JOURNAL OF COMPUTER AND SYSTEM SCIENCES 10, 384–393 (1975)

NP-Complete Scheduling Problems*

J. D. ULLMAN

Department of Electrical Engineering, Princeton University,[†] *Princeton, New Jersey 08540*

Received May 16, 1973

We show that the problem of finding an optimal schedule for a set of jobs is *NP*-complete even in the following two restricted cases.

(1) All jobs require one time unit.

(2) All jobs require one or two time units, and there are only two processor resolving (in the negative a conjecture of R. L. Graham, *Proc. SJCC*, 1972, pp. 205–218).

As a consequence, the general preemptive scheduling problem is also *NP*-complete. These results are tantamount to showing that the scheduling problems mentioned are intractable.

I. INTRODUCTION

The scheduling problem is the following. We are given

(1) a set $S = \{J_1, ..., J_n\}$ of *jobs*,

(2) a partial order $\prec$ on S,

(3) a *weighting function* W from S to the positive integers, giving the number of time units required by each job, and

(4) a number of processors, k.

Informally, we may "execute" up to k jobs at each time unit $t = 0, 1, ..., t_{\max}$. If job J is first executed at time t, then we assume it is executed at times $t, t + 1, ..., t + W(J) - 1$, and at no other times. The *scheduling problem* is to minimize $t_{\max}$ under the constraint that if $J \prec J'$, then J' does not begin execution until at least $W(J)$ time units after J begins execution. The reader is referred to [1] for a survey of results on the scheduling problem.

* Work supported by NSF Grants GJ 474 and GJ 35570. A preliminary version of this paper appeared in *Operating Systems Review*, October, 1973.

† Part of this work was done while the author was on leave at the University of California, Berkeley.

Following [2, 3], the class of problems known as *NP-complete* problems has received heavy attention recently. A survey of results in this area can be found in [4], and some papers discussing problems closely related to scheduling are [5–7]. Informally, a problem is in $\mathcal{NP}$ if it is accepted by a nondeterministic Turing machine in polynomial time. Many apparently hard combinatorial problem such as the "traveling salesman" problem are in $\mathcal{NP}$. An *NP-complete* problem P_0 is one which enjoys the following property. If P_0 has a deterministic polynomial time algorithm, then so does every problem in $\mathcal{NP}$. Since for no *NP*-complete problem has a less than exponential time algorithm been found, showing a given problem to be *NP*-complete is tantamount to a proof that it has no polynomial time algorithm, and in fact, likely requires exponential time. For rigorous statements of the above, see [2–4].

Since a Turing machine may only accept or reject a tape, problems in $\mathcal{NP}$ are normally couched in a way that requires a yes–no answer. We may therefore express the time scheduling problem as follows.

(P1): *General scheduling problem.* Given a set S of n jobs, a partial order $\prec$ on S, a weighting function W, a number of processors k and a time limit t, does there exist a total function f from S to $\{0, 1,..., t - 1\}$ such that

(i) if $J \prec J'$, then $f(J) + W(J) \leqslant f(J')$,

(ii) for each J in S, $f(J) + W(J) \leqslant t$, and

(iii) for each i, $0 \leqslant i < t$, there are at most k values of J for which $f(J) \leqslant i < f(J) + W(J)$?

It is easy to see that P1 is in $\mathcal{NP}$ if the encoding is such that at least n symbols are required to represent any n job problem. A Turing machine can guess the assignment f of jobs to starting times and check that the assignment satisfies (i)–(iii) above. In [3] it is shown that (P1) is *NP*-complete by proving that every problem in $\mathcal{NP}$ can be transformed[1] to (P1) in polynomial time. In fact, the special case of two processors and empty relation $\prec$ is *NP*-complete. Here, we show two other special cases of (P1) to be *NP*-complete.

(P2): *Single execution time scheduling.* Here, we restrict (P1) by requiring $W(J) = 1$ for all jobs J.

