September 1981 Report. No. STAN-CS-81-880

Well Structured Parallel Programs Are Not Easier to Schedule

by

Ernst W. Mayr

Department of Computer Science

Stanford University
Stanford, CA 94305

Well structured parallel programs are not easier to schedule

by

Ernst Mayr
Department of Computer Science
Stanford University

September 15, 1981

Abstract:

The scheduling problem for unit time task systems with arbitrary precedence constraints is known to be NP-complete. We show that the same is true even if the precedence constraints are restricted to certain subclasses which make the corresponding parallel programs more structured. Among these classes are those derived from hierarchic *cobegin–coend* programming constructs, level graph forests, and the parallel or serial composition of an out-tree and an in-tree. In each case, the completeness proof depends heavily on the number of processors being part of the problem instances.

Key Words and Phrases: Scheduling, parallel programs, NP-complete, well-structured

This work was supported in part by the Deutsche Forschungsgemeinschaft. Grant No. 13 Ma 870/1-1, and in part by Office of Naval Research contract N00014-81-K-0269 and by National Science Foundation grant IST-7921977. Reproduction in whole or in part is permitted for any purpose of the United States government.

1. Introduction

Technological progress has made it possible to design and construct computer architectures with a large number of processors. The intention is to make use of the apparent mutual independence of many activities in (sequential or parallel) programs or task systems, thus achieving shorter overall execution times. Because this hardware - time tradeoff is one of the main justifications to build parallel computers, the scheduling problem, i.e., the problem to assign activities to processors such as to respect their inherent precedence constraints and simultaneously to minimize time, has attracted considerable practical and theoretical interest.

For finite task systems, the scheduling problem could in principle be solved by enumerating all possible schedules and comparing them. However, in general it does not make any sense at all to invest much more time in finding a good schedule than this schedule can then save. Unfortunately, it turned out very soon that already basic variants of the scheduling problem belong to the class of combinatorial optimization problems which are NP-complete and for which only more or less enumerative solution methods of exponential complexity are known [4,9,11]. Efficient algorithms which produce optimal schedules and require only polynomial time are known only for the following few cases:

(i) the scheduling on an arbitrary number of identical processors of a unit-time task system whose precedence constraints form an in-forest (resp., out-forest) [8];

(ii) the scheduling of an arbitrary unit-time task system on two identical processors [3];

(iii) the scheduling on an arbitrary number of identical processors of a unit-time task system whose incomparability graph is chordal [10].

For an extended list of complexity results for scheduling problems, see [2,7].

While in [11] arbitrary precedence constraints are used to show NP-completeness of the scheduling problem of unit-time task systems on an arbitrary number of identical processors, it is the purpose of this paper to show that even well structured precedence constraints are of no help. In particular, we prove that precedence constraints as they derive from parallel constructs in programming languages like *Concurrent Pascal* or *Algol 68* still render the scheduling problem NP-complete. The same holds true for precedence constraints consisting of forests of level graphs or in-trees and one or more out-trees (or, symmetrically, out-trees and one or more in-trees). In the reductions employed for the proofs, the number of available processors plays a significant role, an observation in support of the difficulty (or, maybe, impossibility) to show NP-completeness for a fixed number of processors.

2. Preliminaries and notation

A *task system* is a finite set $T = \{t_1, \ldots, t_v\}$ of *tasks*. For our purposes, all tasks in T require unit time to get executed. *A precedence constraint* on a task system T is a partial order $\prec$ over T. The relation $t_i \prec t_j$ means that the execution of t_j cannot start until t_i is finished. In the sequel, we usually represent $(T, \prec)$ by a directed acyclic graph *(dag)* with node set T and an edge from t_i to t_j iff $t_i \prec t_j$ and there is no t_k such that $t_i \prec t_k \prec t_j$ (i.e., all transitive edges are omitted).

A *schedule* for $(T, \prec)$ on m processors $(m \in N)$ is a mapping s from T *onto* some initial segment $\{1, \ldots, l\}$ of N such that

(i) $t_i \prec t_j \Rightarrow s(t_i) < s(t_j)$ for all $t_i, t_j \in T$;

(ii) $1 \leq |s^{-1}(r)| \leq m$ for all $r \in \{1, \ldots, l\}$.

We say that $t \in T$ is executed at (time-)step $s(t)$, and we call 1 the length of the schedule s. Note that if t_i and t_j are executed at the same time-step they must be incomparable with respect to $\prec$, i.e., neither $t_i \prec t_j$ nor $t_j \prec t_i$ holds.

An *instance* of the *scheduling problem for unit-time task systems on an arbitrary nurnber of (identical) processors* is given by a quadruple $(T, \prec, m, l)$ where

(i) T is a finite task system, without loss of generality denoted by the numerals for 1 through $|T|$;

(ii) $\prec$ is a partial order over T, without loss of generality denoted by a list of the edges in the dag defined by $\prec$ as noted above;

(iii) m and l arc positive integers in radix notation.

