done. However in the proof of correctness it is desirable to distinguish between guesses and final values. For this reason an array T is used for the guessed *TO*-values. Pass I guesses the *to* nodes of jumps and stores them in T. Pass II copies T to *TO* and then modifies *TO* to the final jumps.

The *COUNT* array is used to compute when a node can be jumped. For each node y, *COUNT* (y) is initially the number of immediate predecessors of y. *COUNT* (y) is decreased each time a predecessor is executed. When *COUNT(y)* is zero y can be jumped.

Q is a priority queue used to compute the highest node to jump. An element of Q is a node, with priority given by its level. The queue primitives used are *INSERT* and *EXTRACT−MAX*. These are defined as usual [AHU] with one slight modification: *EXTRACT−MAX* finds and deletes the node of highest level in Q. The modification is that when there is a tie the node deleted is the one that was inserted most recently. (This *lifo* policy is necessary for the proper computation of substitute nodes). Q can be implemented using the data structure of van Emde Boas [E], since the priorities are integers in the range 1 to n.

When *level* l is processed Q contains exactly the nodes that can be jumped from l (after line 10 of the algorithm). Hence the highest jump can be found by an EXTRACT-MAX operation. (It is convenient to keep a dummy node 0 in Q to handle idle jumps. Node 0 is on a fictitious level 0. A jump to 0 is an idle jump.)

READY is a list of nodes where *COUNT* is zero and hence can be inserted in Q. Nodes are placed in *READY* by an auxiliary procedure *SCAN* (line 2). Nodes

are transferred from *READY* to Q in lines 5 and 10. *NODES* is a list of nodes to be processed by *SCAN* (see lines 1, 7, 15).

To keep track of free nodes, each node is initially marked "free". The mark is changed to "nonfree" when groups of nonfree nodes are discovered (line 15). The *SUB* array stores substitute nodes: for each level l, $SUB(l)$ is the substitution level l. *SUB* is computed in Pass I (line 6) and used in Pass II (line 18).

The variable *slot* keeps track of the current slot in the schedule. Its values range from 1 to $2f + 1$. It is updated using the mod^+ function, where $a\ mod^+ b = a\ mod\ b$ if $b \nmid a$, and b if $b \mid a$.

By consulting *slot* it can be determined whether or not a level has a jump.

The algorithm works as follows. Pass I processes levels l in decreasing order, $l = L, \ldots, 1$. For each l, *SCAN* "executes" the free nodes by decreasing *COUNT* values. Nodes that can be jumped from l are inserted into Q, and the jump from l is computed. If it is *to* a nonfree node, nodes are marked nonfree (and *SCAN*ned).

Pass II processes levels l in increasing order, $l = 1, \ldots, L$. For each l, a correct node $FROM(l)$ is found. If $FROM(l)$ happens to be jumped by Pass I, the jump is switched to go to $SUB(l)$ instead of $FROM(l)$.

Now we give the algorithm in pseudo-Algol. The algorithm computes an HLF schedule for processor speeds f and $f + 1$, $f \geq 1$.

procedure *H;* **begin**

procedure *SCAN;* **begin**

1. **for** $x \in$ *NODES* **do**
   for each edge (x, y) **do begin**

2. decrease $COUNT(y)$ by 1;

if $COUNT(y) = 0$ **then** add y to $READY$;

end end $SCAN$;

Initialization:

3. partition the nodes of the dag into levels L, ..., 1; set $COUNT(y)$ to the number of immediate predecessors of y, for each node y; set $READY$ to contain all initial nodes of the dag; mark each node y free; initialize Q to contain dummy node 0 on level 0; $slot \leftarrow 0$;

Pass I:

4. **for** $l \leftarrow L$ **to** 1 **by** -1 **do begin**

5. remove each node from $READY$ and insert it in Q;

6. remove the remaining nodes of l from Q; **comment** these are the highest priority nodes in Q; let $SUB(l)$ be the last such node;

7. $NODES \leftarrow$ the free nodes of level l; $SCAN$;

8. $u \leftarrow$ the number of nodes on level l that are not jumped;
$slot \leftarrow (slot + u)\ mod^{+}(2f + 1)$;

9. **if** $slot = 2f + 1$ **then comment** no jump; $T(l) \leftarrow -1$

else comment jump; **begin**

10. **for** $y \in READY$ **do**
if some free node on l does not immediately precede y **then**
remove y from $READY$ and insert it in Q;

11. let y be the highest node in Q; $T(l) \leftarrow y$; $t \leftarrow level(y)$;

12. **if** $y = 0$ **then** $slot \leftarrow 0$

else begin comment nonidle jump;

13. $slot \leftarrow slot + 1$; remove y from Q;

let z be the highest node in Q;

14. **if** $t > level(z)$ **then comment** the jump is to a nonfree node of t **begin**

15. $NODES \leftarrow$ the nodes of level t that are jumped and still designated free;

mark each $x \in$ NODES nonfree;

SCAN

end end end end Pass I;

Pass II:

let $TO(k) = T(k)$ for $1 \leq k \leq L$ **comment** TO and T can be the same array;

16. **for** $l \leftarrow 1$ **to** L **do**

if $TO(l) \geq 0$ **then begin**

17. let $FROM(l)$ be a node on level l, that does not immediately

precede $TO(l)$ if $TO(l) > 0$;

18. **if** $FROM(l) = TO(k)$ for some k **then** $TO(k) \leftarrow SUB(l)$;

end end H.

Now we prove that the algorithm is correct. Let the *H schedule* be the one computed by the algorithm, i.e., the level schedule with jumps $(FROM(l), TO(l))$, $L \geq l \geq 1$. The proof is organized as follows: Lemmas A.0-4 give the basic properties of Pass I. Lemma A.5 shows how Pass II modifies jumps to get the H schedule. Corollaries A.1-4 give properties of the H schedule that are analogous to Lemmas A.1-4. These properties include the facts that H is a valid schedule (Corollary A.3) and H has an HLF-like property (Corollary A.4). The latter is used to prove that H is HLF (Lemma A.6) and has a block structure (Lemma

A.8).

The proof assumes in its organization that the algorithm runs to completion. Inspection reveals two places where the algorithm could conceivably halt prematurely: In line 6 node $SUB(l)$ might not exist if no nodes are removed from Q; in line 18 a node $FROM(l)$ with the desired properties might not exist. We assume at the outset of the proof that in both cases if a node does not exist, the algorithm skips to the next line and continues execution. We will see that actually the nodes exist: A remark following Lemma A.3 shows SUB(l) exists, and Corollary A.2 shows $FROM(l)$ exists.

The proof treats 0 as a dummy node on a fictitious level 0. Thus a level l with TO(l) = 0 jumps to node 0. Similarly an assertion like "$level\ (TO(l))\ > k$" means TO(l) is a real node, above level k.

It is easy to see that every node is inserted into Q in line 5 or line 10 and removed from Q in line 6 or 13. It is convenient to define a function $R(y)$ to indicate when this occurs. Specifically,

$R(y)$ = the value of l when y is inserted in Q.

Loosely speaking $R(y)$ tells when node y can be jumped. This gives another characterization of "free". Call a node *free* if it is still marked free at the end of the algorithm.

Lemma A.0. A node x is free if and only if $R(x) \leq R(SUB(level(x)))$.

Proof. For convenience let $t = level(x)$. Let y be the last node (if any) on level t that causes nodes of t to be marked nonfree (in line 15). Observe that the first node of t to be inserted in Q after y is removed is $SUB(t)$. (This follows from the *lifo* policy used in Q for nodes on the same level.) Any node x (on level t) inserted after $SUB(t)$ is free and has $R(x) \leq R(SUB(t))$. Any node x inserted before $SUB(t)$ is nonfree and has $R(x) > R(SUB(t))$. ∎

The following property of Pass I says that if a level l jumps a free node of a level t then no subsequent jump from above t goes below t.

Lemma A.1. Let l be a jumping level where node $T(l)$ is free. Let k be a jumping level where $l \geq k > level(T(l))$. Then $level(T(k)) \geq level(T(l))$; if equality holds then $T(k)$ is free.

Proof. Since $T(l)$ is free, node $SUB(level(T(l)))$ is in Q from levels $R(T(l))$ to $level(T(l))$. Hence all such levels k jump to $level(T(l))$ or higher. Further if $level(T(k)) = level(T(l))$ then $T(k)$ is free since $SUB(level(T(k)))$ is in Q at level k. ∎

The next Lemma will be used to show that H respects precedence. It says that Pass I executes any immediate predecessor of a node y at level $R(y)$ or earlier.

Lemma A.2. If $x \rightarrow y$ then either $level(x) \geq R(y)$ or x is nonfree and $x = T(l)$ for some level $l > R(y)$.

Proof. If $x \rightarrow y$ then x must be scanned before $COUNT(y)$ decreases to zero. If x is scanned in line 7 then clearly $level(x) \geq R(y)$. If x is scanned in line 15 then x is nonfree and is jumped from a level above $R(y)$. ∎

The next Lemma will be used to show that FROM nodes exist.

Lemma A.3. Let y be a node with $R(y) \geq k$ for some jumping level k. Then k contains a free node x, $x \not\rightarrow y$.

Proof. First observe that any level l has a node that is not jumped in Pass I (and hence is free). This is obvious for $l = L$. By induction assume it holds for level $l + 1$. Let v be a free node on level $l + 1$ and let w be a node on level l with $v \rightarrow w$. Lemma A.2 shows $l + 1 \geq R(w)$. If w is jumped clearly $l + 1 = R(w)$.

The test of line 10 shows some free node v' on level $l + 1$ does not precede w. So v' precedes a node w' on level l, and $l + 1 \geq R(w')$. Clearly w' is not jumped. The observation now follows by induction.

Now we prove the contrapositive of the Lemma. Suppose all free nodes of k precede y. This it is easy to see from line 10 that $R(y) < k$. ∎

Notice that the observation in the proof implies that node $SUB(l)$ always exists in line 6, so the algorithm does not abort.

The next Lemma essentially shows the HLF property for Pass I. To motivate its statement let l be a jumping level and let $z = T(l)$. The HLF property implies that any node y above $level(z)$ cannot be jumped from l. Thus if $level(y) > level(z)$ and y is scheduled after l then all free nodes of l precede y. This is Lemma A.4(a). Lemma A.4(b) shows the related fact, that all nonfree nodes must indeed be jumped, or equivalently, a free node cannot be substituted for a nonfree node.

Lemma A.4. Let l be a jumping level. Let y be a node executed after l by Pass I, i.e., $l > level(y)$ and $y \neq T(k)$ for any $k \geq l$. Let $z = T(l)$ and suppose that either

(a) $level(y) > level(z)$

or

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

Then all free nodes of l precede y.

Proof. First note that without loss of generality y has no predecessors executed after l. For let x be such a predecessor. It is easy to see that x satisfies the hypotheses of the Lemma (in particular, alternative (a)) and the conclusion for x gives the conclusion for y.

So any predecessor x of y is executed before or at level l. Suppose $level(x) < l$. So x is jumped from above l. (x is not jumped from l since $x \neq z$ as $level(x) > level(y) \geq level(z)$.) Since level l jumps below $level(x)$ all nodes of $level(x)$ have been removed from Q by the time l is processed. So x has already been scanned. Thus $R(y) \geq l-1$, and $R(y) = l-1$ only if the test of line 10 fails, i.e., all free nodes of l precede y. Hence it suffices to show that $R(y) = l-1$.