[1] The notion of polynomial time transformation is central to the theory of *NP*-complete problems. We say problem P *polynomially transforms* to problem P' if there is a function f which takes an instance x of P and produces deterministically and in polynomial time an instance $y = f(x)$ of P', such that the answer to y is "yes" if and only if the answer to x is "yes." Again, a more formal expression of this notion can be found in [3, 4].

It is important to observe that if P polynomially transforms to P', and P is *NP*-complete, then so is P'. This method is the principal one which has been used to show new problems to be *NP*-complete since Cook [2] showed that every problem in $\mathcal{NP}$ polynomially transforms to a particular problem (the tautology problem).

(P3): *Two processor, one or two time unit scheduling.* Here, restrict the number of processors to two and the range of the weighting function to $\{1, 2\}$.

The question of whether (P2) and (P3) are *NP*-complete is of some interest, since algorithms [8, 9] have been recently discovered that solve (P2) for the two processor case (i.e., the intersection of (P2) and (P3)) deterministically in polynomial time. These results suggest the possibility that (P2) and/or (P3) themselves have deterministic polynomial time algorithms. Our first result shows that such cannot be the case. We do not, however, rule out the possibility that for each number of processors k there is a polynomial algorithm to solve (P2) for this fixed k.

II. Single Execution Time Scheduling

We begin by introducing a slightly more complex problem which can be polynomially transformed to (P2). We shall later show this new problem (P4) to be *NP*-complete, thus proving the *NP*-completeness of (P2).

(P4): *Single execution time scheduling with variable number of processors.* Given a set S of n jobs, a relation $\prec$ on S, a time limit t, and a sequence of integers $c_0, c_1, \ldots, c_{t-1}$, where $\sum_{i=0}^{t-1} c_i = n$, does there exist a function f from S to $\{0, 1, \ldots, t-1\}$ such that

(i) $f^{-1}(i)$ has exactly c_i members, and

(ii) if $J \prec J'$, then $f(J) < f(J')$?

In what follows, we assume problems are encoded in such a way that the length of a string representing an n job problem is a polynomial in n and at least $O(n)$. Moreover, we assume that the encoding is sufficiently natural that the jobs, the relation $\prec$ and so on can be determined from the string easily, certainly in polynomial time.

Lemma 1. (P4) *polynomially transforms to* (P2).

Proof. Given an instance of (P4), introduce new jobs I_{ij} for $0 \leqslant i < t$ and $0 \leqslant j \leqslant n - c_i$. Let the old jobs be related by $\prec$ as before and let $I_{ij} < I_{i+1,k}$ for $0 \leqslant i < t - 1$ and arbitrary j and k. If we choose $n + 1$ processors and time limit t, we have an instance of (P2). Since in any solution, exactly $n + 1 - c_i$ of the new jobs must be executed at the ith time unit, the instance of (P2) will have a solution if and only if the original instance of (P4) does.

Clearly the time needed to construct the instance of (P2) is at most quadratic in the length of the representation for the instances of (P4). Thus, (P4) has been polynomially transformed to (P2). ∎

We next introduce the 3-satisfiability problem, shown to be *NP*-complete in [2]. The problem can be expressed as the question of whether a Boolean expression which is the product of sums of three literals has an assignment to its variables which makes the expression true. The following is an interpretation of the problem due to [3].

3-satisfiability. Given a set of "variables" x_i, $1 \leqslant i \leqslant m$, and a collection of sets $D_1, \ldots, D_n$, where $m \leqslant 3n$, such that each D_j consists of exactly three of the elements x_i or $\bar{x}_i$ (called *literals*), does there exist a map f from $\{1, 2, \ldots, m\}$ to $\{\textbf{true}, \textbf{false}\}$ such that for each j, $1 \leqslant j \leqslant n$, either some $x_i \in D_j$ and $f(i) = \textbf{true}$, or $\bar{x}_i \in D_j$ and $f(i) = \textbf{false}$?

We assume the encoding of the 3-satisfiability problem is at least $O(n)$ in length and at most polynomial in n.

LEMMA 2. *The 3-satisfiability problem polynomially transforms to* (P4).