Theorem ([11]):

The set

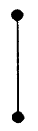
$$\{(T, \prec, m, l);\ \text{there is a schedule for } (T, \prec) \text{ on m processors of length} \leq l\}$$

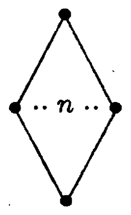
is NP-complete.

In the proof of this theorem in [11], task systems with precedence constraints from a completely general class are used. We want to improve on the above result by showing that precedence constraints of a very natural structure suffice to make the above scheduling problem NP-complete.

Definition:

A dag *(directed acyclic graph)* is a *hierarchic parallel graph* (HPG) if and only if it can be obtained in a finite number of steps from the axiom • by the graph grammar with the following rules:

(i) any node • can be replaced by  ;

(ii) any node • can be replaced by 

, for any n ∈ N.

(Note that all edges point downward; thus all edges entering a node which is being replaced, afterwards enter the topmost node of the replacing graph, and correspondingly, the edges which leave the node then become outgoing edges of the bottommost node.)

We should like to remark that HPG's are closely related to parallel control structures of programs in high level programming languages like *Concurrent Pascal* [1] or *Algol68* [12].

Let H be some directed acyclic graph. Then H defines, in an obvious way, a task system T_H — *the* set of nodes of H — together with a partial order $\prec_H$ — the partial order over T_H generated by the edges of H.

Theorem 1:

The Hierarchic Programs Scheduling Problem
HPSP$=_{\mathrm{def}}$ $\{(H, \mathrm{m}, l)$; H is an HPG, and there is a schedule for $(T_H, \prec_H)$ on m processors of length $\leq l\}$.
is NP-complete.

It is clear that HPSP is in NP. As a matter of fact, it is not hard to test whether a given graph is an HPG, and then HPSP is a restriction of the general scheduling problem of [11] which is in NP. In the next two sections we will show that HPSP is NP-complete. This is achieved by efficiently reducing to HPSP the satisfiability problem 3SAT for sets of clauses with three different literals each [5].

3. A basic task system which is hard to schedule

Let $L = L_1 \wedge \ldots \wedge L_r$ be a propositional formula in three literal conjunctive normal form over the set of variables $\{x_1, \ldots, x_n\}$, i.e., every clause L_i is a disjunction of three different literals (from three different variables) in $\{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$.

We first present a directed acyclic graph H'_L which by itself is not an HPG, but consists of $2n$ HPG components, and which is hard to schedule. For the time being we assume in addition that the number m of processors available in the system is not constant but changes in a predetermined manner at every time-step. We will then show in the next section how to dispose of this assumption, and also, how to transform H'_L into a hierarchic parallel graph.

Let H'_L be defined as shown in Figure 1 where all edges are considered as directed downward. The graph H'_L consists of 2n connected components, one for each literal in $\{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$. Each component has exactly $n + 2r + 2$ levels. Within each component, every level contains either one or two tasks. The i-th component has two tasks exactly on level $\lceil \frac{i+2}{2} \rceil$ and on all levels $n + 2j + 1$ such that $1 \leq j \leq r$ and the literal belonging to component i (which is $x_{\frac{i+1}{2}}$ if i is odd, and $\overline{x}_{\frac{i}{2}}$ if i is even) does not occur in L_j. Also, within each component every task on level i has (directed) edges going to every task on level i + 1, for all $1 \leq i < n + 2r + 3$, and two tasks on any level are always followed by just one task on the next level. Obviously, each component forms an HPG.

The next two lemmas show that there is a schedule for H'_L of length at most $n + 2r + 3$ if and only if the formula L is satisfiable.

Lemma 1:

If L is satisfiable then there is a schedule for $(T_{H'_L}, \prec_{H'_L})$ which in every time-step uses at most as many processors as indicated in Figure 1, and whose length is $n + 2r + 3$.

Proof:

Let $V \subseteq \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$ be the set of literals set true under some fixed truth assignment to the variables $x_1, \ldots, x_n$ that satisfies L, and let $\tilde{V}$ be the set of those components of H'_L corresponding to literals in V. Consider the schedule s which, for $1 \leq j \leq n + 2r + 2$, assigns j to all tasks on level j in components in V, and j + 1 to all other tasks on level j. We claim that s satisfies the condition in the lemma. This is certainly true for time-step 1 because $|\tilde{V}| = n$. In time-step 2, n processors are used to execute the remaining tasks on level 1, and another n + 1 processors are used to execute *all* level 2 tasks of the n components in $\tilde{V}$. This is possible because V contains either x_1 or $\bar{x}_1$ but not both. The same reasoning now applies up through time-step n + 2 after which exactly the n tasks on level $n + 2$ in components in V have been executed. In time-step $n + 3$, n processors are used to execute the remaining tasks on level $n + 2$. Another $2n - 3$, $2n - 2$, or $2n - 1$ processors are used to execute all tasks on level n + 3 of the components in $\tilde{V}$, depending on whether 3, 2, or 1 literals of L_1 are in V. In the first two cases, two resp. one of the available $3n - 1$ processors remain idle at time-step n + 3. As V contains the 'true' literals under a satisfying assignment for L it contains at least one literal of L_1. Thus, $2n - 1$ processors certainly suffice to execute all tasks on level n + 3 of the components in $\tilde{V}$.

