Proof of the 4/3 Conjecture for Preemptive vs. Nonpreemptive Two-Processor Scheduling

E. G. COFFMAN, JR. AND M. R. GAREY

AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. We consider the classical scheduling problem in which a given collection of tasks with lengths $t_1, t_2, \ldots, t_n$ are to be run on two processors, subject to specified precedence constraints among the tasks, so as to minimize the completion time of the last-finishing task, the so-called makespan of the schedule. A schedule is said to be nonpreemptive if each task, once started, is run continuously until its completion t_i time units later, whereas a preemptive schedule allows the running of a task to be temporarily suspended and resumed at a later time, that is, run in noncontiguous pieces whose lengths merely sum to the task length t_i. A long-standing conjecture is that, for any set of tasks and precedence constraints among them, the least makespan achievable by a nonpreemptive schedule is no more than 4/3 the least makespan achievable when preemptions are allowed. In this paper, we prove this conjecture.

Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management—*scheduling*; F.2.2. [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and problems—*sequencing and scheduling*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Makespan scheduling, preemptive scheduling

1. *Introduction*

An instance of the two-processor scheduling problem consists of a directed acyclic graph, commonly called a *precedence graph*, and a set of positive running times, one for each task (vertex) in the graph. An edge from task A to task B in the graph means that task B cannot start until task A is finished. Either of two scheduling disciplines can be assumed: preemptive or nonpreemptive. In the former, a task can be interrupted, set aside, and then resumed later. There is no limit to the number of such preemptions. In the latter, a task, once started, must be run to completion without interruption. Given the discipline, the problem is to find a schedule on the two processors which minimizes the makespan, that is, the completion time of a latest finishing task.

An analysis of the complexity of these problems, along with several of their variants and special cases, can be found in [1] and [5]. In addition to standard

Authors' address: AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974-2070.

complexity issues, the following fundamental question has been posed: Compared to optimal nonpreemptive scheduling, how much more efficient can optimal preemptive scheduling be? More precisely, over all instances I, what is a least upper bound on $\omega_{NP}(I)/\omega_P(I)$, where ω_P and ω_{NP} denote optimal preemptive and optimal nonpreemptive makespans, respectively? This paper proves the following 4/3 bound.

THEOREM 1. *For all instances* I,

$$\frac{\omega_{NP}(I)}{\omega_P(I)} \leq \frac{4}{3}.$$

In addition, there are instances achieving the bound.

The problem of proving Theorem 1 has remained open for two decades; the 4/3 bound originated as a conjecture based on a paper by Liu [4]. However, although a proof of the 4/3 bound seems difficult, a very simple example has been known to achieve the 4/3 ratio. It consists of 3 equal-length, mutually independent tasks, and is illustrated in the timing diagrams of Figure 1. This example can be extended to one having an arbitrarily large number of tasks by replicating this example many times, with a partial order that requires each triplet to be completed before the next triplet can be started.

We list below various special cases for which the 4/3 bound has been proved. (A general discussion can be found in a recent paper by Hong and Leung [3].)

(1) The sum of the task running times in any longest chain of the precedence graph is at most one-third the sum of all task running times [3].
(2) All task running times are in the set $\{1, k\}$ for some integer $k \geq 1$ [2].
(3) All task running times are in the set $\{1, 2, 3, 4\}$ [3].
(4) The precedence graph has a longest chain c such that for every task T not in c, either T is preceded by no task in c, or T is succeeded by no task in c [3]. (Note that tree-structured precedence graphs are included in this special case.)

The proof below of Theorem 1 makes no use of these special cases. Indeed, we do not see how these special cases can help streamline the proof of Theorem 1.

2. *Proof of Theorem* 1

We begin with notational matters and some simple observations. The arguments in the proof make frequent reference to timing diagrams such as those in Figure 1. In these figures, the processors are called simply the *upper* and *lower processor*. Tasks are denoted by letters near the beginning of the alphabet and sets of tasks are denoted by letters near the end of the alphabet. A task's name also denotes its length, or running time.