Proof. Given an instance of 3-satisfiability as above, we construct the following instance of (P4) (the single execution time scheduling problem with a variable number of processors). The jobs we shall denote:

$$x_{ij} \text{ and } \bar{x}_{ij} \text{ for } 1 \leqslant i \leqslant m \text{ and } 0 \leqslant j \leqslant m,$$
$$y_i \text{ and } \bar{y}_i \text{ for } 1 \leqslant i \leqslant m,$$
$$D_{ij} \text{ for } 1 \leqslant i \leqslant n \text{ and } 1 \leqslant j \leqslant 7.$$

The relation $\prec$ is given by:

(i) $x_{ij} \prec x_{i,j+1}$ and $\bar{x}_{ij} \prec \bar{x}_{i,j+1}$, for $1 \leqslant i \leqslant m$ and $0 \leqslant j < m$;

(ii) $x_{i,i-1} \prec y_i$ and $\bar{x}_{i,i-1} \prec \bar{y}_i$ for $1 \leqslant i \leqslant m$;

(iii) let us consider D_{ij}, where $a_1 a_2 a_3$ is the binary representation of j. (Note that the case $a_1 = a_2 = a_3 = 0$ cannot occur.) Let D_i consist of literals z_{k_1}, z_{k_2}, z_{k_3}, where each z independently stands for x or $\bar{x}$, in a fixed order. Then for $1 \leqslant p \leqslant 3$, if $a_p = 1$, we have $z_{k_p,m} \prec D_{ij}$. If $a_p = 0$, we have $\bar{z}_{k_p,m} \prec D_{ij}$, where $\bar{z}$ stands for $\bar{x}$ or x, should z be x or $\bar{x}$, respectively.

The time limit is $m + 3$, and constants c_i, $0 \leqslant i \leqslant m + 2$ are:

$$\begin{aligned} c_0 &= m, \\ c_1 &= 2m + 1, \\ c_i &= 2m + 2, \qquad \text{for} \quad 2 \leqslant i \leqslant m, \\ c_{m+1} &= n + m + 1, \\ c_{m+2} &= 6n. \end{aligned}$$

We shall show that the above instance of (P4) has a solution if and only if the given instance of 3-satisfiability does. The intuitive idea behind the proof is that we may imagine x_i (or $\bar{x}_i$) to be **true** if and only if x_{i0} (or $\bar{x}_{i0}$, respectively) is executed at time 0. We shall see that the presence of the y's and $\bar{y}$'s forces exactly one of x_{i0} and $\bar{x}_{i0}$ to be executed at time 0 and the other to be executed at time 1. Then, the requirement that $n + m + 1$ jobs be executed at time $m + 1$ is tantamount to the requirement that for each i, there is one j such that D_{ij} may be executed at that time (there cannot be more than one). But this condition is equivalent to saying that the sum of terms which D_i represents has truth value **true** when those of the x_i's and $\bar{x}_i$'s which were executed at time 0 are given the value **true**.

We first show that in any solution to the instance of (P4), we may not execute both x_{i0} and $\bar{x}_{i0}$ at time 0 for any i. Suppose we did. Then since $c_0 = m$, there would be some j such that neither x_{j0} nor $\bar{x}_{j0}$ was executed at time 0. Then neither y_j nor $\bar{y}_j$ would be executed at or before time j, as, for example, y_j must be preceded by $x_{j0}, x_{j1}, \ldots, x_{j,j-1}$, each executed strictly before the next in the sequence. The total number of jobs which could be executed at or before time j is thus seen to be

(1) at most $m(2j + 1)$ of the x's and $\bar{x}$'s, that is, $z_{0i}, z_{i1}, \ldots, z_{ij}$ if z_{i0} was executed at time 0 and $z_{i0}, z_{i1}, \ldots, z_{i,j-1}$ if not (again, z stands for x or $\bar{x}$), and

(2) at most $2(j - 1)$ of the y's, specifically $y_1, \bar{y}_1, y_2, \bar{y}_2, \ldots, y_{j-1}, \bar{y}_{j-1}$.