To do this suppose the contrary, $R(y) \geq l$. This implies that y has been inserted in Q by the time line 11 is reached for level l. Also y has not been deleted from Q, from the hypothesis. Thus level l jumps to $level(y)$ or higher, i.e., $level(z) \geq level(y)$. So alternative (b) holds. But if y is free then SUB($level(y)$) is already in Q at level l. So z is also free. This is the desired contradiction. ∎

Now we examine how Pass II computes TO-values.

Lemma A.5. For any jumping level k, TO(k) is either $T(k)$ or SUB($level(T(k))$). In the latter case $T(k)$ is free. In both cases $R(TO(k)) \geq k$.

Proof. At the start of Pass II any value $TO(k)$ is $T(k)$. Line 18 may change TO(k) from $T(k)$ to $SUB(l)$ where $l = level(T(k))$. This is done only if $T(k)$is free (by line 17). Further TO(k) is not changed again since the new value is still on $level(T(k))$.

It remains only to show $R(TO(k)) \geq k$. Since a node is jumped after it is inserted in Q, $R(T(k)) \geq k$. An if $T(k)$ is free Lemma A.0 implies $R(SUB(level(T(k))) \geq k$. ∎

Corollary A.1. For any jumping level k, $level(T(k)) = level(TO(k))$. $T(k)$ is free if and only if TO(k) is free. If $T(k)$ is nonfree then $T(k) = TO(k)$. ∎

Now we can show that the H schedule is well-defined, i.e., the FROM and TO arrays specify the jumps of a level schedule. This means first that the FROM nodes specified in line 17 actually exist; second, no FROM node is itself jumped.

Corollary A.2 For any jumping level l node FROM(l) exists and is not jumped (i.e., FROM(l) $\neq$ TO(k) for any k).

Proof. When lines 17-18 are executed for level l, TO(l) has its final value. By Lemma A.5 $R(TO(l)) \geq l$. So in line 17 node FROM(l) exists by Lemma A.3. Line 18 ensures that FROM(l) is not jumped.

The next result shows that the H schedule is a valid schedule, i.e., it respects the precedence constraints.

Corollary A.3. If $x \rightarrow y$ then the H schedule completes node x before it starts node y.

Proof: Since H is a level schedule the conclusion is clear if y is not jumped. So suppose y = TO(l) for some level l.

Lemma A.5 shows $R(y) \geq l$. So from Lemma A.2 either $level(x) \geq l$ or x is nonfree and $x = T(k)$ for some level $k > l$.

First suppose $level(x) \geq l$. Then x is completed before y unless x = FROM(l). But the latter is impossible by line 17.

So suppose x is nonfree and $x = T(k)$, $k > l$. By Corollary A.1 $x = TO(k)$. So x is jumped before l and the desired conclusion follows. ∎

Finally we show a version of the HLF property for H. This version is analogous to Lemma A.4, and implies the HLF property and the block structure of the H schedule.

Corollary A.4. Let l be a jumping level. Let y be a node executed after level l in

the H schedule. Let $z = TO(l)$ be the node jumped from l, where either

(a) $level(y) > level(z)$

or

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

Then all free nodes of l precede y.

Proof: It suffices to show that the hypotheses of Lemma A.4 hold for y since the Lemma has the desired conclusion.

We first show that y is executed after l by Pass I. Since this holds for the H schedule, $l > level(y)$ and $y \neq$ TO(k) for $k \geq l$. So it suffices to show that $y \neq T(k)$ for any $k \geq l$. Suppose on the contrary that $y = T(k)$. This means Pass II changes TO(k). So y is free. Lemma A.1 (and Corollary A.1) show $level(z) \geq level(y)$. So alternative (b) holds. Now Lemma A.1 (and Corollary A.1) show that z is free. But this contradicts (b).

It remains to show that alternatives (a) or (b) of Lemma A.4 hold. Each is implied by its counterpart in Corollary A.4, by Corollary A.1. ∎

Lemma A.6. The H schedule is an HLF schedule.

Proof. Let the H schedule have jump sequence $(t_1, \ldots, t_k)$. Let S be an arbitrary level schedule with jump sequence $(s_1, \ldots, s_r)$. We wish to show that $(t_1, \ldots, t_k) \geq (s_1, \ldots, s_r)$ where $\geq$ denotes lexicographic order. This is done by proving inductively that for all i, $1 \geq i \geq min(k, r)$,

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

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

Note that for any index i if inequality holds in (i) then the induction is com-

pleted trivially and the desired conclusion follows. On the other hand if (i) holds with equality for $i = min(k,r)$ then it is easy to see that $k = r$, $(t_1, \ldots, t_k) = (s_1, \ldots, s_r)$ and again the desired conclusion follows.

So assume that (i)-(ii) hold for indices strictly less than i. We prove (i)-(ii) for i as follows. As mentioned above we can assume $(t_1, \ldots, t_{i-1}) = (s_1, \ldots, s_{i-1})$ if $i > 1$. This implies that the ith jumping level is the same in both schedules, call it l. Let the jump from l be *from* node x in schedule S and *to* node z in schedule H. (Thus $z = TO(l)$ and $level(z) = t_i$.) We will show that (i) and (ii) both follow from Corollary A.4.

First observe two properties that hold for both S and H:

(1) All nonfree nodes of l are jumped from above l.

(2) All nonfree nodes of $level(z)$ are jumped from above l, if z is free.

(1) is obvious for H. (2) holds for H because of Lemma A.1 and Corollary A.1. Furthermore (1) and (2) for H imply their counterparts for S because of (ii).

Next observe that node x is free. For x is not jumped in S and so it is free by (1).

Property (i) means that in S, l jumps to $level(z)$ or below. Equivalently if g is a level with $l > g > level(z)$ then S does not jump to g. To show this suppose H executes b nodes of g before l and a nodes of g after l. (Of course H does not execute any nodes of g at level l.) S executes b nodes of g before l, since (i) holds with equality. Further if y is a node on level g that H executes after l, then by Corollary A.4(a), x precedes y. (Recall x is a free node of l.) So S executes y after l. Thus S executes a nodes of g after l. No nodes of g remain for S to jump from l. This proves (i).

Property (ii) means that z is free if and only if S jumps a free node of $level(z)$. If z is free, (2) implies that S can only jump a free node of $level(z)$, as

desired.

On the other hand suppose that z is nonfree. Let y be a free node of $level(z)$. H executes y after l (by Lemma A.1 and Corollary A.1). So x precedes y by Corollary A.4(b). Thus S can only jump a nonfree node of $level(z)$, as desired.

This completes the proof of correctness of algorithm H. Before analyzing the time we show how Corollary A.4 also implies the block structure of the schedule.

Blocks are defined from boundary levels. The *boundary levels* l_i, $1 \le i \le B+1$, are defined as follows: $l_1 = 1$. For $i > 1$, l_i is the lowest jumping level such that $l_i > l_{i-1}$ and either

(a) l_i jumps below l_{i-1}, i.e., $level\ (\mathrm{TO}(l_i)) > l_{i-1}$,

or

(b) l_i jumps to a nonfree node on l_{i-1}, i.e.,

$level(\mathrm{TO}(l_i)) = l_{i-1}$ and $\mathrm{R}(\mathrm{SUB}(l_{i-1})) < l_i$.

Let l_B be the last value defined using the above criteria, and set $l_{B+1} = L + 1$. Note that any level l with an idle jump ($\mathrm{TO}(l) = 0$) is a boundary level l_i. (This follows from the convention that 0 is a dummy node on level 0.)

For $1 \le i \le B$, *block* X_i consists of all nodes scheduled after level l_{i+1}, up to and including l_i, except for the node (if any) jumped from l_i. It is easy to see that $X_i = \{\, x \mid l_{i+1} > level(x) \ge l_i$ and x is not jumped from l_{i+1} or above $\}$. Any node is in exactly one block, except for a node jumped from a boundary level (which is in no block).

Now we show the block precedence property.

Lemma A.7. For a block X_i, $1 < i \le B$, any node $x \in X_i$ on level l_i precedes all

nodes of X_{i-1}, i.e., $x \overset{*}{\rightarrow} X_{i-1}$.

Proof. First note that for any block X_i, $1 \leq i \leq B$, any node $x \in X_i$ on level l_i is free. For suppose on the contrary that x is nonfree. So x is jumped from some level l. Since x is on l_i it is easy to see that $l \geq l_{i+1}$. But then $x \notin X_i$, a contradiction.

To show the Lemma take any $x \in X_i$ on level l_i and any $y \in X_{i-1}$. x is free by the above remark; similarly if y is on l_{i-1} it too is free. This shows the hypotheses of Corollary A.4 are satisfied for level l_i and node y. Thus $x \overset{*}{\rightarrow} y$.

Lemma A.8. For a block X_i, $1 < i \leq B$, $X_i \overset{*}{\rightarrow} X_{i-1}$.

Proof. Consider any node $x \in X_i$. by Lemma A.6 it suffices to show that x has a successor z on level l_i with $z \in X_i$.

The definition of block shows that $level(x) \geq l_i$. Clearly we can assume $level(x) > l_i$. So x has a successor z on level l_i. z is executed after x, whence after level l_{i+1}. So $z \in X_i$ as desired. ∎

This completes the proof of our first goal:

Theorem 5.2. Assume that $\Delta = 1$.

(a) In an HLF schedule any block X is executed in consecutive time slots $i, i+1, \ldots, j$ with no intervening idle time. Here $1 \leq i, j \leq 2f + 1$ and $i \neq 2$.

(b) For any block X_i, $1 < i \leq B$, $X_i \overset{*}{\rightarrow} X_{i-1}$.

Proof. (a) follows from the definition of block. (b) is Lemma A.8. ∎

Now we analyze the timing of algorithm H. We show that the Q instructions use $O(n \log \log n)$ time while the remainder of the algorithm is $O(m + n)$.

First we describe some additional data structures. The dag is stored in an adjacency structure: each node has a list of its immediate predecessors and a list of its immediate successors. Level information is stored in two ways: An array *LEVEL* gives the level of each node, i.e., node x is on $LEVEL(x)$. Also each level has a list of the nodes on that level. This data structure for level information is constructed in line 3 when levels are found in $0(n)$ additional time.

An array T' indicates when each node is jumped in Pass I. More precisely for each node x, $T'(x) = l$ if and only if $x = T(l)$; if x is not a T-value then $T'(x) = -1$. T' is initialized to -1 in line 3 and values are assigned to T' when T is assigned in line 11. Clearly the total time spent computing T' is $0(n)$.

With these data structures it is easy to see that, excluding Q operations, line 10 and line 15, the time is $0(m + n)$, because $0(1)$ time is spent on each edge, node, or level: Line 3 finds the levels of the dag by using predecessor lists in a modified topological sort [Kn]. Line 7 uses the list of nodes on level l to find the free nodes on l. Line 8 uses T' to find the number of nodes on l that are not jumped. Line 17 finds node FROM(l) by flagging the immediate predecessors of TO(l) that are on level l and finding a free, unflagged node on the list of nodes on level l. For line 18, note that FROM(l) = TO(k) if and only if FROM(l) = $T(k)$. So line 18 uses T' to find level k.

Now we discuss the remaining lines 10 and 15. The only hard part of line 10 is the test that some free node on level l does not immediately precede y. To do this the algorithm stores, for each level l, a count of the free nodes on l. This count is computed when the free nodes of l are accessed in line 7. Line 10 computes the number of free immediate predecessors of y on level l. This number is less than the count for l if and only if the test has an affirmative answer. So the total time spent in the test in line 10 is $0(m + n)$.