In the next time-step, the remaining tasks on level $n + 3$ and the n tasks on level n + 4 of components in V are executed for which at most $2n + n = 3n$ processors are needed. Again we may now observe inductively that after time-step $n + 2r + 2$ all tasks in level $n + 2r + 1$ have been executed and there are exactly those n tasks on level $n + 2r + 2$ left which are not in components in $\tilde{V}$. These n tasks can be scheduled for the n processors available in time-step $n + 2r + 3$. ▮

Lemma 2:

If there is a schedule for $(T_{H'_L}, \prec_{H'_L})$ of length at most $n + 2r + 3$ which at every time-step uses at most the number of processors indicated in Figure 1, then L is satisfiable.

Proof:

First observe that any task on level i+1 can be executed only if all tasks on level i of the same components have been executed before. As there are 2n components each of which has exactly $n + 2r + 2$ levels and as there are only n processors available at the first step, every admissible schedule for $(T_{H'_L}, \prec_{H'_L})$ has a length at least $n + 2r + 3$. Further, as there also are only n processors available in the last step every admissible schedule s for $(T_{H'_L}, \prec_{H'_L})$ of length $n + 2r + 3$ satisfies the following property:

There is a set $\tilde{V}$ of exactly n components of H'_L such that for all j with $1 \leq j \leq n + 2r + 2$,

(I) under s all tasks on level j of components in $\tilde{V}$ are executed at time-step j, and all tasks on level j of components not in $\tilde{V}$ are executed at time-step j + 1.

Let V be the set of literals belonging to the components in $\tilde{V}$. We are now going to show that V defines a satisfying truth assignment for L via x_i :=true iff $x_i \in V$, for $1 \leq i \leq n$. Assume first that there is some minimal i, $1 \leq i \leq n$, such that V contains x_i and $\bar{x}_i$. Then $n + 2$ processors are needed in time-step $i + 1$ to execute all tasks on level i + 1 of the components in $\tilde{V}$, and only n (resp. $n - 1$, if

$i = 1$) processors are left to complete the execution of level i of H'_L. As i was chosen minimal, there are, however, n + 1 (resp. n, if i = 1) tasks left on level i. Hence, V must contain, for every i, either x_i or $\bar{x}_i$.

Next assume that there is some minimal j, $1 \leq j \leq r$, such that V contains no literal occurring in L_j. Then 2n processors are needed in time-step n + 2 j + 1 to execute all tasks on level n + 2 j + 1 of the components in $\tilde{V}$, and only $n - 1$ processors are available to execute the remaining n tasks of level $n + 2j$, in contradiction to property (I). Hence, V must contain, for every j, at least on literal in L_j, i.e., V gives rise, in the way indicated above, to a satisfying truth assignment for L. ▮

In the next section, we shall show how to embed H'_L in a hierarchic parallel graph in such a way that at each time-step exactly the proper number of processors is available for the tasks in the embedded subgraph.

4. MPG's are hard to schedule

In this section, we prove our main

Theorem 2:

HPSP is NP-complete.

Proof:

Let L and H'_L be as in the previous section. We now define the instance (H_L, m, l) of the *Hierarchic Programs Scheduling Problem* with H_L as in Figure 2, $m = 3n + 1$, and $l = n + 2r + 8$.

H_L has $n + 2r + 8$ levels. Note that every directed path from the topmost to the bottommost node of H_L which travels along the left part of H_L in Figure 2 contains $n + 2r + 8$ nodes. As a consequence, every schedule of length $\leq l$ has in fact length $= l$ and must execute these tasks level by level. The construction of H_L thus assures that for all time-steps $i + 3$ with $1 \leq i \leq n + 2r + 3$ the number of processors available for the right part of H_L in Figure 2 (which is H'_L) is exactly the same as for H'_L in the previous section at time-step i.

H_L obviously is an HPG and can be constructed from L in polynomial time (though we omit the details of this construction). This establishes, together with Lemmas 1 and 2, the claim of the theorem. ▮

We should like to mention that HPSP still remains NP-complete if the size of m and 1 in the instances is taken from their unary representation.