The total number of jobs executable by time j is thus at most $2mj + 2j + m - 2$. However, for $1 \leqslant j \leqslant m$,

$$\sum_{i=0}^{j} c_i = 3m + 1 + (j - 1)(2m + 2) = 2mj + 2j + m - 1.$$

We may conclude that in any solution to this instance of (P4), exactly one of x_{i0} and $\bar{x}_{i0}$ is executed at time 0. Moreover, we can determine the exact jobs which are executed at each time between 1 and m, given which of x_{i0} and $\bar{x}_{i0}$ is executed at time 0. That is, at time t we must execute z_{it} if z_{i0} was executed at time 0 and $z_{i,t-1}$ if not. Moreover, we must execute y_t (respectively, $\bar{y}_t$) at time t if x_{t0} (respectively, $\bar{x}_{t0}$) was executed at time 0 and execute y_{t-1} (respectively, $\bar{y}_{t-1}$) at time t if x_{t0} (respectively, $\bar{x}_{t0}$) was executed at time 1.

At time $m + 1$ we can execute the m remaining x's and $\bar{x}$'s and the one remaining y or $\bar{y}$. Since $c_{m+1} = m + n + 1$, we must be able to execute n of the D's if we are to have a solution. We observe that for each pair D_{ij} and $D_{ij'}$, $j \neq j'$, there is at least one k such that x_{km} precedes D_{ij} and $\bar{x}_{km}$ precedes $D_{ij'}$, or vice versa. Since we have already proven that exactly one of x_{km} and $\bar{x}_{km}$ can be executed by time m, it follows that for each i, at most one of $D_{i1}, D_{i2}, \ldots, D_{i7}$ can be executed at time $m + 1$.

Moreover, if we assign the truth value **true** to x_k (respectively, $\bar{x}_k$) if and only if x_{k0} (respectively, $\bar{x}_{k0}$) was executed at time 0, then there will be one of $D_{i1}, D_{i2}, \ldots, D_{i7}$

executable at time $m + 1$ if and only if D_i takes the value **true** under this assignment of values to the variables. We conclude that a solution to the instance of (P4) exists if and only if the original product of sums is satisfiable. ▮

EXAMPLE. Let us consider the Boolean expression $(x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + \bar{x}_3 + x_4)$. That is, $n = 2$, $m = 4$, $D_1 = \{x_1, \bar{x}_2, x_3\}$, $D_2 = \{\bar{x}_1, \bar{x}_3, x_4\}$, and $c_0, ..., c_6$ are, respectively, 4, 9, 10, 10, 10, 7, 12. One possible solution to the corresponding instance of (P4), which is based on the assignment of **true** to x_1, x_2, $\bar{x}_3$ and x_4 is shown in Fig. 1. Lines represent the relation $<$, except at time 6, when the many lines necessary are omitted. ▮

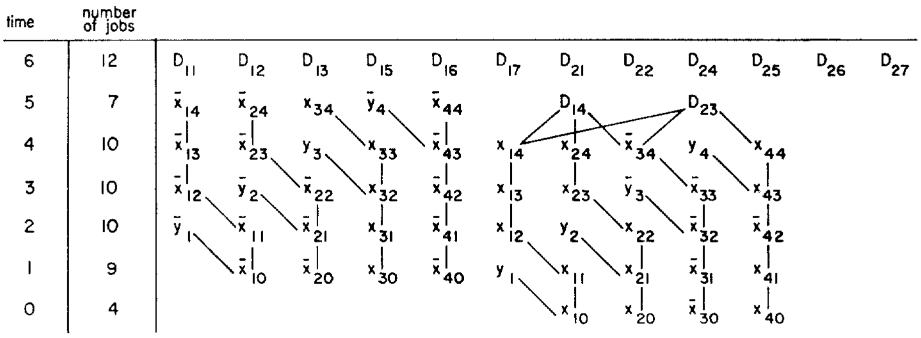

FIGURE 1

We may put Lemmas 1 and 2 together to obtain the following.