A *segment* of a schedule S refers to that part of S restricted to a given time interval $[t, t']$, with $0 \leq t < t' \leq \omega(S)$, where $\omega(S)$ denotes the makespan of S. In an *elementary* segment, each processor either runs a piece of only a single

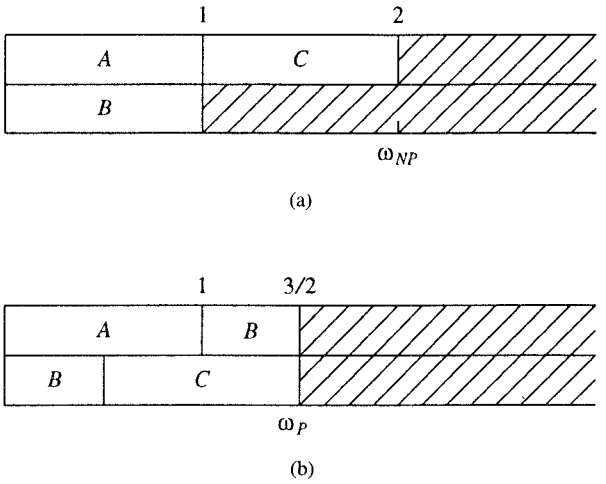


FIG. 1. An instance with unit-length tasks achieving $\omega_{NP}/\omega_P = 4/3$. (a) Nonpreemptive schedule. (b) Preemptive schedule.

task, or is idle, for the duration of the segment. Processor assignments do not change during an elementary segment; and if both processors are busy during such a segment, they are running different tasks. For example, in Figure 1(b) two vertical lines defining an elementary segment must both be in $[0, \frac{1}{2}]$, $[\frac{1}{2}, 1]$, or $[1, \frac{3}{2}]$. For a given instance, a preemptive schedule can then be defined as a finite sequence of elementary segments over consecutive time intervals beginning at time 0 such that (i) the constraints of the precedence graph are respected, and (ii) each task receives a total time precisely equal to its running time. Without loss of generality, we may restrict attention to schedules in which one processor or the other is busy throughout the duration of the schedule.

It is useful to note that interchanging the processor assignments over any segment of a preemptive schedule leaves a valid preemptive schedule. We use this fact repeatedly. Consistent with this property, no preemption is considered to occur when the running of a task is switched instantaneously from one processor to another. It is easy to see that all such preemptions can be eliminated without creating other preemptions or lengthening the schedule.

We also have occasion to exploit symmetries such as the following: Let I' denote the instance I with the direction of the edges in its precedence graph reversed. Then the reverse of any schedule for I gives a schedule for I' having the same makespan.

We now turn to a proof of Theorem 1. We show by induction on the number of tasks that an optimal preemptive schedule can be transformed into a nonpreemptive schedule at most 4/3 as long. The proof technique consists of a sequence of increasingly difficult reductions of the class of preemptive schedules that needs to be considered. As we go along, various easy cases will be identified; these are disposed of by techniques that combine the inductive hypothesis with a constructive argument. The proof is organized into a sequence of ten steps that are numbered for convenient later reference. The more difficult cases are those in Steps (8)–(10). The first step starts with two simple

observations and a fundamental reduction. It concludes by eliminating a class of easy cases.

(1) Let S be an optimal preemptive schedule for instance I.

PROPOSITION 1(a). *If I violates the theorem, then S may be assumed to have no idle time, that is, both processors are busy throughout* $[0, \omega(S)]$.

PROOF. Suppose that S has $x > 0$ time units of idle time in $[0, \omega(S)]$. Add a task A to I, which is independent of all other tasks in I and has length x. Scheduling A during the idle periods of S yields a valid preemptive schedule that has no idle time and that still violates the bound. □

PROPOSITION 1(b). *Suppose S runs a piece of a single task A on one of the processors over some time interval $[t, t']$. Then we may assume that each task running on the other processor during $[t, t']$ runs in a single piece. Also, we can assume that those tasks other than A that finish in $[t, t']$ are scheduled first, before any tasks in that segment that finish after t'.*

PROOF. Tasks that finish in $[t, t']$ must either precede or be independent of any task in $[t, t']$ that finishes later than t'. Thus, a simple interchange argument suffices to prove the result. □

PROPOSITION 1(c). *We may assume that S begins by running an earliest finishing task nonpreemptively.*

PROOF. Let A be a task finishing first at time t, on the upper processor say, and suppose that it is scheduled in the two or more pieces shown in Figure 2(a). Consider the successive exchanges of schedule segments shown, by which the first two pieces of A are moved left so that they run nonpreemptively starting at time 0. By the definition of A, no task starting in $[0, t)$ can finish there, so A and all tasks in $X \cup Y$ are mutually independent. Thus, the exchanges in Figure 2 do not violate precedence constraints, and A remains an earliest finishing task. Extending the exchanges in Figure 2 to all pieces of A then yields the desired result. □

Note that by symmetry, we also have:

PROPOSITION 1(c′). *S can be transformed into a schedule that is at most as long, and that finishes by running nonpreemptively a latest starting task on one of the processors.*

Note that we do *not* claim that S can be transformed so that the properties in Propositions 1(c) and 1(c′) hold simultaneously.