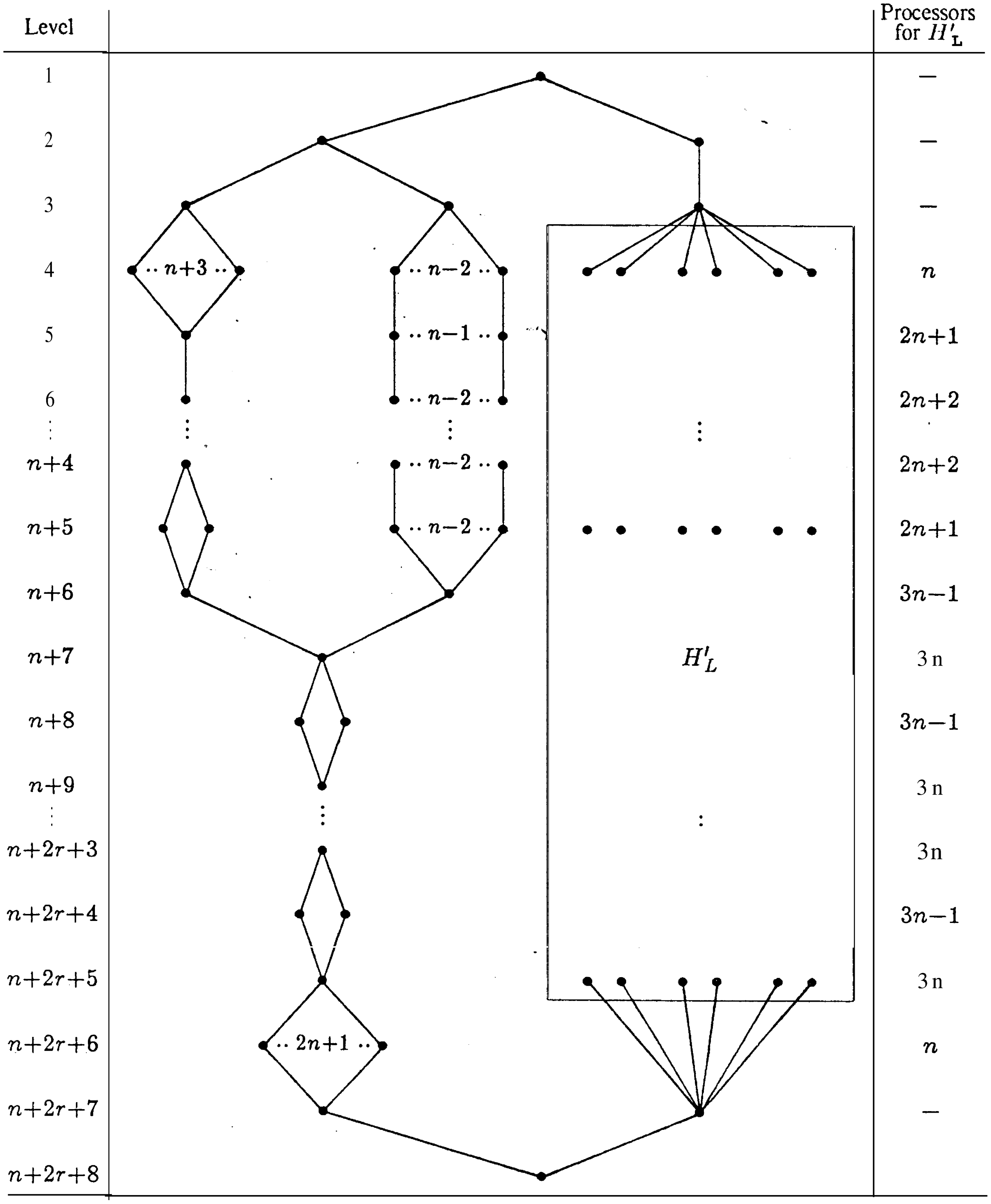


Figure 2

The hierarchic parallel -graph II_L

5. Level graphs and forests

In this section, we extend the result of the previous section to a class of seemingly very simple precedence constraints.