The only hard part of line 15 is constructing the NODES list. To do this the algorithm maintains, for each level t, a list of the nodes on t that are jumped and still designated free. (Nodes are inserted in the list in line 11 and deleted in line 15.) Line 15 empties the list for level t into NODES. So the total time associated with these lists and line 15 is $0(n)$.

Finally we discuss the time for operations on Q. It is convenient to implement Q as a priority queue over the levels $1, \ldots, L$. A level l is in the priority queue when at least one node on level l is in Q. An entry for l in the queue points to a stack of all the nodes on l that are in Q. This make it easy to implement the *lifo* deletion policy for Q. Also the test of line 14 can be done without actually finding z.

Every node is inserted into Q in line 5 or line 10 and deleted in line 6 or line 13. There are L additional FIND-MAX operations in line 11. Hence the total time for all Q operations is $0(n \log\log n)$. [E]

Observe that the space is $0(m + n)$ since all data structures use $0(1)$ space for each node, edge or level [E2]. Now we summarize the results.

Theorem 5.1. Assume that $\Delta = 1$. An HLF schedule can be found in $0(m+n \log\log n)$ time and $0(m+n)$ space.

Proof. Algorithm H finds an HLF schedule by Lemma A.6. The time and space bounds follow from the above discussion. ∎

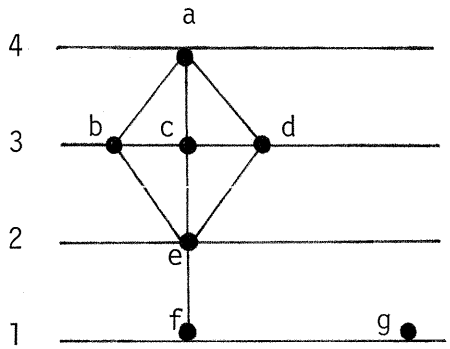
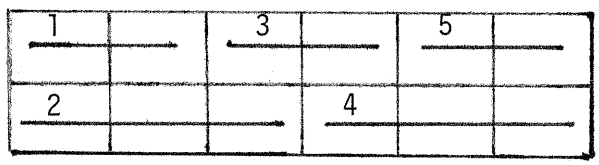


Figure 2.1

Dag with four levels.

(a)



(a)

(b)

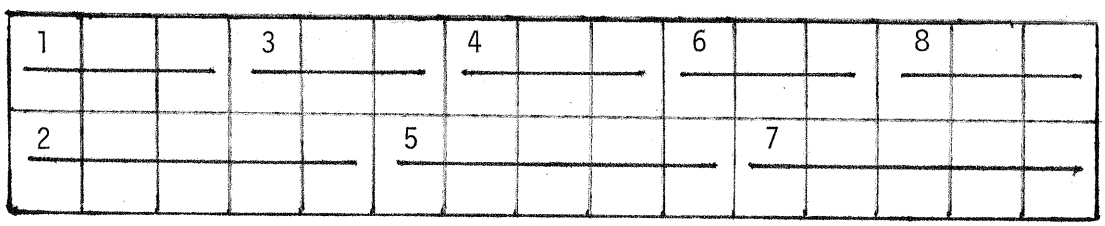
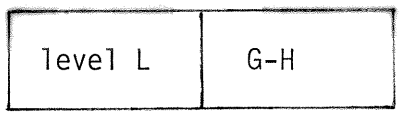
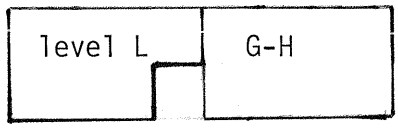


Figure 2.2

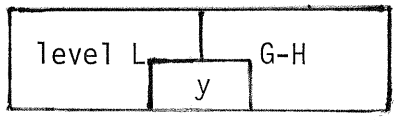
Execution periods (a) $\frac{f}{s} = \frac{2}{3}$ (b) $\frac{f}{s} = \frac{3}{5}$.



(a) No jump.



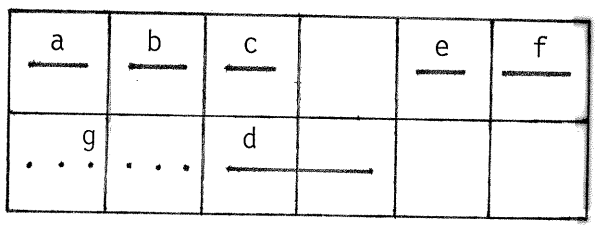
(b) Idle jump.



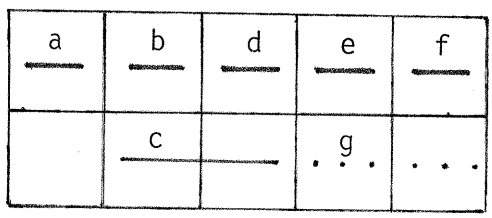
(c) Jump to y.

Figure 2.3

Level schedule definition.



(a) $\omega = 6$



(b) $\omega^* = 5$

Figure 2.4

Level schedules for Figure 2.1, $\frac{f}{s} = \frac{1}{2}$.

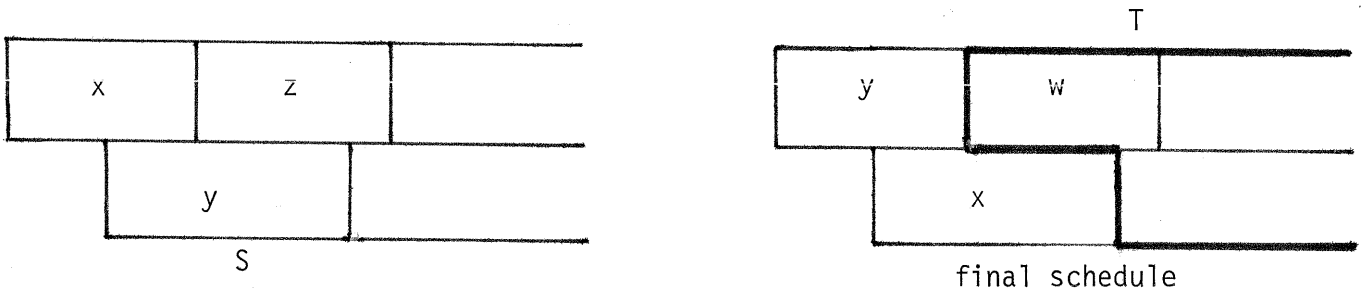


Figure 3.1

Schedules in the proof of Lemma 3.2.

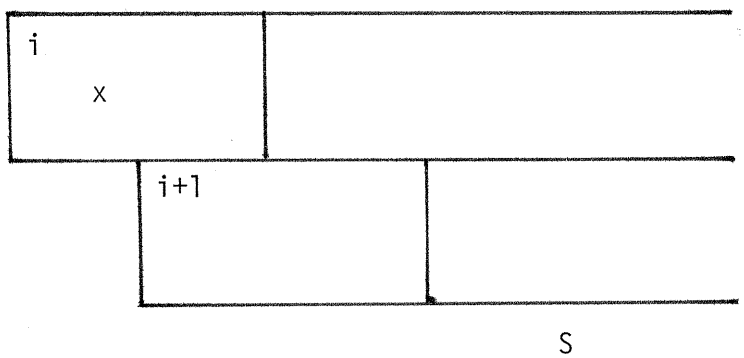


Figure 3.2

Schedule in the proof of Lemma 3.3.

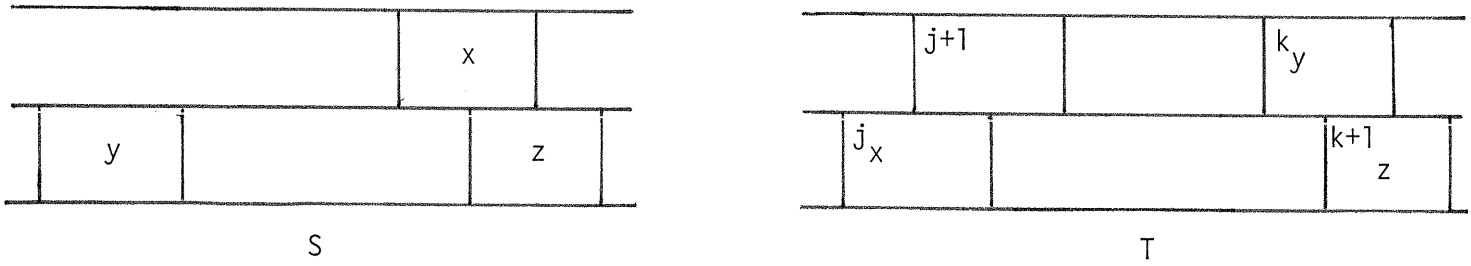


Figure 3.3

Schedules in the proof of Lemma 3.4.

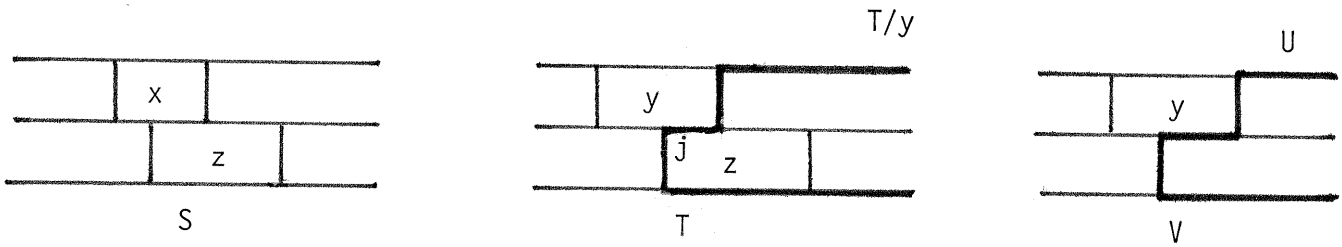


Figure 3.4

Schedules in the proof of Lemma 3.5.

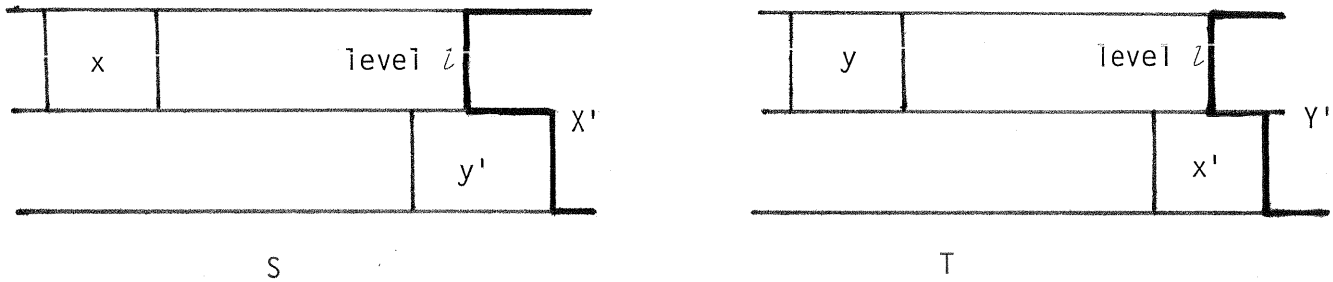


Figure 3.5

Schedules in Lemma 3.6.

Figure 3.6

Schedules in the proof of Theorem 3.1.