Assume hereafter that we have an optimal preemptive schedule S with the properties in Propositions 1(a), 1(b), and 1(c). We prove by induction on the number n of tasks, that S can be transformed into a nonpreemptive schedule S^* that (i) is at most $4/3$ as long as S and (ii) runs the earliest finishing task of S at the beginning of the schedule; if S has two earliest finishing tasks, then S^* can be made to start with either one (but not necessarily both). It is trivial to prove this result for $n \leq 2$, so $n \geq 3$ is assumed hereafter. Thus, suppose the inductive hypothesis holds for $1, 2, \ldots, n-1$ tasks, and consider the case of n tasks, $n \geq 3$.

We note that, by symmetry, the desired result holds if and only if for any S there exists a nonpreemptive schedule S^* at most $4/3$ as long, which *ends* in a *latest starting* task of S (either can be chosen if there are two such tasks). This

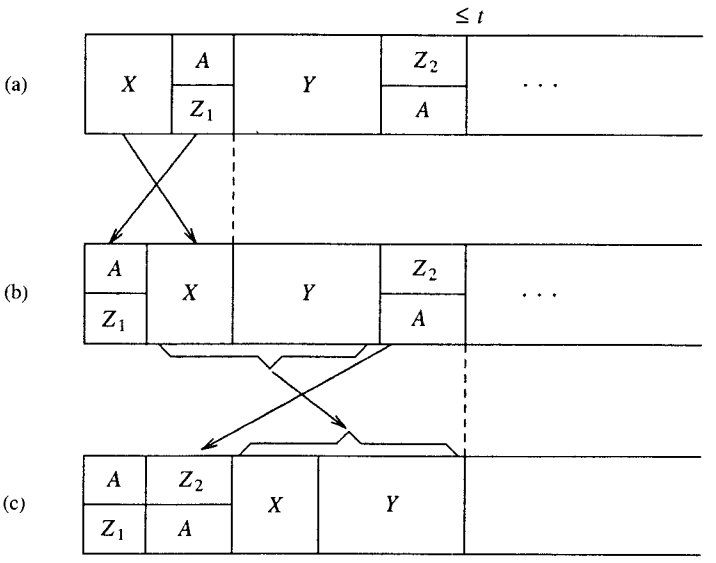


FIG. 2. Property in Proposition 1(c).

fact will be used several times when applying the inductive hypothesis; it will be called the *dual* inductive hypothesis. We continue to use the star superscript to denote nonpreemptive schedules.

As the first application of the method, we prove:

PROPOSITION 1(d). *The inductive step holds if S begins by running two tasks nonpreemptively.*

PROOF. Let S start as shown in Figure 3, where B as well as A runs nonpreemptively. By Proposition 1(b), Figure 3 is organized as shown, where OK is a possibly empty, nonpreemptive sequence of whole tasks, and where no task in X finishes before the end of $B = B_1 \cup B_2$. (Hereafter, OK, OK', etc. will be used generically to represent nonpreemptive sequences of whole tasks.) We may assume $n \geq 4$, for otherwise S must be nonpreemptive.

Now split the schedule at the arrow into the segments S_L and S_R shown. By definition of OK and X, the piece B_2 looked upon as a whole task is a first finishing task in S_R (or has length 0, in the degenerate case). Clearly, S_R has at least one fewer task than S, so by the inductive hypothesis there exists a nonpreemptive schedule S_R^* corresponding to S_R such that $\omega(S_R^*) \leq (4/3)\omega(S_R)$ and such that, if $B_2 > 0$, S_R^* starts with B_2. Now putting S_R^* adjacent to and just after S_L yields a nonpreemptive schedule S^* for I such that

$$\frac{\omega(S^*)}{\omega(S)} \leq \frac{\omega(S_L) + (4/3)\omega(S_R)}{\omega(S_L) + \omega(S_R)} \leq \frac{4}{3},$$

and such that S^* starts with A. □

(2) By Propositions 1(c) and 1(d), we may confine ourselves hereafter to optimal preemptive schedules that begin by running nonpreemptively, a unique earliest finishing task. Throughout the remainder of the proof, we continue