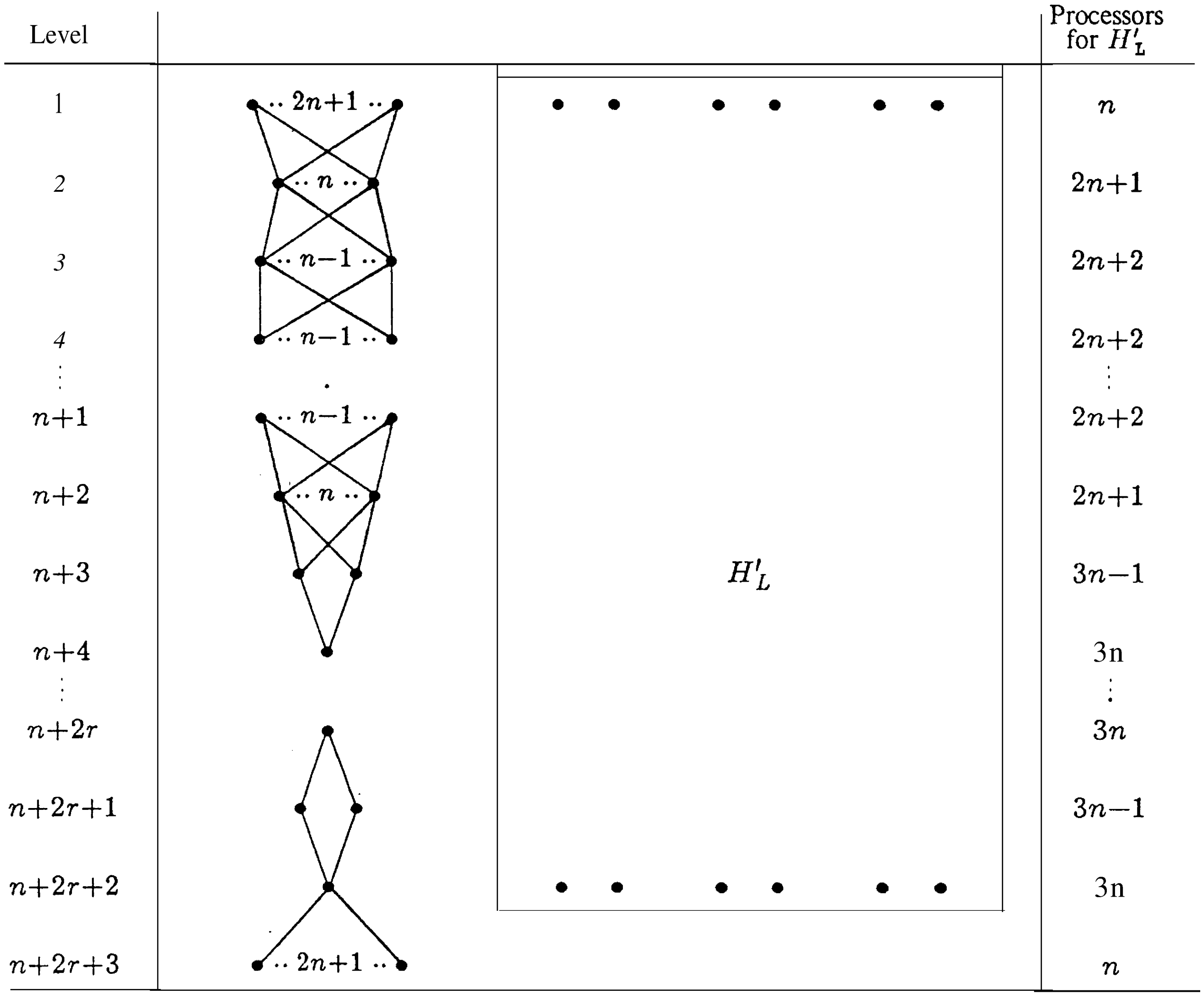


Figure 3

The level forest $\overline{H}_L$ for $L = L_1 \wedge \ldots \wedge L_r$, with $L_1 = \overline{x}_1 \vee x_2 \vee \overline{x}_n$, $L_2 = x_1 \vee x_i \vee x_n$, $L_r = x_2 \vee \overline{x}_i \vee x_n$

Definition:

A directed acyclic graph H is a *level graph* iff its node set T_H can be partitioned into sets $T_1, \ldots, T_s$ such that, for all $1 \leq i < s$, there is an edge from every node in T_i to every node in T_{i+1}.

A level *forest* is a directed acyclic graph consisting of finitely many level graph components.

Note that every component of H'_L in Section 3 is a level graph, and hence, that H'_L is a level forest.

Theorem 3:

The scheduling problem with an arbitrary number of identical processors and unit-time task systems with level forests as precedence constraints is NP-complete.

Proof:

We also use a reduction of 3SAT to the above problem. We noted already that H'_L is a level forest. Now Figure 3 provides an embedding of H'_L into a level forest graph $\overline{H}_L$ which by the same argument as in the proof of Theorem 2 has a schedule on m = 3n + 1 processors which is of length $1 \leq n + 2r + 3$ if and only if L is satisfiable. ∎

6. In-forests with one out-tree

While trees (in-trees or out-trees) were the first class of precedence constraints for which a polynomial time scheduling algorithm was found [8] (a result which easily generalizes to in-forests and out-forests) we shall show in this section that already the simplest combination of the two kinds of trees makes the scheduling problem hard.

Let MF *(mixed forest)* be the class of directed acyclic graphs each of whose components is either an in-tree or an out-tree, and let 2MF be the subclass of MF whose members have at most two components.

Theorem 4:

The scheduling problem with an arbitrary number of identical processors, unit-time task systems, and with elements of 2MF as precedence constraints is NP-complete.

Corollary: --

The scheduling problem for MF-graphs is NP-complete.