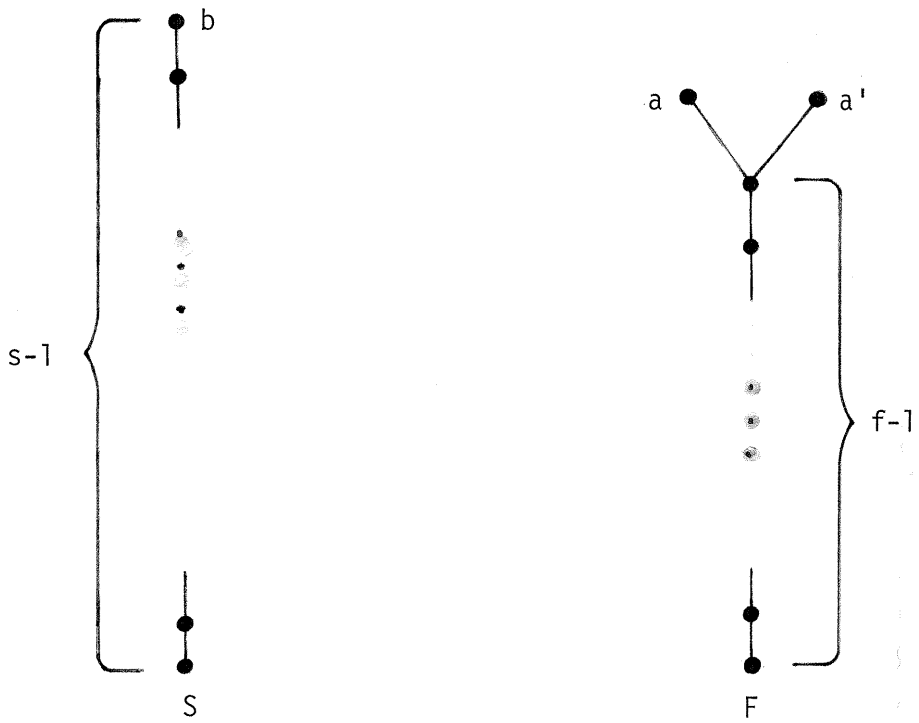


Figure 3.7

Nonlevel dag for 2f > s.

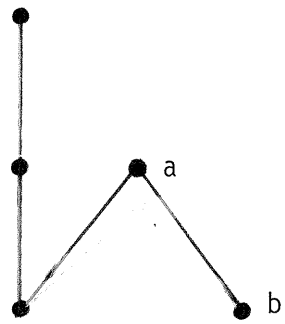


Figure 3.8

Nonlevel dag for $2f < s$.

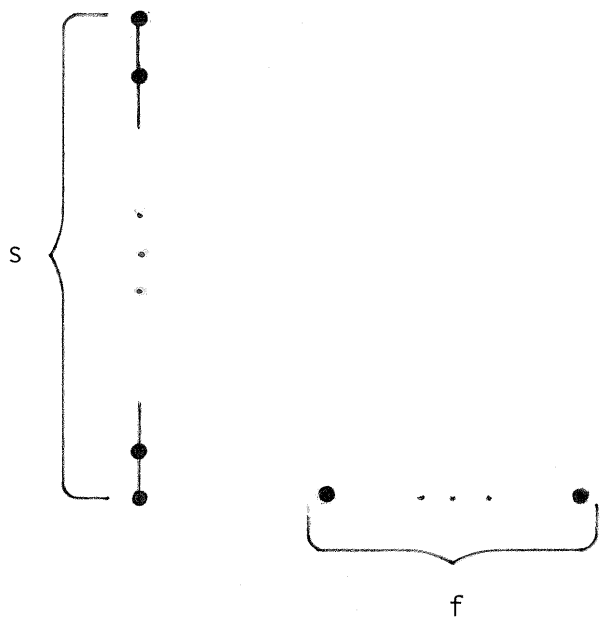


Figure 4.1

Bad dag for AI schedule.

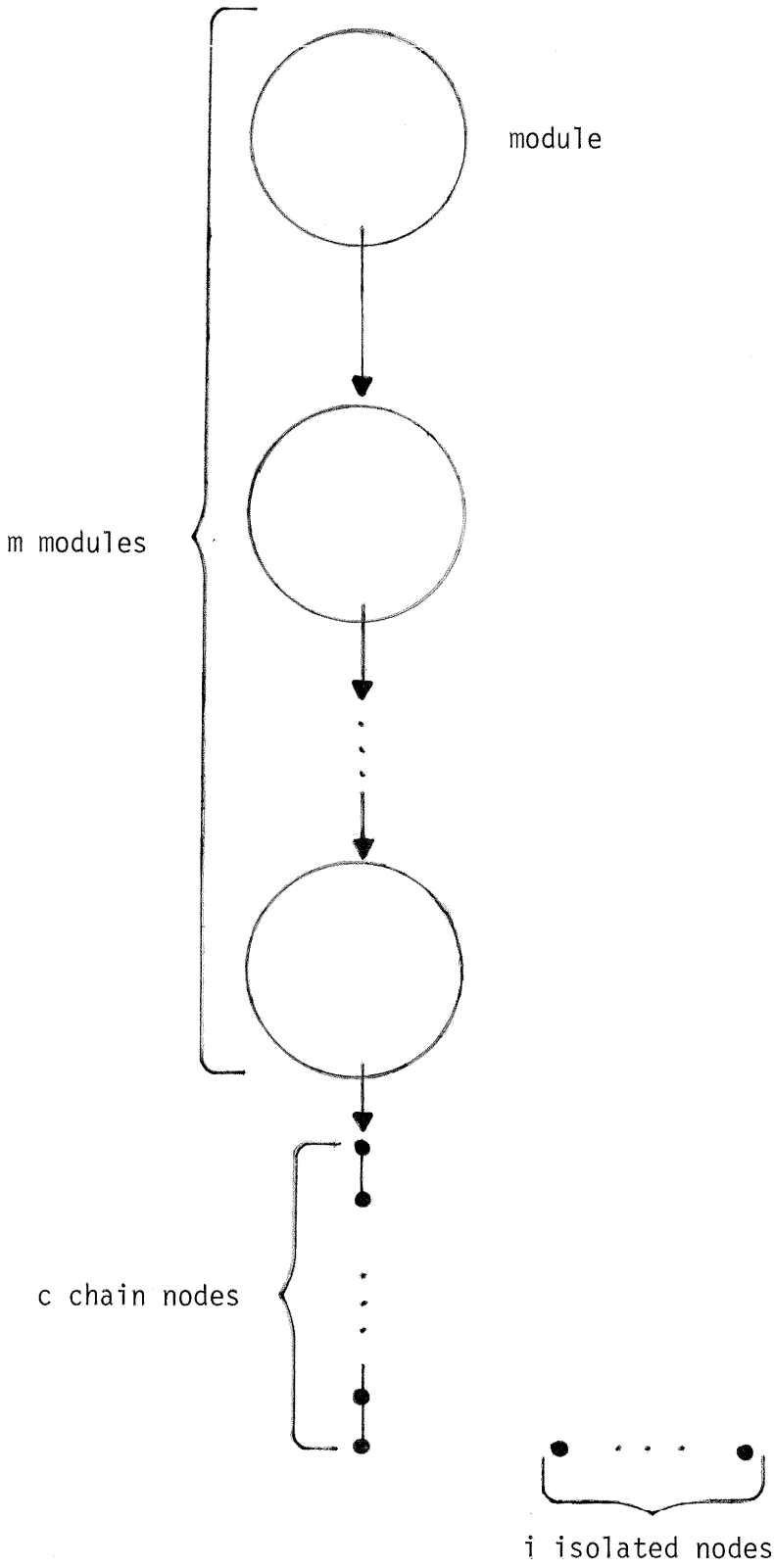


Figure 4.2

Bad dag skeleton for CAI schedule.

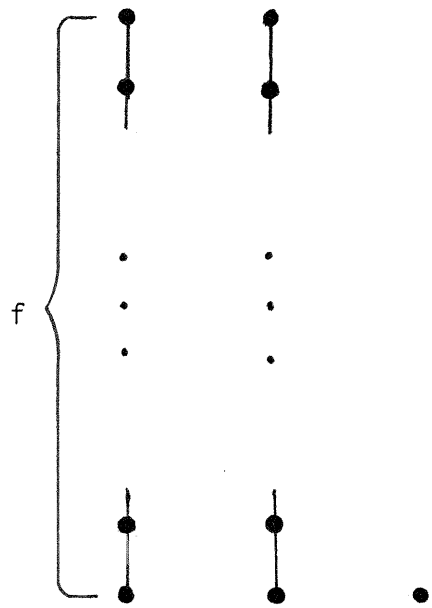


Figure 4.3

Module for $\Delta = 1$.

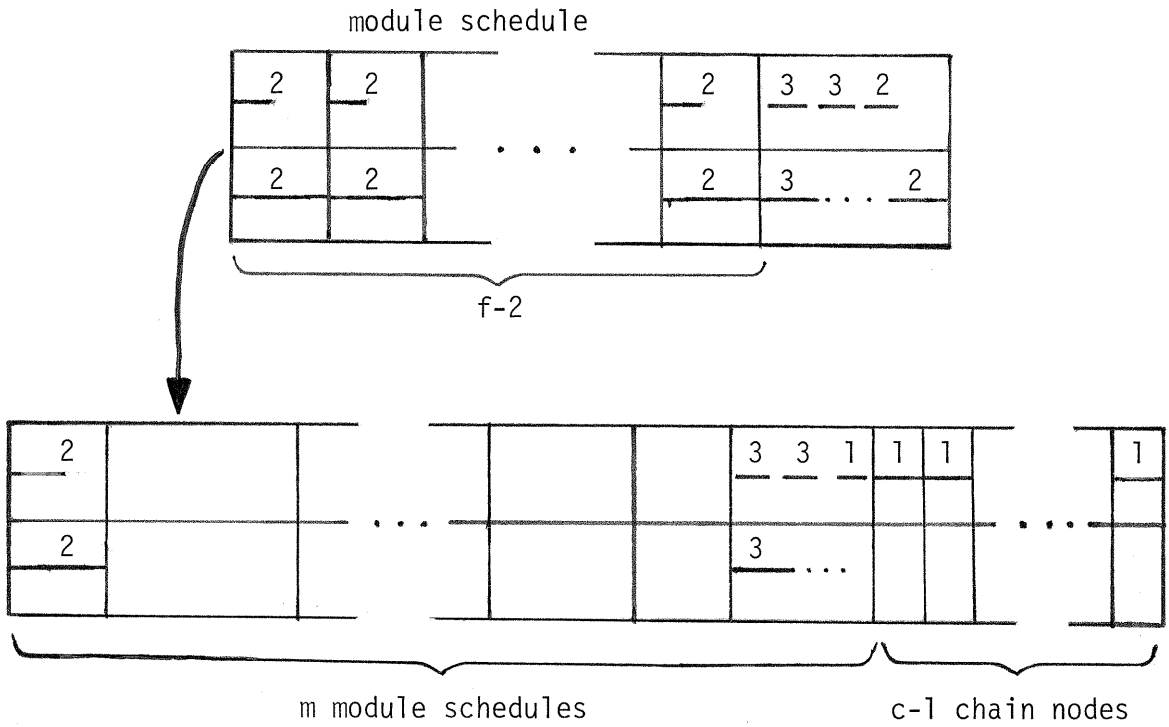


Figure 4.4

The CAI schedule.