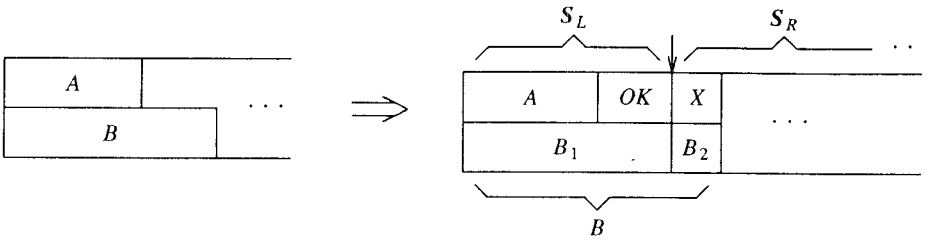


FIG. 3. Schedule S starting with two whole tasks A and B

to call this task A and assume for convenience that it runs on the upper processor. Also, B will always refer to an earliest finishing task independent of A. Such a task clearly exists, since at least one task must run while A is running (by Proposition 1(a)) and all such tasks are independent of A.

In the remainder of this step, we further reduce the class of schedules that need to be considered.

PROPOSITION 2. *An optimal preemptive schedule for I can be assumed to have the form of either Figure 4(a), where A is covered by a piece of B and at least one piece of B is run beyond A, or Figure 4(b), where both A and C are run nonpreemptively and B is preempted exactly once.*

PROOF. We first suppose that no optimal preemptive schedule for I has the form in Figure 4(a), and then show by a sequence of interchange arguments that an optimal preemptive schedule can then be transformed into a schedule of the form in Figure 4(b).

If S is an optimal preemptive schedule for I not in the form of Figure 4(a), then there must be a task C, other than B, that runs while A is running in S. Assume without loss of generality, that C is an earliest-starting such task, and that if a piece of B also runs while A is running, then it runs before the piece of C. Then S has the form shown in Figure 5, where the piece of B under A may not exist, X may be empty, and at least one piece of B runs beyond A. For the time being, it is convenient to assume that the pieces of B are run on the lower processor.

The first interchange argument shows that the schedule in Figure 5 can be further refined so as to have the form in Figure 6, where all pieces of B beyond A are covered by pieces of C. To see this, suppose S has instead the form in Figure 7(a), where some new task D runs in an elementary segment of length $\epsilon > 0$ along with B. Then for ϵ sufficiently small the exchange producing Figure 7(b) can be made, thus increasing the amount of B under A and decreasing the amount of C under A. (The pieces exchanged are called *ϵ-pieces* of B and C.) Since B is an earliest finishing task independent of A, tasks B and C are independent of each other and all tasks in $X \cup Y$. Thus, the exchange cannot violate any precedence constraints. Trivially, B also remains an earliest finishing task not preceded by A. By iterating exchanges of this type until they can no longer be applied, we must arrive at a schedule like Figure 6, or one having the form in Figure 8, where B has increased under A by the size of C, and C has been eliminated under A. Clearly, X cannot be empty if C is eliminated, because if it were, the schedule would look like Figure 4(a), a contradiction. Thus, X has an earliest starting task D as shown in Figure 8.

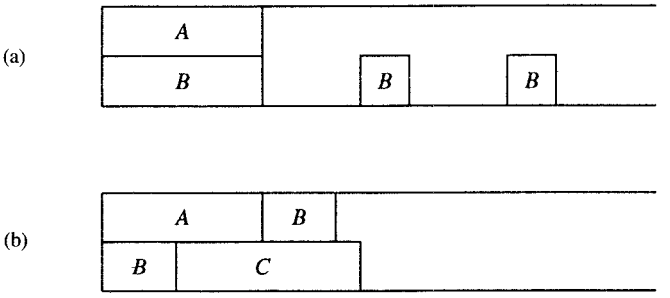


FIG. 4. Normal forms.

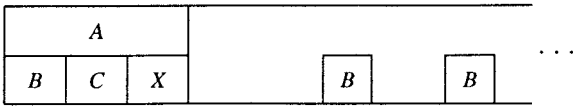


FIG. 5. A schedule not in the form of Figure 4(a).

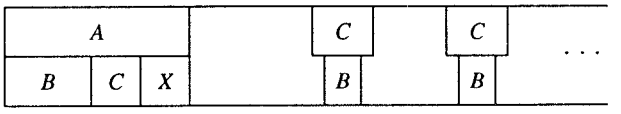


FIG. 6. A first refinement.