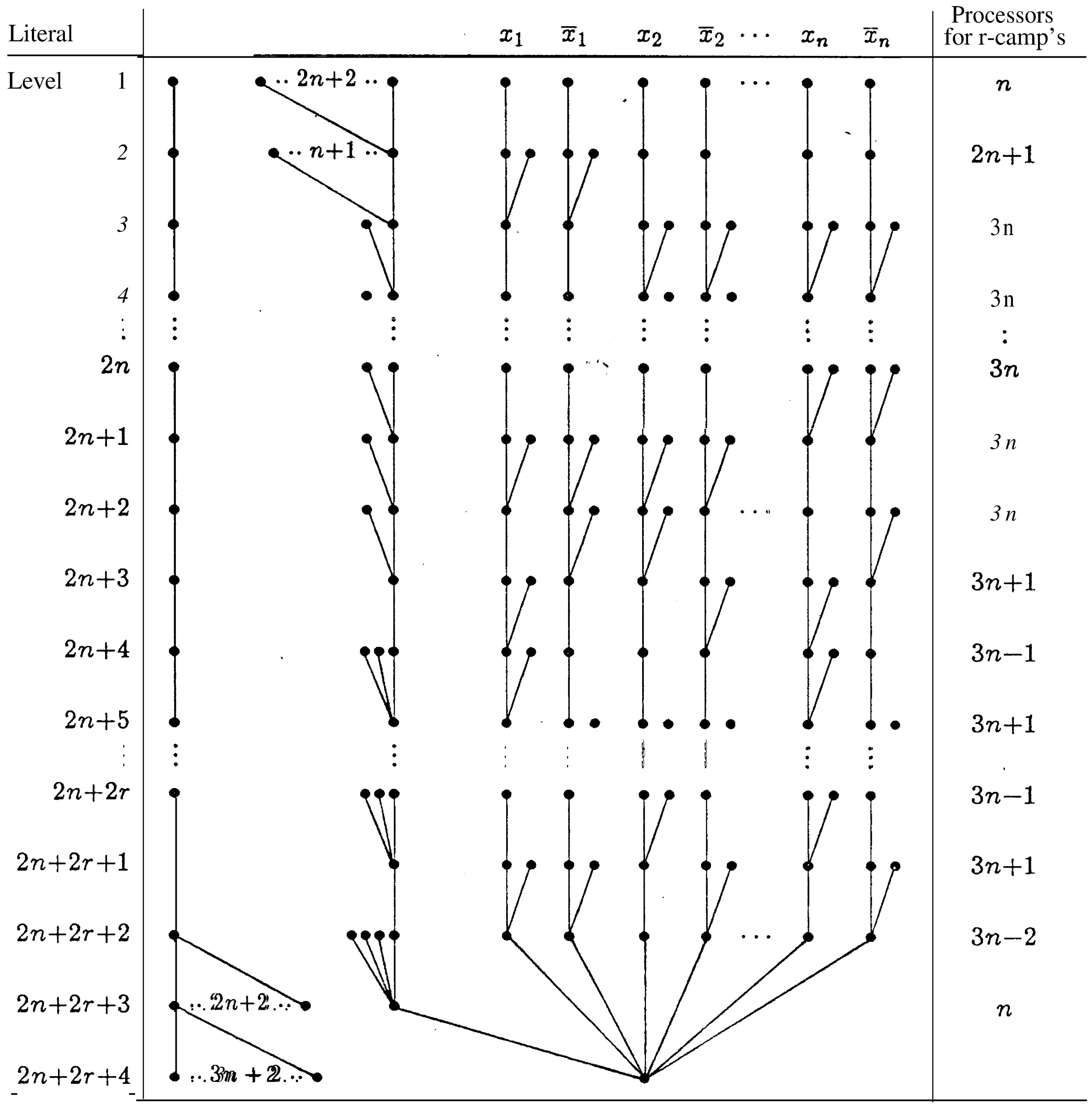
Proof of the Theorem:

A variant of 3SAT which is also NP-complete, is One-in-three-3SAT, i.e., the problem to determine for an arbitrary propositional formula I, in 3-conjunctive normal form whether there is a satisfying truth assignment to the variables in L such that, in every clause L_i, exactly one literal is assigned true [S]. For a given $L = L_i \wedge \ldots \wedge L_r$ with variables $v_1, \ldots,$ v,, we construct $\tilde{H}_L$ as indicated in Figure 4. $\tilde{H}_L$ consists of one in-tree and one out-tree (again all edges are considered directed downward).

Further, let $\tilde{H}_L$ be $\tilde{H}_L$ without its level $2n+2r+4$ nodes and the incident edges. $\tilde{H}_L$ consists of 2n connected components of 2n + $2r$ + 2 levels each (these in-tree components are called *r-components)* and two components of 2n + $2r$ + 3 levels (called *l-components),* one in-tree and one out-tree. Each r-component contains, on every level, either one or two tasks, and the i-th r-component (which belongs to $x_{\frac{i+1}{2}}$ if i is odd, and $\bar{x}_{\frac{i}{2}}$, if i is even) has two tasks on levels 2 $\lceil \frac{i}{2} \rceil$ and 2 j + 1 for all j $\neq \lceil \frac{i}{2} \rceil$, $1 \leq$ j $\leq n$, as well as on level $2n$ + 2 j (resp., 2n + 2j + 1) if the corresponding literal does (resp., does not) occur in L_j.