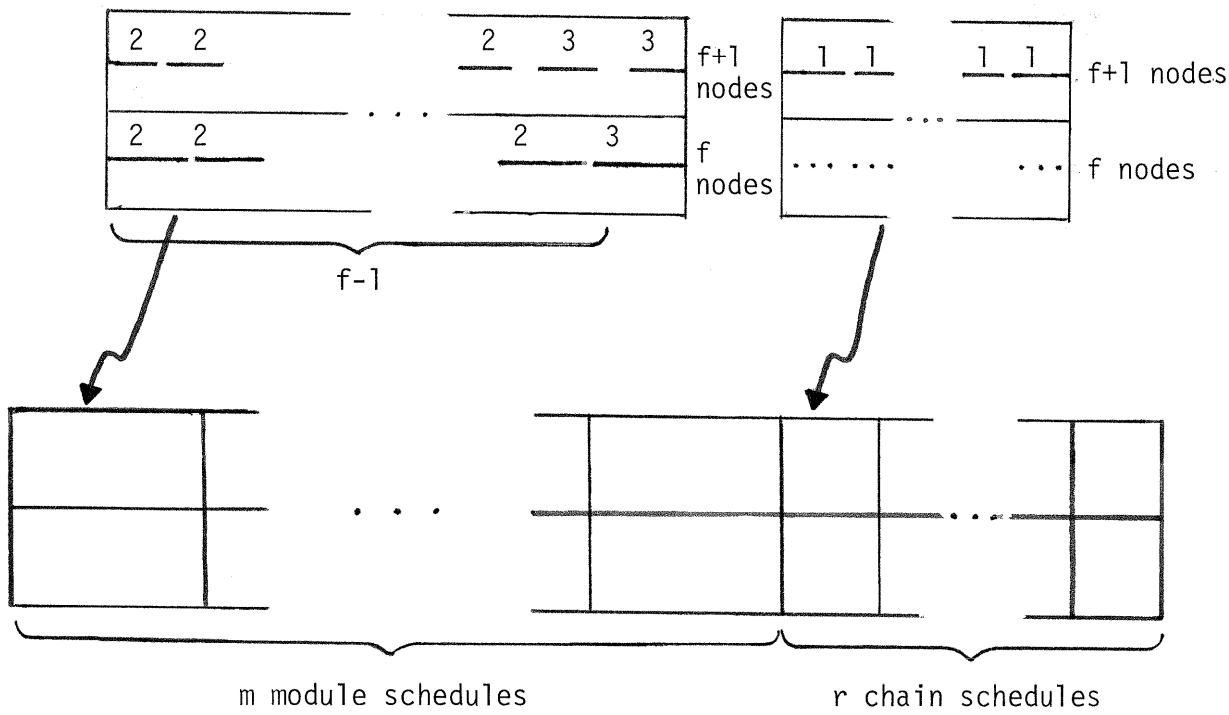


Figure 4.5

The optimum schedule.

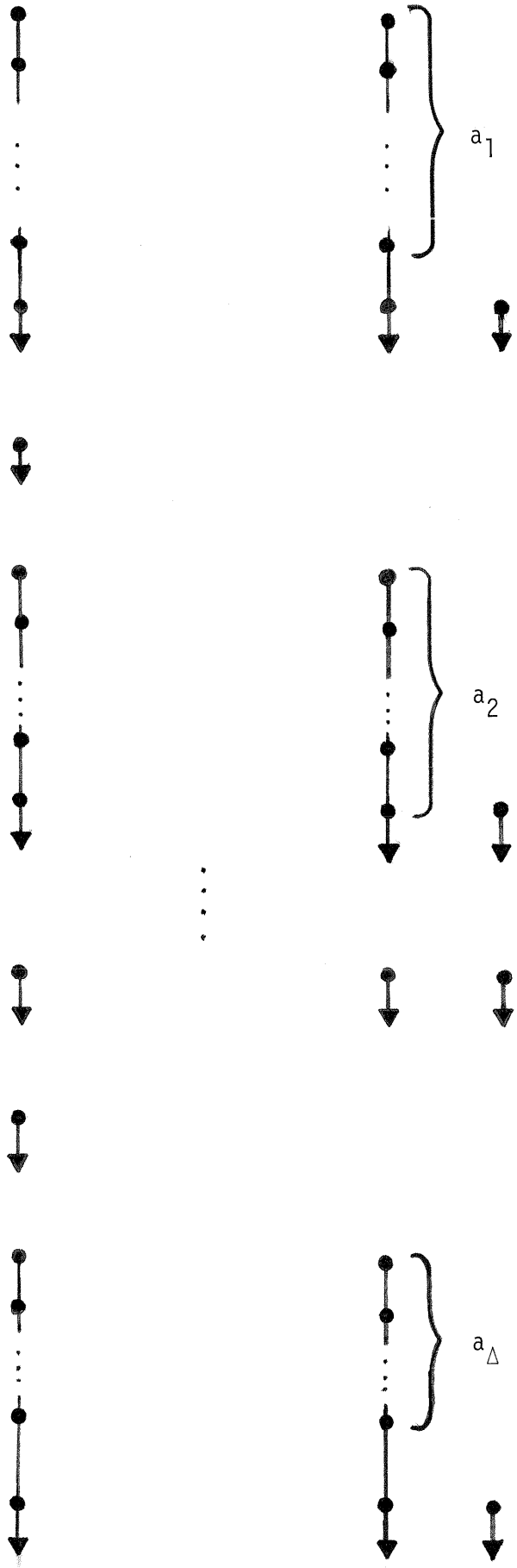
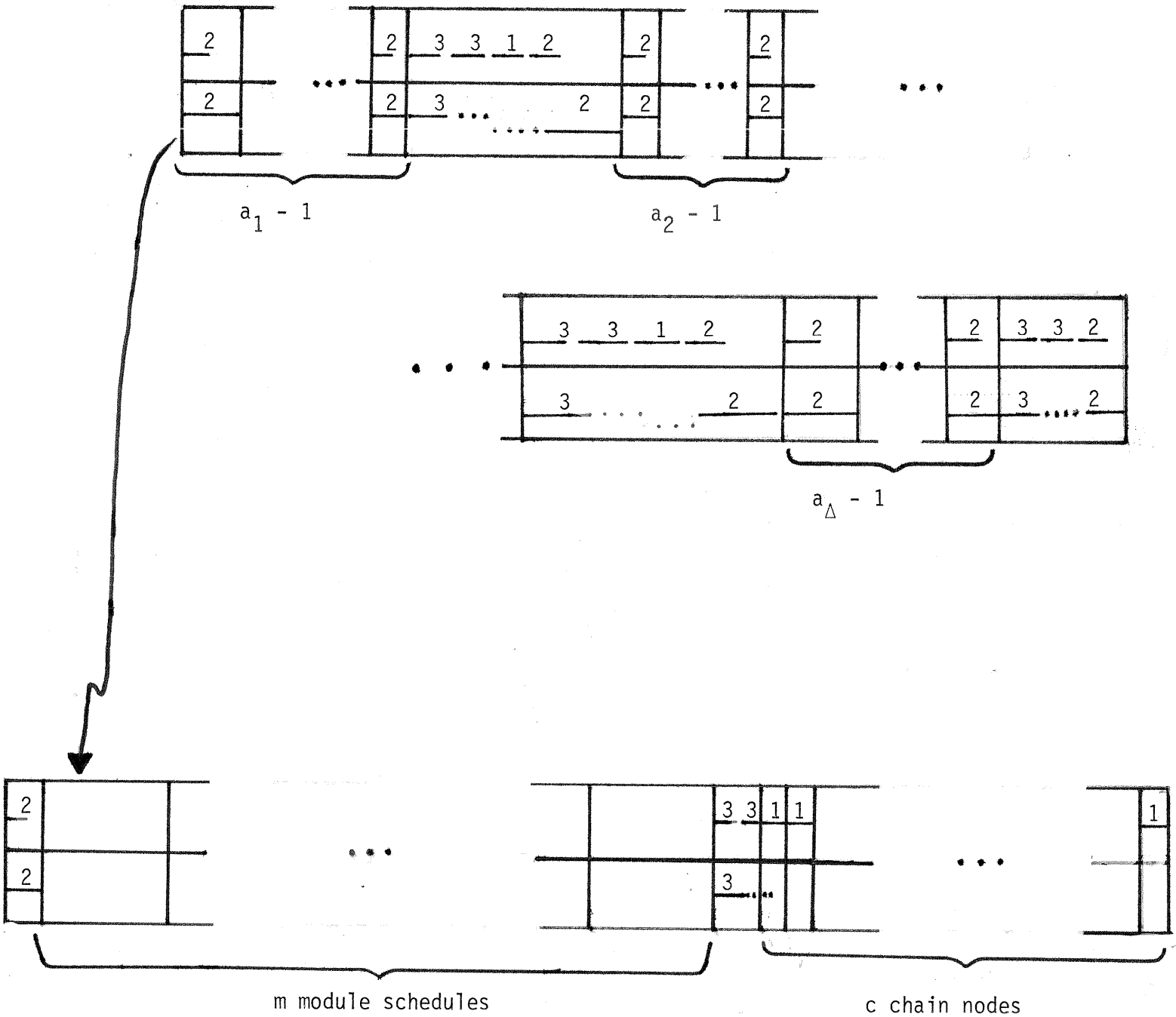


Figure 4.6

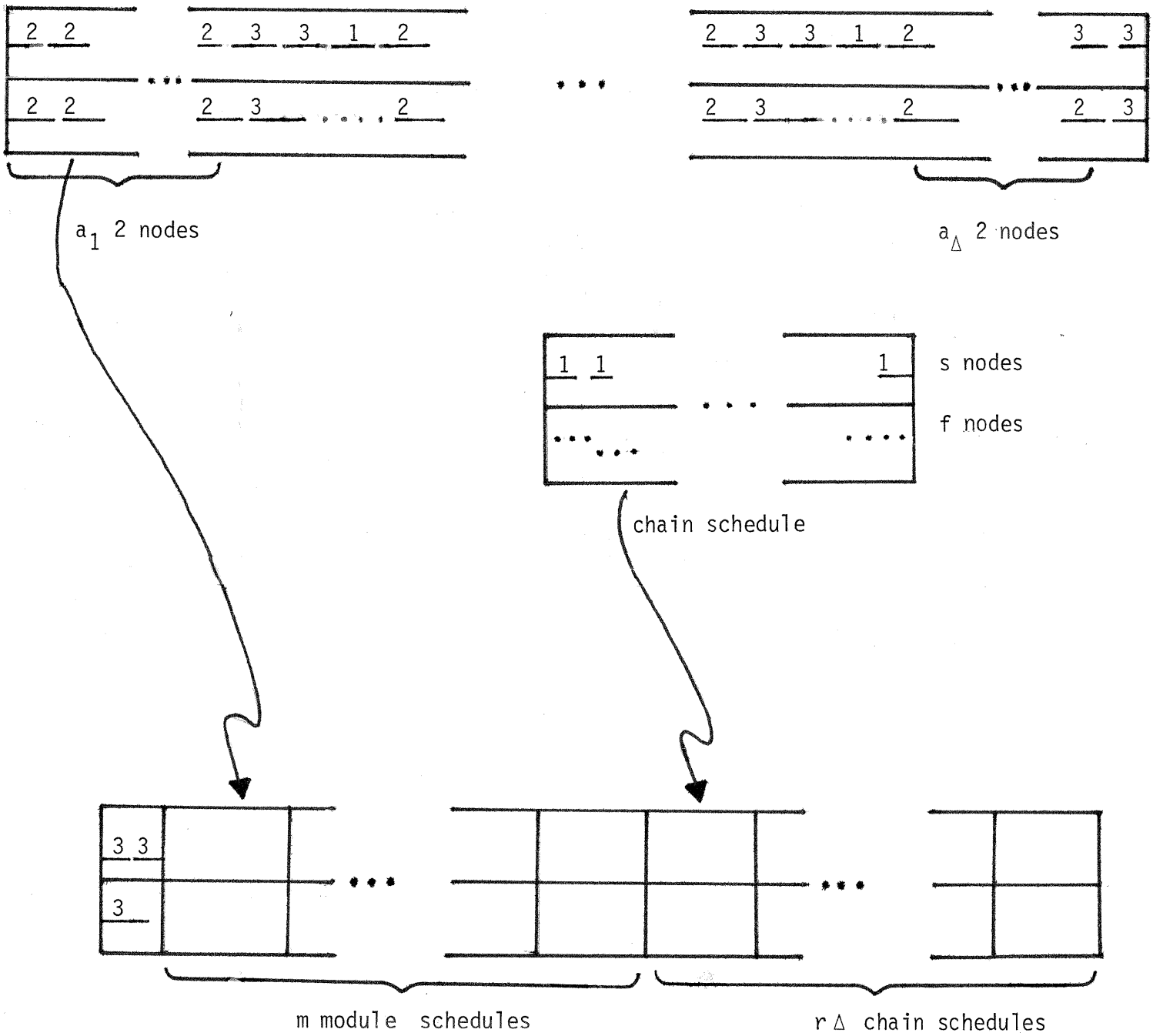
Module for $\Delta > 1$.