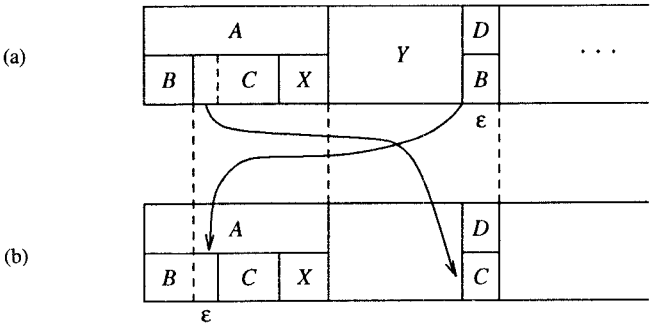
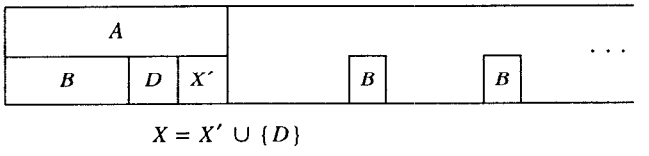


FIG. 7. The first interchange argument.



$X = X' \cup \{D\}$

FIG. 8. Elimination of C under A.

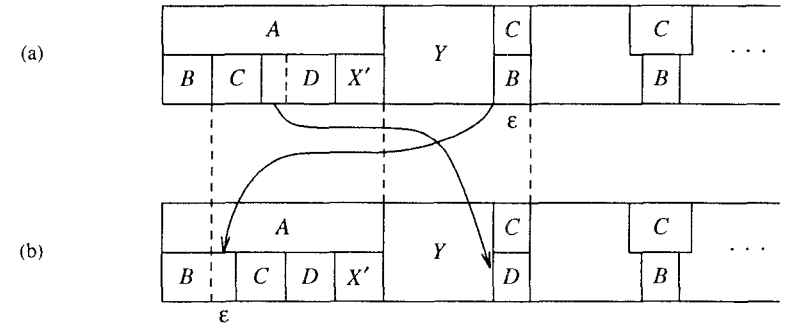


FIG. 9. The second interchange argument.

But then Figure 8 has the same form as Figure 5, so the interchange argument can be repeated. Each exchange increases the amount of B under A, so the exchanges of Figure 7 must eventually yield a schedule like Figure 6.

The second interchange argument shows that S can be put into the form of Figure 6 with X empty. For suppose X is nonempty and that D is the earliest starting task in X, then a transformation of the type shown in Figure 9 must be possible for some $\epsilon > 0$ sufficiently small. In analogy with the earlier argument, tasks B, C, and D are independent of each other and all tasks in $X' \cup Y$. Thus, no precedence constraints are violated in Figure 9(b). Since, by assumption, B cannot be made to cover A in an optimal schedule, iterating the transformation on all tasks in the set X of Figure 6 must eventually lead to a schedule as in Figure 10, with B still an earliest finishing task independent of A. It is possible that the piece of B under A still does not exist, but in that case neither of the first two interchange arguments could have been applied (each moves some of B under A), and S must have started as in Figure 10, with A covered by a single piece of C.

The third interchange step puts Figure 10 into the form of Figure 11, where all elementary segments containing both B and C are moved to the left so that they start right after A, and B runs with at most one preemption. Without loss of generality, we now put the piece of B beyond A on the upper processor and all pieces of C on the lower processor. The exchanges in this transformation are exactly analogous to those used in the proof of Proposition 1(c), and justified by the same argument.

Now suppose C is preempted at least once in Figure 11, and hence the schedule is still not in the form of Figure 4(b). Our last interchange argument shows that all preemptions of C can be eliminated. The exchange operation needed to accomplish this is shown in Figure 12, where D is the first task to run after the first piece of C on the lower processor. Clearly, C is independent of D and all tasks in X, and in Figure 12(b), D still starts after A and B have finished. Thus, since no task finishing time can be increased by the transformation, no precedence constraints are violated in Figure 12(b). Trivially, B remains an earliest finishing task independent of A. The transformation increases the amount of B under A and decreases the gap between the first two pieces of C. Iterating the transformation must therefore eventually collapse all pieces of C into a single piece as in Figure 4(b). It remains to observe that, at the end of this transformation, there will certainly be a piece of B

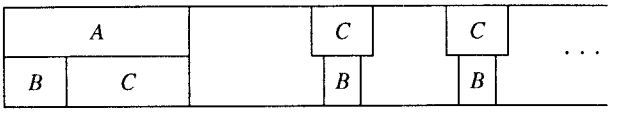


FIG. 10. After the second interchange step.

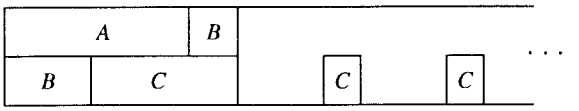


FIG. 11. The third interchange argument.