THEOREM 1. *The single execution time scheduling problem* (P2) *is NP-complete.*

Proof. By Lemmas 1 and 2 and the *NP*-completeness of 3-satisfiability [2]. ▮

We shall have use for a slight strengthening of Theorem 1, a result which has effectively been proven already. Define (P5) to be (P2) with the restriction that $n = kt$, that is, all processors must be in use at all times up to the limit for a solution to exist. Then in Lemma 1, we actually showed (P4) polynomially transforms to (P5). As a consequence, we may state the following.

THEOREM 2. *Single execution time scheduling with* $n = kt$ (P5) *is NP-complete.*

Another consequence of Theorem 1 is that the *preemptive scheduling problem* is *NP*-complete.[2] In preemptive scheduling, we are not required to finish a job as soon as possible once it has begun. A formal definition of the problem is as follows.

[2] The author is indebted to Peter Denning for making this observation.

(P6): *Preemptive scheduling.* Given a set S of n jobs, a partial order $\prec$ on S, a weighting function W, a number of processors k, and a time limit t, does there exist a total function f from S to subsets of $\{0, 1,\ldots, t-1\}$, such that

(i) $f(J)$ has $W(J)$ members for all J in S,

(ii) if $J \prec J'$, i is in $f(J)$ and i' in $f(J')$, then $i < i'$, and

(iii) for each i there are at most k values of J for which $f(J)$ contains i?

THEOREM 3. *The preemptive scheduling problem* (P6) *is NP-complete.*

Proof. (P2) is just (P6) with $W(J)$ restricted to be 1 for all J. Thus, (P2) polynomially transforms to (P6) trivially, and the result follows from Theorem 1. ▌

III. TWO PROCESSOR SCHEDULING WITH WEIGHTS OF ONE AND TWO

We now show that (P3) is also NP-complete.

LEMMA 4. (P5) *polynomially transforms to* (P3).

Proof. Suppose we are given an instance of (P5) with time limit t, number of processors k, and partial order $\prec$ on a set S of kt jobs. We construct the following instance of (P3). The jobs are:

(1) X_i for $0 \leqslant i < T$, where $T = (4k+1)t$;

(2) Y_{ij} for $0 \leqslant i < t$ and $0 \leqslant j \leqslant k$; and

(3) J and J' for each J in S.

The weighting function W is $W(J) = 2$ for J in S and $W(Z) = 1$ for all other jobs Z.

The relation $\sqsubset$ is specified by:

(i) $X_i \sqsubset X_{i+1}$ for $0 \leqslant i < T-1$,

(ii) $X_u \sqsubset Y_{ij} \sqsubset X_{u+2}$, where $u = (4k+1)i + 2j - 1$ (in the case $i = j = 0$, where $u = -1$, we have only the relation $Y_{00} \sqsubset X_1$),

(iii) $J' \sqsubset J$ for all J in S, and

(iv) $K \sqsubset J'$ for K and J in S if and only if $K \prec J$.

The time limit is T, defined in (1) above.

We first observe by (i) that one processor must be devoted to processing an X at each time unit if the time limit is to be met. We may assume the first processor is used exclusively for this purpose. By (ii), the Y's must be executed on the second processor at very specific times, as shown in Fig. 2.

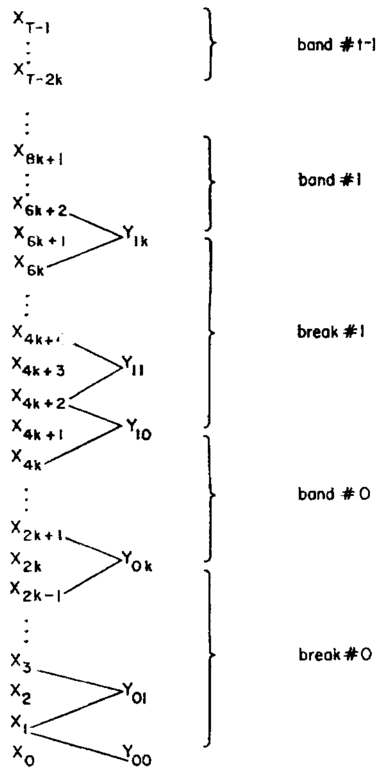

FIGURE 2

That is, progressing in time, we have an alternation of ***breaks***, in which there is one time unit available on processor 2 every other time unit, and ***bands***, in which $2k$ consecutive time units are available on processor 2. Since the unprimed jobs J in S require two time units each, it is clear that they must be executed in the bands only. As there are kt unprimed jobs, they must completely fill the bands, which means that the primed jobs J' for J in S must be executed exclusively during the breaks.