module schedule

Figure 4.7

The CAI schedule



module schedule

Figure 4.8

The optimum schedule

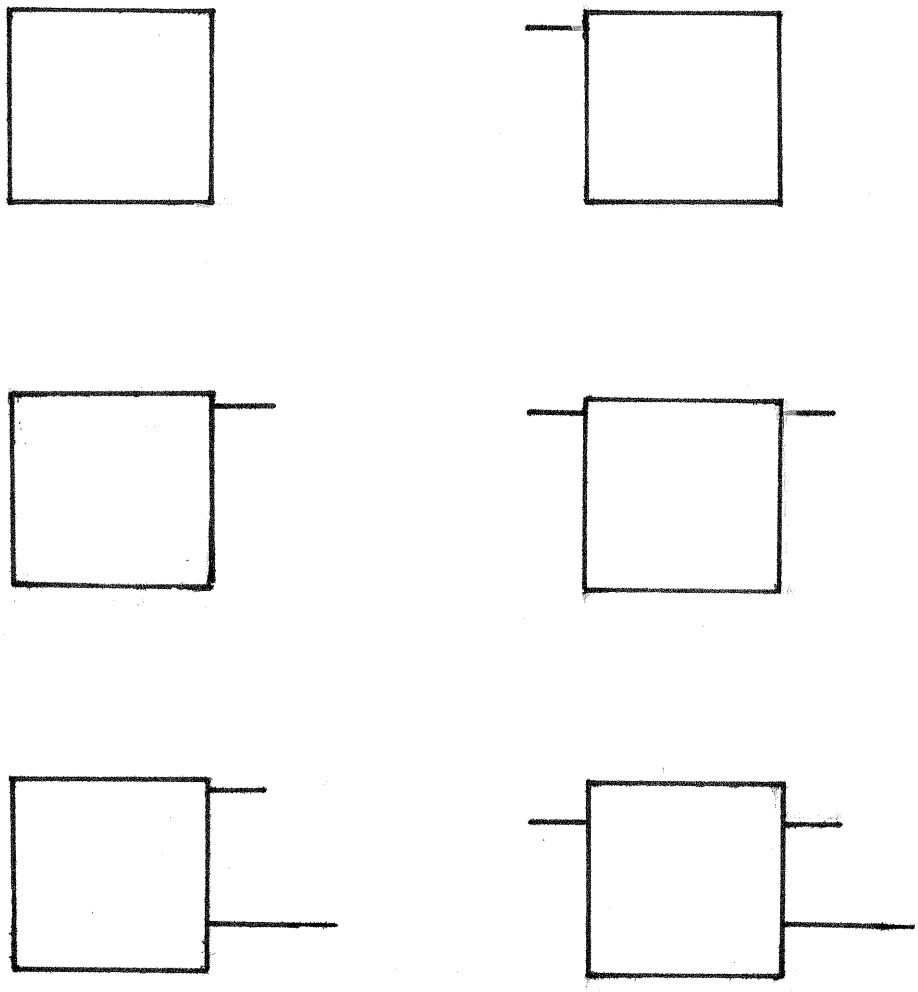


Figure 5.1

The six block shapes for $\frac{f}{s} = \frac{1}{2}$.

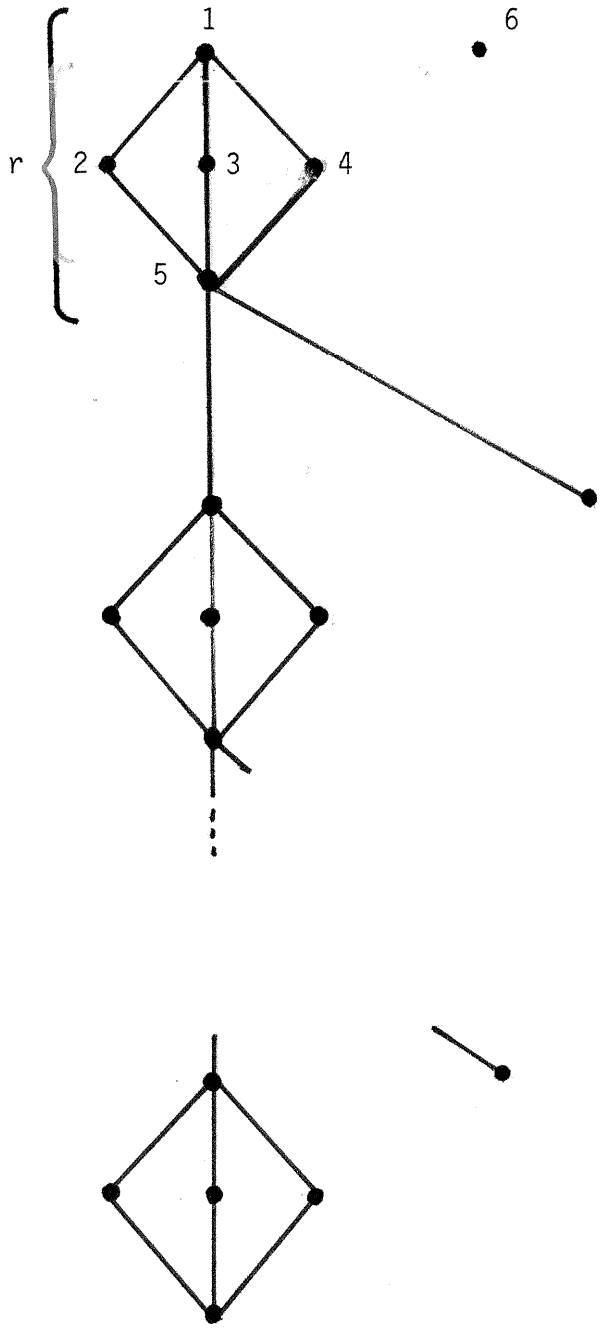


Figure 5.2

Bad dag for the HLF schedule with $\frac{f}{s} = \frac{1}{2}$.

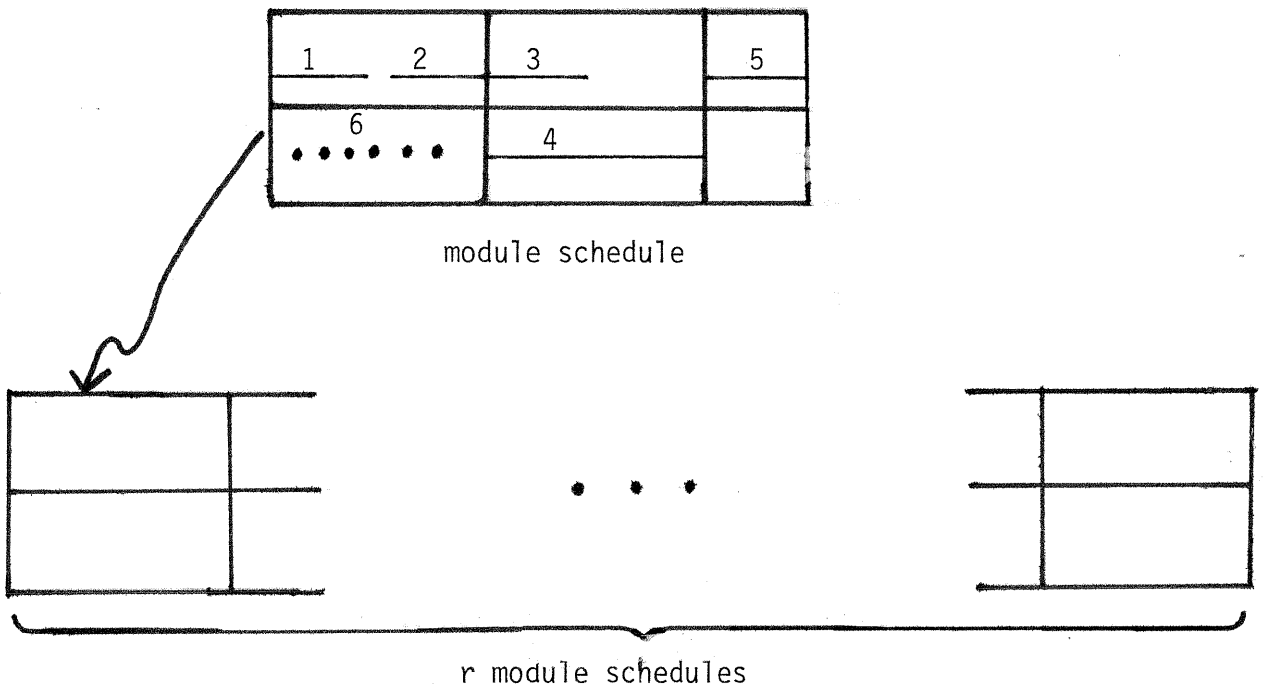


Figure 5.3

The HLF schedule.

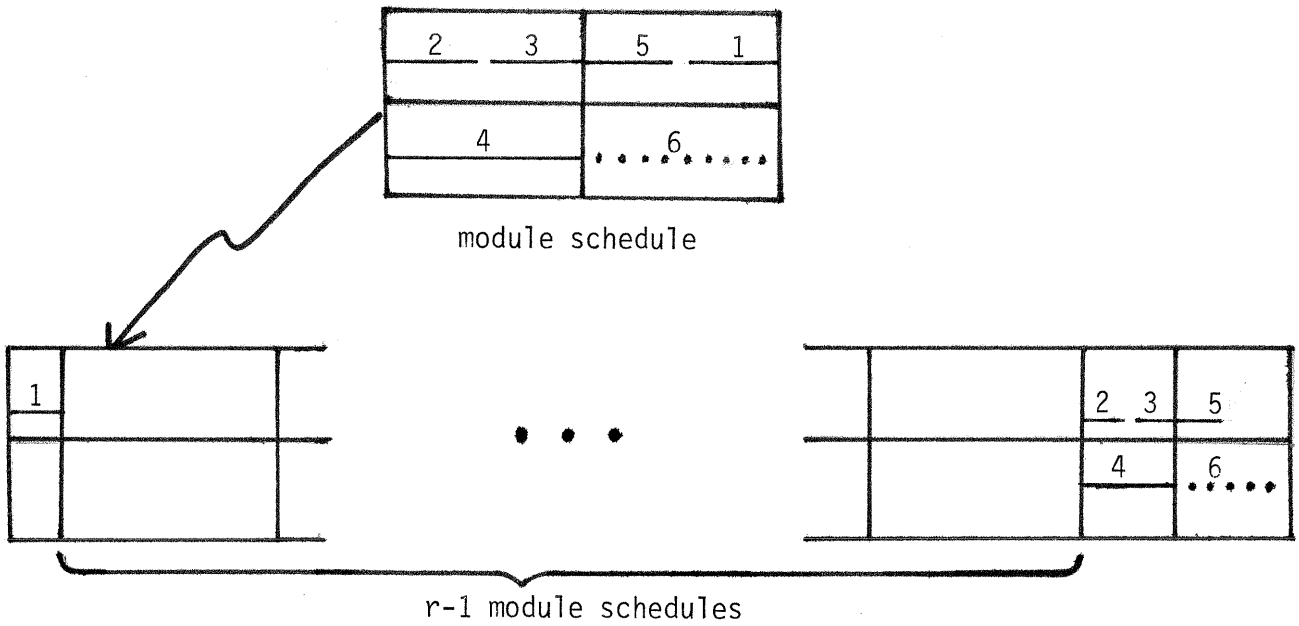


Figure 5.4

The optimum schedule.

	slot 1	2	3	4	5
first	0	—	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$
last	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{1}{3}$	1	0

Table I

Slot contributions to T(X) for $\frac{f}{s} = \frac{2}{3}$.