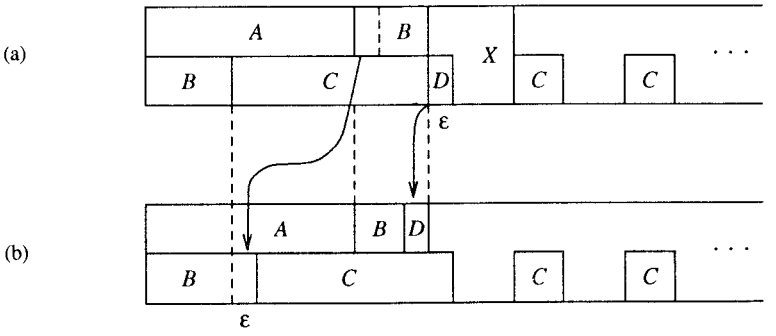


FIG. 12. The fourth and last interchange argument.

under A; for otherwise C would have been run nonpreemptively, violating our assumption that (by Proposition 1(d)) the schedule begins by running exactly one task nonpreemptively. Hence, we have shown that an optimal preemptive schedule for I can be assumed to have the form of either Figure 4(a) or 4(b). □

(3) This step eliminates schedules of the form of Figure 4(b) as possible counterexamples to the theorem.

PROPOSITION 3. *If there is an optimal preemptive schedule for I of the form in Figure* 4(*b*), *then the inductive step holds.*

PROOF. Break down the schedule in Figure 4(b) as shown in Figure 13, where by Proposition 1(b), *OK* denotes a sequence of all tasks other than A, B, and C that start after the beginning of C and finish before the end of C.

Next, separate the schedule at the point indicated by the arrow into two preemptive schedules S_L and S_R. Look upon S_L as a preemptive schedule of independent tasks A, B, and C', where $C' = C_1 + C_2 + C_3'$, and look upon S_R as a preemptive schedule for the tasks in I with A and B removed and with C replaced by C_3''. By definition of OK, C_3'' is a first finishing task in S_R, so by the inductive hypothesis there exists a nonpreemptive schedule S_R^* for the tasks in S_R such that $\omega(S_R^*) \leq (4/3)\omega(S_R)$ and C_3'' starts at the beginning of S_R^*.

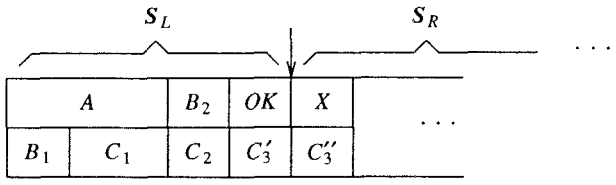


FIG. 13. Inductive argument for Figure 4(b).

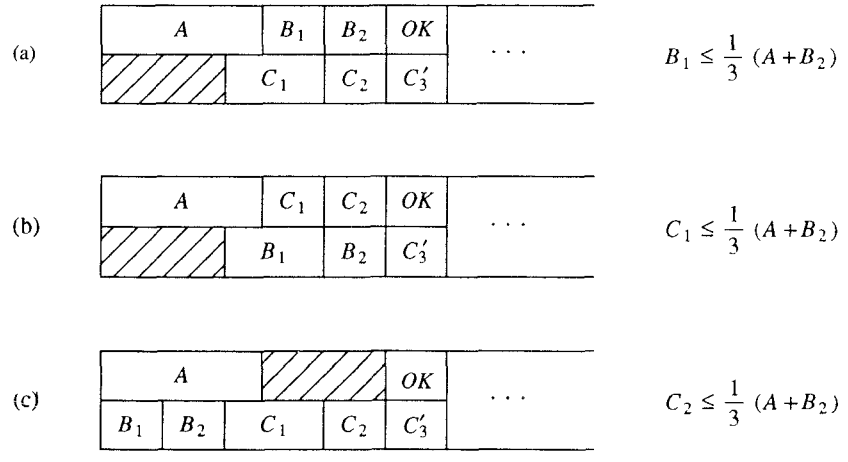


FIG. 14. Nonpreemptive schedules S_L^* with $A = B_1 + C_1$ and $B_2 = C_2$ as in Figure 13

We now convert S_L to a nonpreemptive schedule S_L^* ending with C_3' and *OK*. In Figure 13, it is clear that at least one of B_1, C_1, and C_2 must be no greater than $1/3\ (A + B_2) = 1/3\ (A + C_2)$. We have the following case analysis:

(a) $B_1 \le (1/3)\,(A + B_2)$. In this case, construct the schedule in Figure 14(a). We have

$$\frac{\omega(S_L^*)}{\omega(S_L)} = \frac{A + B_1 + B_2 + C_3'}{A + B_2 + C_3'} \le \frac{(4/3)(A + B_2) + C_3'}{A + B_2 + C_3'} \le \frac{4}{3}.$$

Putting S_L^* adjacent to and before S_R^*, then gives a schedule S^* such that

$$\frac{\omega(S^*)}{\omega(S)} = \frac{\omega(S_L^*) + \omega(S_R^*)}{\omega(S_L) + \omega(S_R)} \le \frac{(4/3)\omega(S_L) + (4/3)\omega(S_R)}{\omega(S_L) + \omega(S_R)} = \frac{4}{3},$$

and such that C' and C_3'' are combined into the original task C, which is run nonpreemptively. Then S^* is the desired nonpreemptive schedule starting with A.

(b) $C_1 \le 1/3(A + B_2) = 1/3(A + C_2)$. Construct the schedule in Figure 14(b) and get

$$\frac{\omega(S_L^*)}{\omega(S_L)} = \frac{A + C_1 + C_2 + C_3'}{A + C_2 + C_3'} \le \frac{(4/3)(A + C_2) + C_3'}{A + C_2 + C_3'} \le \frac{4}{3}.$$

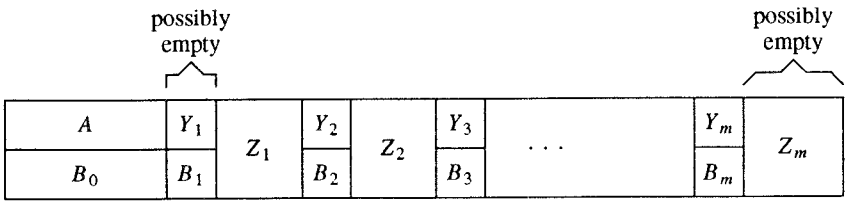


FIG. 15. Breakdown of Figure 4(a).

Then proceed as above to get a nonpreemptive schedule S^* starting with A, such that $\omega(S^*)/\omega(S) \leq 4/3$.

(c) $C_2 \leq 1/3(A + B_2) = 1/3(B_1 + C_1 + B_2)$. Use the schedule in Figure 14(c) and argue as above to get

$$\frac{\omega(S_L^*)}{\omega(S_L)} = \frac{B_1 + B_2 + C_1 + C_2 + C_3'}{B_1 + C_1 + C_2 + C_3'} \leq \frac{(4/3)(B_1 + C_1 + B_2) + C_3'}{B_1 + C_1 + B_2 + C_3'} \leq \frac{4}{3}.$$

As before, we obtain a nonpreemptive schedule S^* starting with A, such that $\omega(S^*)/\omega(S) \leq 4/3$.

Thus, in the case of schedules with the form of Figure 4(b), we have shown that the inductive step holds.

(4) We are left with schedules of the form in Figure 4(a). This step further simplifies such schedules, expressed in the form shown in Figure 15.

Assume that B is as far left as possible in the sense that, for each $i \geq 2$, no rescheduling that leaves $B_1, B_2, \ldots, B_{i-1}$ fixed, will allow us to move some or all of B_i further to the left.

PROPOSITION 4(a). *For each* i, $1 \leq i \leq m$, $Y_i \cup Z_i$ *consists only of whole tasks.*

PROOF. We show first that a new task (i.e., a task not started previously) must begin $Y_i, 1 \leq i \leq m$. This is trivial for $i = 1$, so suppose $i \geq 2$, and suppose task D begins Y_i, but D is not a new task. Let E be a task other than D that ends Z_{i-1}, on the lower processor, say. Then since D, E, B, and whatever runs simultaneously with E are mutually independent, an exchange of the form shown in Figure 16 must be possible for some sufficiently small ϵ, thus contradicting our initial assumption that B_i was as far left as possible.

We now prove that tasks starting in $Y_i \cup Z_i$ must end there. Suppose a task D runs in Z_i but does not end in Z_i. Then, D must be independent of the new task, say E, starting Y_{i+1}. Thus, the exchange of ϵ-pieces, one from B_{i+1} and one from D, shown in Figure 17 can be made, again contradicting our initial assumption. This proves that all tasks run in segment Z_i, $1 \leq i \leq m$, must finish in that segment.