We now show that there is a schedule for $(T_{\tilde{H}_L}, \prec_{\tilde{H}_L})$ on m = 3n + 3 processors of length at most $2n$ + $2r$ + 3 if and only if I, is in One-in-three-3SAT.

First, let $V \subseteq \{x_1, \bar{x}_1, \ldots,$ x,, $\bar{x}_n\}$ be the set of literals set true under some fixed truth assignment to



Literal
x_1 $\overline{x}_1$ x_2 $\overline{x}_2$ $\cdots$ x_n $\overline{x}_n$
Processors for r-camp's
Level 1
2
3
4
$2n$
$2n+1$
$2n+2$
$2n+3$
$2n+4$
$2n+5$
$2n+2r$
$2n+2r+1$
$2n+2r+2$
$2n+2r+3$
$2n+2r+4$
$\cdot\cdot\ 2n+2\ \cdot\cdot$
$\cdot\cdot\ n+1\ \cdot\cdot$
$\cdot\cdot\ 2n+2\ \cdot\cdot$
$\cdot\cdot\ 3n+2\ \cdot\cdot$
n
$2n+1$
$3n$
$3n$
$3n$
$3n$
$3n$
$3n+1$
$3n-1$
$3n+1$
$3n-1$
$3n+1$
$3n-2$
n

Figure 4

The mixed forest $\tilde{H}_L$ where $L_1 = \overline{x}_1 \vee x_2 \vee \overline{x}_n$, $L_2 = x_1 \vee x_i \vee x_n$, $L_r = x_2 \vee \overline{x}_i \vee x_n$

the variables $x_1, \ldots, x_n$ such that V contains exactly one literal of every clause L_i of L, and let $\tilde{V}$ be the set of those components of $\tilde{H}_L$ determined by the literals in V. Consider the schedule s which, for all $1 \leq j \leq 2n + 2r + 2$, assigns j to all tasks on level j of all the r-components in V and the two l-components, and j + 1 to all tasks on level j of all the other r-components. Level $2n + 2r + 3$ of the I-components is assigned 2n + 27 + 3 under s. We leave it to the reader to verify that, in fact, s is a correct schedule for ($T_{\tilde{\tilde{H}}_L}, \prec_{\tilde{\tilde{H}}_L}$).

For the other direction assume that there is a schedule s for $(T_{\tilde{\tilde{H}}}, \prec_{\tilde{\tilde{H}}_L})$ on m = $3n + 3$ processors, of length $\leq 2n + 2r + 3$. As the Z-components of $\tilde{H}_L$ consist of $2n + 2r + 3$ levels, we must in fact have the length of s equal $2n + 2r + 3$. Now note that also because of the Z-components, in the first and last time-step at most n processors are available for the $2n$ r-components. As these components all have $2n + 2r + 2$ levels the following property must hold:

(II) There is a set $\tilde{V}$ of exactly n r-components in $\tilde{H}_L$ such that, for all j with $1 \leq j \leq 2n + 2r + 2$, under s all tasks on level j of components in $\tilde{V}$ are executed at or prior to time-step j, and in every r-component not in $\tilde{V}$, there is at least one task on level j not yet executed after time-step j (i.e., its value under s is > j). Furthermore, all tasks on level j are executed at the latest at time-step j + 1.

Let V be the set of literals belonging to the r-components in $\tilde{V}$. We shall show that V defines, as in Lemma 2, a truth assignment satisfying L, and also that V contains exactly one literal of every clause L_j of L.

As 2n + 3 processors are needed to execute the level 1 tasks in the two Z-components, and because of property (II), all tasks executed at time-step 1 are of level 1. Let $\tilde{V}$ be the set of those r-components whose level 1 tasks are executed in the first step, and let V be defined as above.

Assume first that there is some minimal i such that V contains either both x_i and $\bar{x}_i$ or none of the two literals. It easily follows from the construction of $\tilde{\tilde{H}}_L$ and property (II) that after time-step 2i − 1

a) all levels j with $1 \leq j < 2i - 1$ are completed,

b) all tasks on level $2i - 1$ of components in $\tilde{V}$ have been executed, and

c) no other tasks have been executed so far.

At time-step $2i$, three (resp., $n + 2$ if i = 1) processors are needed to execute all tasks on level 2i of the Z-components, and $2n - 1$ (resp., n if $i = 1$) processors have to be used to complete level 2i − 1. Therefore, n + 1 processors are available for tasks which are on levels $\geq$ 2i and executable at time-step 2i. If V contains both x_i and $\bar{x}_i$, these n + 1 processors do not suffice to execute the $n + 2$ tasks on level 2i in the components in $\tilde{V}$, contradicting property (II). If V contains neither x_i nor $\bar{x}_i$, then n processors suffice to execute all tasks on level 2i of the components in $\tilde{V}$, and the one remaining processor could be used for any task on some level $\geq$ 2i all of whose predecessors have already been executed. Let us assume instead that one processor is added to the m processors available at time-step 2i + 1. It follows from the construction of $\tilde{H}_L$, however, that in this case 3n + 5, processors are necessary at time-step 2i + 1 to assure property (II). Thus, we again obtain a contradiction, and we conclude that, for all i, V

must contain either x_i or $\overline{x}_i$. Furthermore, a simple counting argument shows that, under s, in time-step j, where $1 \leq j \leq 2n + 1$, only tasks on levels j or $j - 1$ are executed.