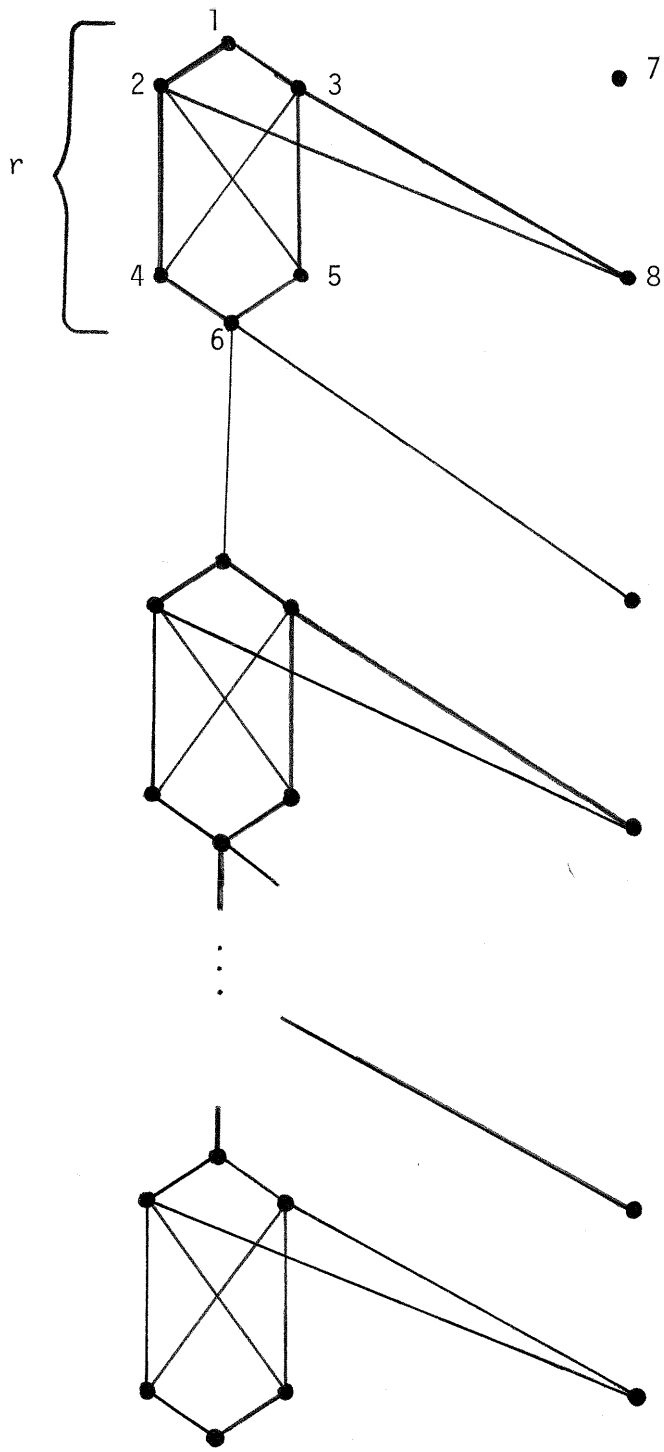


Figure 5.5

Bad dag for the HLF scheudle with $\frac{f}{s} = \frac{2}{3}$.

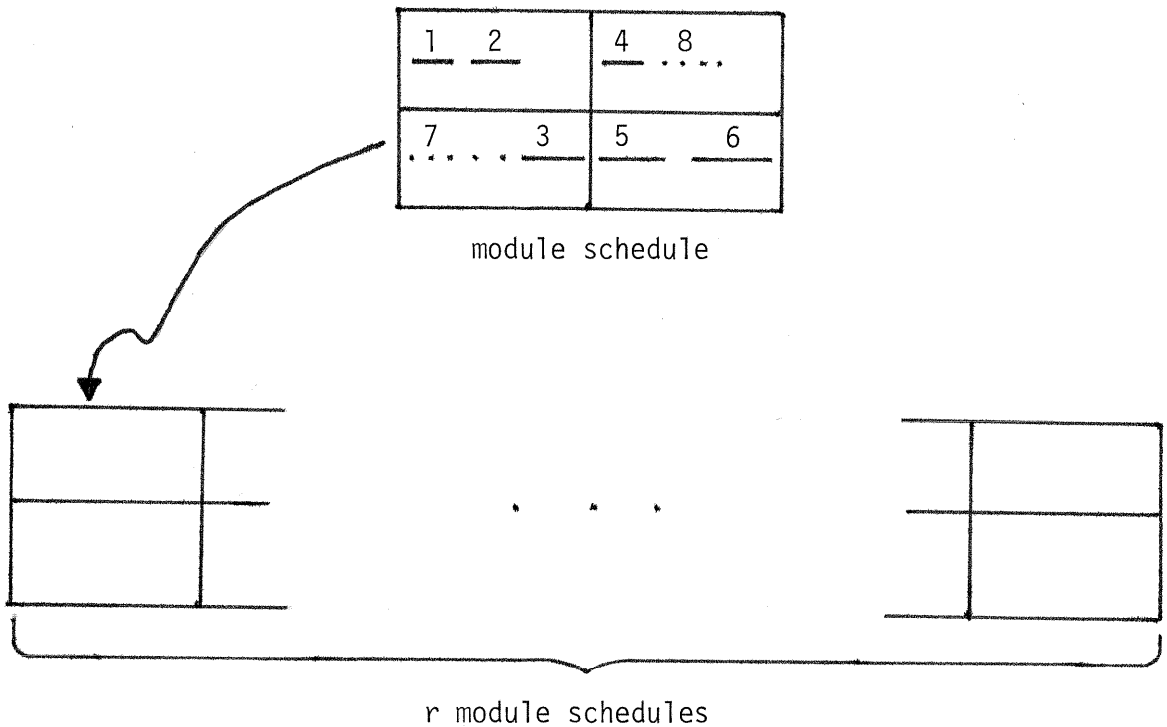


Figure 5.6

The HLF schedule.

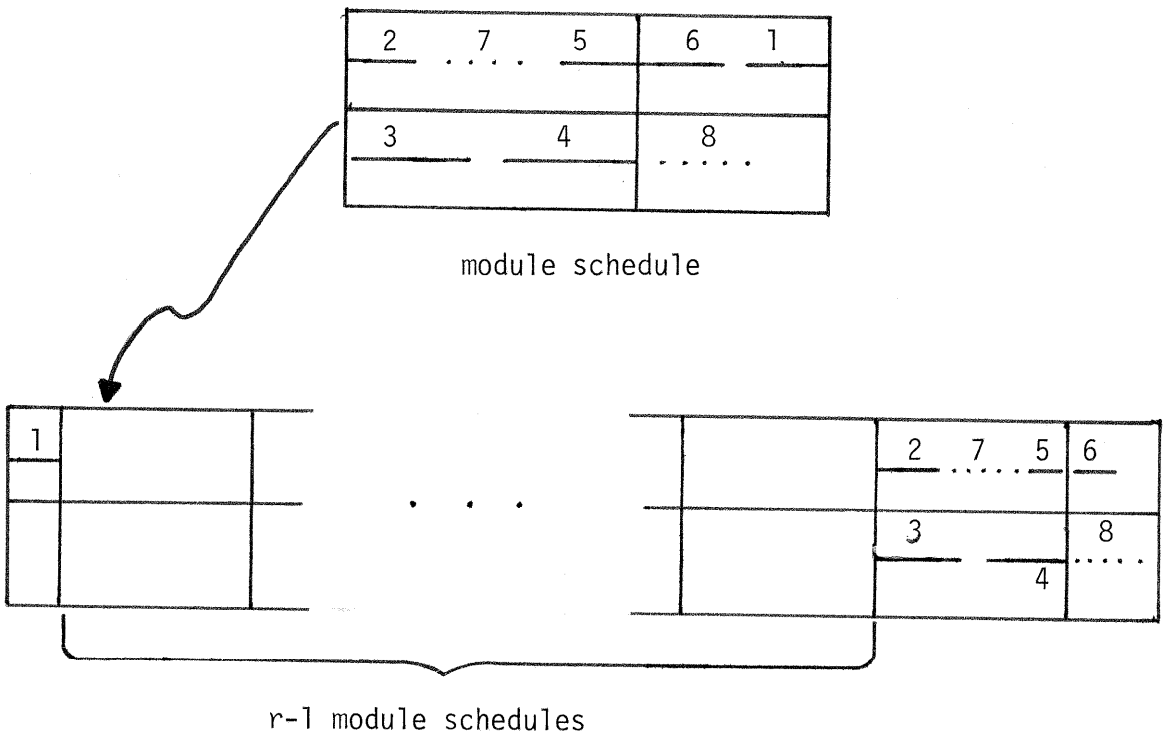


Figure 5.7

The optimum schedule.

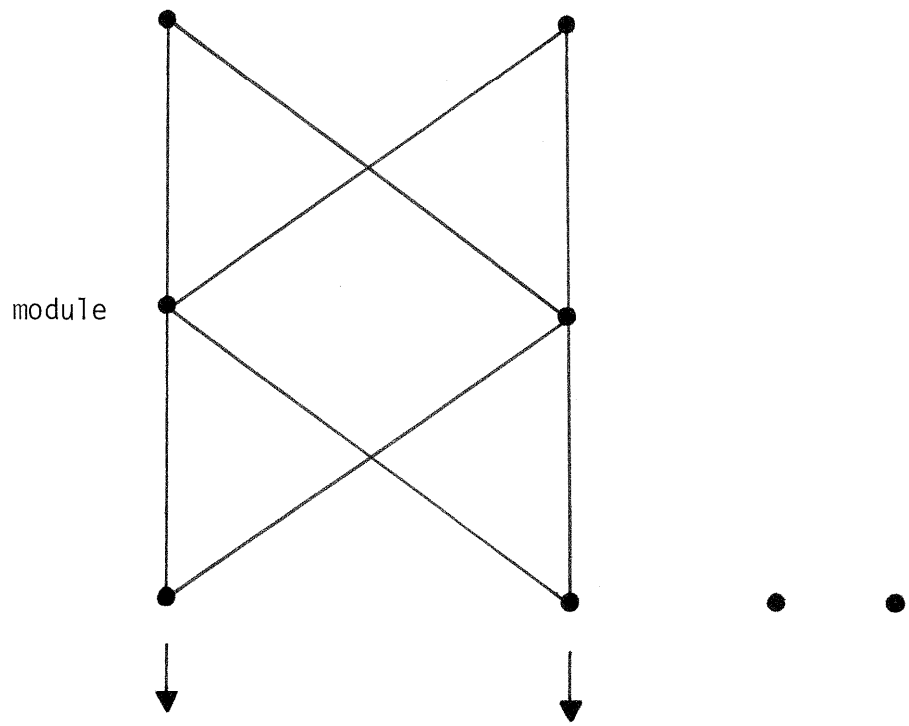


Figure 5.8

Module for HLF with $\frac{f}{s} \geq \frac{4}{5}$.