As a consequence, if jobs J and K are executed in the same band, it is not possible that $J \prec K$. For if so, since we have $J \sqsubset K' \sqsubset K$, it follows that K' would also be executed in that band, violating what we have just concluded. Thus, if our instance of (P3) has a solution, we can find a solution to the original instance of (P5) by executing at time unit i exactly those jobs executed in the ith band.

Conversely, if we have a solution to the given instance of (P5), we can find a solution to the constructed instance of (P3) by executing J' in break i and J in band i whenever J is executed at time unit i in the original problem. Hence, the given instance of (P5) has been polynomially transformed to an instance of (P3), proving the lemma.

THEOREM 4. *Two processor scheduling with jobs of one or two time units* (P3) *is NP-complete.*

Proof. By Theorem 2 and Lemma 4. ▌

IV. OPEN PROBLEMS

While the results presented here and in [3, 8, 9] go a long way toward distinguishing tractable from intractable cases of the scheduling problem, there are several open problems which deserve the reader's attention.

(1) Is there a fixed k for which the k-processor equal execution scheduling problem is NP-complete? If so, prove it and find, if possible, polynomial time algorithms for the problem with $3, 4, \ldots, k-1$ processors. If not, are there polynomial algorithms A_k, for all k, that solve the equal execution time problem for k processors? Theorem 1 does not rule out this possibility, although it does say that either:

(i) there is no polynomial (in $\log k$) algorithm to write down a program for A_k, or

(ii) there is no constant c such that the time complexity of A_k is $O(n^c)$ for all k.

(2) Are there combinations of integers k and m such that preemptive scheduling of k processors with jobs requiring $1, 2, \ldots, m$ time units is of polynomial time complexity, or alternatively, NP-complete? References [8, 9] say that there is such a polynomial algorithm for the case $k = 2$ and $m = 1$, since preemptive and nonpreemptive scheduling are the same when $m = 1$. The reader should also be aware that for any k and m, the preemptive scheduling problem can be reduced to k processor single execution time scheduling by replacing a job J with $W(J) = n$ by jobs $J_1 \prec J_2 \prec \cdots \prec J_n$ of one time unit each.

REFERENCES

1. R. L. GRAHAM, Bounds on multiprocessing anomalies and packing algorithms, *Proc. SJCC* (1972), 205–218.
2. S. A. COOK, The complexity of theorem proving procedures, *in* "Proc. 3rd ACM Conference on Theory of Computing," May 1970, pp. 151–158.
3. R. M. KARP, Reducibility among combinatorial problems, TR3, Department of Computer Science, University of California at Berkeley, April 1972.
4. A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, MA, 1974.
5. R. SETHI, Complete register allocation problems, *in* "5th Annual ACM Symp. on Theory of Computing," May 1973.

6. J. BRUNO, E. G. COFFMAN, AND R. SETHI, Scheduling independent tasks to reduce mean finishing time, *Operating Systems Rev.* **7** (1973), 102–103; also see *CACM* **17** (1974), 382–387.
7. M. R. GAREY AND D. S. JOHNSON, Complexity results for multiprocessor scheduling under resource constraints, *in* "Proc. 8th Annual Princeton Conf. on Inf. Sci. and Systems," August 1974.
8. E. G. COFFMAN AND R. L. GRAHAM, Optimal scheduling for two-processor systems, *Acta. Inf.* 1 (1972), 200–213.
9. M. FUJI, T. KASAMI, AND N. NINOMIYA, Optimal sequence of two equivalent processors, *SIAM J. Appl. Math.* **17** (1969), 784–789.