Now assume that there is some minimal j, $1 \leq j \leq r$, such that V does not contain exactly one literal of L_j. Then, by an argument analogous to the one just presented, we achieve a contradiction to property (II) at time-step 2n + 2 j if V contains more than one literal of L_j, and at time-step 2n + 2j + 1 if V contains no literal of L_j at all. Hence, V provides a truth assignment for $x_1, \ldots, x_n$ showing that L is in One-in-three-3SAT.

It is now immediate that there is a schedule for $(T_{\tilde{H}_L}, \prec_{\tilde{H}_L})$ on m = 3n + 3 processors of length $l \leq 2n + 2r + 4$ if and only if L is a member of One-in-three-3SAT. Again we leave it to the reader to convince himself that the above reduction can be carried out in polynomial time. ▮

The result stated in Theorem 4 has independently been obtained in [6].

As a fin-ther corollary of Theorem 4 and the construction of $\tilde{H}_L$ we obtain that the scheduling problem for preccdcncc constraints decomposable into an out-tree and an in-tree opposing each other is NP-complete. This follows immediately if we add to $\tilde{H}_L$ one node (at the top) with outgoing edges to all nodes in $\tilde{H}_L$ without predecessor, and a second node (at the bottom) with incoming edges from all nodes of $\tilde{H}_L$ without successor.

7. Conclusion

There are several conclusions we should like to point at which can be drawn from the rcsults presented in the previous sections. The first is that restricting the precedence constraints to be either in-forests or out-forests allows a polynomial scheduling algorithm, but that relinquishing this restriction slightly in either one of a number of directions immediately renders the scheduling problcm NP-complete. We have shown this to hold, for examplc, for the parallel composition of an out-tree and an in-tree as well as for their serial, opposing composition. The latter might seem a little bit surprising in view of the polynomial scheduling algorithms for in- and out-trees, respectively. But it is the intricate interleaving of the two trees on different levels which makes them so difficult to schedule together.

We also showed that restricting the precedence constraints to a subclass which is widely considered well-structured and which forms a subset of the precedence constraints originating from parallel constructs in high level programming languages does not help, this subclass is, in a sense, as hard to schedule as the general class. Again the nicely structured precedence constraints still allow the encoding of an NP-complctc combinatorial problem.

The last observation is that in all the reductions given in the previous sections, the number of parallel processors is part of the problem instance, and that this fact is heavily madc use of. This once more supports the conjecture that it might not be possible to prove the scheduling problem on some fixed number of processors to be NP-complete.

8. Rcfcrcnces

1. BRINCH HANSEN, P.: The architecture of concurrent programs.
Englewood Cliffs, N.J.: Prenticc Hall 1977
2. COFFMAN, E.G. (ED.): Computer and job/shop scheduling theory.
New York: Wiley 1976
3. COFFMAN, E.G., GRAIIAM, R.L.: Optimal scheduling for two-processor systems.
Acta Informatica 1 (1972), pp. 200-213
4. COOK, S.A.: The complexity of theorem proving procedures.
Proc. 3rd Ann. ACM STOC (1971), pp. 151-158
5. GAREY, M.R., JOHNSON, D.S.: Computers and intractability: a guide to the theory of NP-completeness.
San Francisco: W.H. Freeman and Company 1979
6. GAREY, M.R., ET AL.: Scheduling opposing forests.
TM-81-11216-44, Bell Labs, Murray Hill, N.J. (1981)
7. GRAHAM, R.L., ET AL.: Optimization and approximation in deterministic sequencing and scheduling: a survey.
In: HAMMER, P.L., ET AL. (EDS.): Annals of Discrete Mathematics 5. Amsterdam-New York-Oxford: North-Holland Publishing Company (1979), pp. 287-326
8. Hu, T.C.: Parallel sequencing and assembly line problems.
Operations Research 9 (1961), pp. 841-848
9. KARP, R.M. : Reducibility among combinatorial problems.
In: MILLER, R.E., THATCHER, J.M. (EDS.): Complexity of computer computations. Ncw York: Plenum (1975), pp. 85-103
10. PAPADIMITRIOU, CH., YANNAKAKIS, M.: Scheduling interval-ordered tasks.
SIAM J. Comput. 8 (1979), pp. 405-409
11. ULLMA N, J.D.: NP-complete scheduling problems.
J. Comput. System Sci. 10 (1975), pp. 384-393
12. WIJNCAARDEN, A. VAN, ET AL.: Revised report on the algorithmic language ALGOL 68.
Berlin-Hcidelbcrg-New York: Springer-Vcrlag 1976