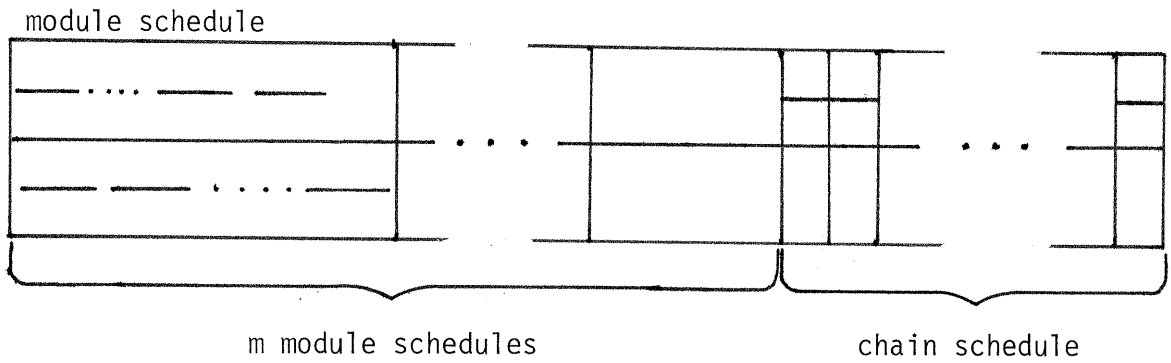


Figure 5.9

The HLF schedule.

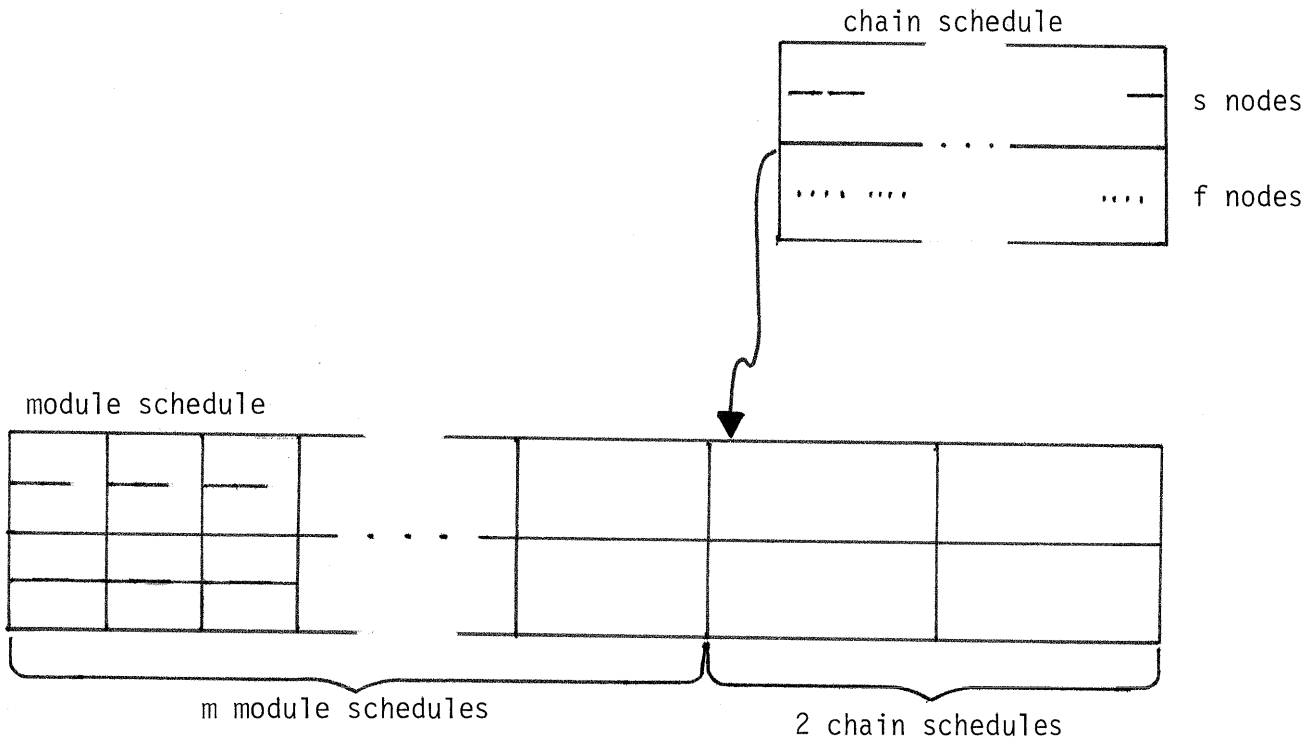


Figure 5.10

The optimum schedule.

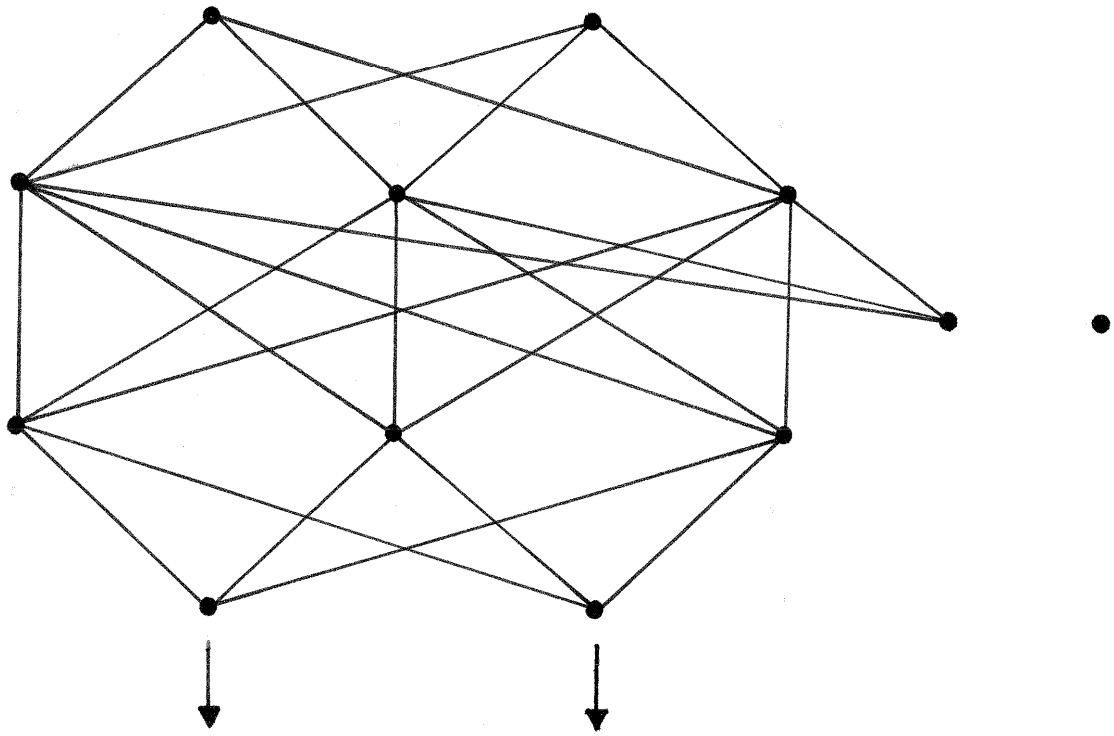


Figure 5.11

Module for HLF schedule with $\frac{f}{s} \geq \frac{3}{4}$.

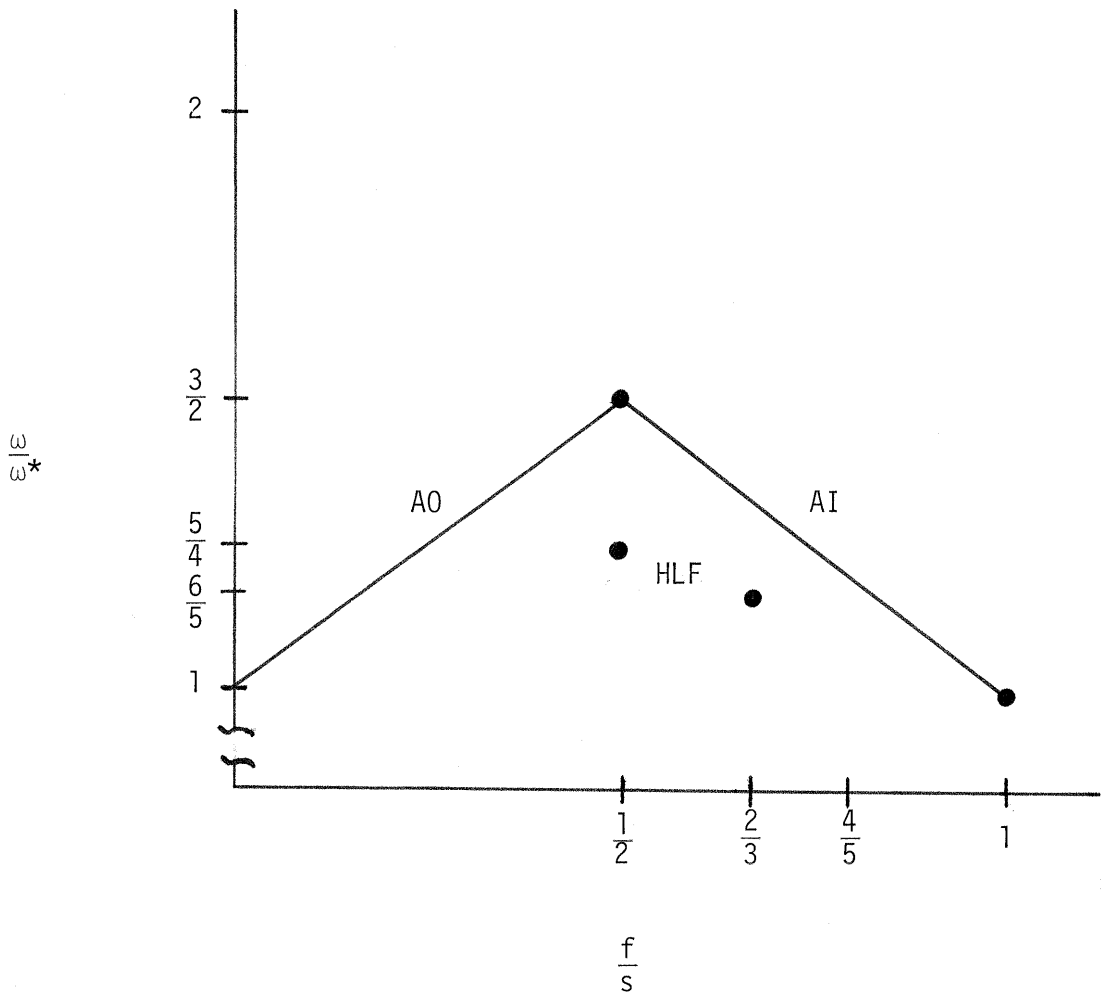


Figure 6.1

Accuracy bounds for approximate schedules.