Next, suppose that Y_i contains a piece of some task D that does not finish by the end of segment Z_i. Then, $i < m$ and no part of D can start Y_{i+1} (since a new task must start at Y_{i+1}). By the previous paragraph, D does not run in Z_i. Let E be a task that starts Z_i, say on the lower processor. Then, since E must be independent of B and D, the transformation shown in Figure 18 must be possible. Again, we get a contradiction, so all tasks in Y_i end by the end of Z_i.

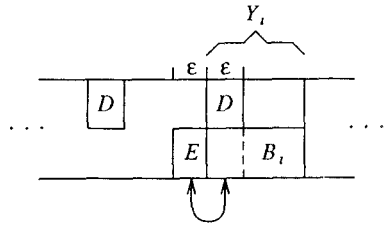


FIG. 16. Y_i starting with a previously run task.

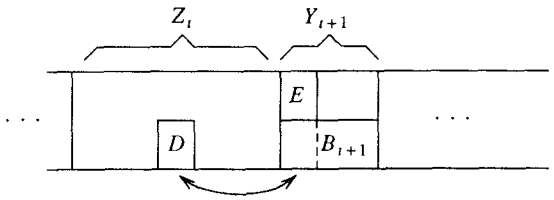


FIG. 17. Exchange of ϵ-pieces of D and B.

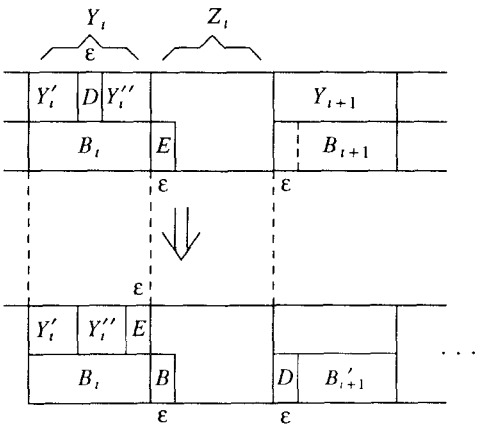


FIG. 18. Moving an ϵ-piece of B_{i+1} to the left.

We have shown that any task run in the segment $Y_i \cup Z_i$ must finish in that segment. It follows by considering $Y_1 \cup Z_1$, $Y_2 \cup Z_2, \ldots$ in turn that a task runs during segment $Y_i \cup Z_i$ if and only if it both starts and finishes there. □

PROPOSITION 4(b). *For each i, $2 \leq i \leq m$, Y_i consists of whole tasks plus at most one partial task, which finishes in a single piece at the beginning of Z_i (and hence runs nonpreemptively).*

Remark. The reader should be sure to notice that this proposition does not hold for $i = 1$.

PROOF. Suppose there is more than one unfinished task in Y_i, $i > 1$. Then by our assumption on the ordering of tasks of Y_i (Proposition 1(b)), the two latest starting tasks in Y_i, say D and E, are unfinished. Then, since $i \geq 2$, there is an ϵ-piece of some task F at the end of Z_{i-1} (on the lower processor) such that we can make the transformation shown in Figure 19, since F is indepen-

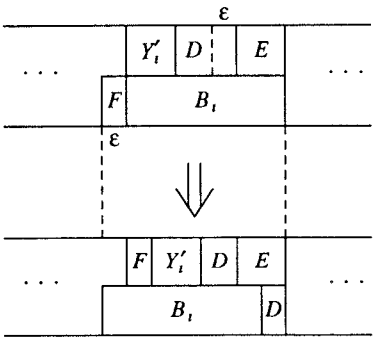


FIG. 19. Moving B_t left.

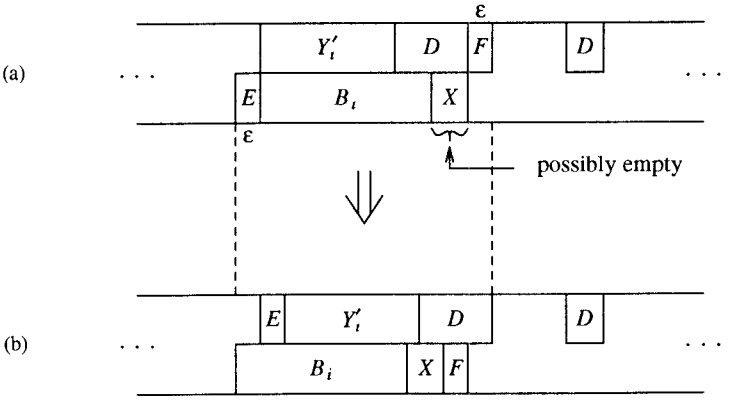


FIG. 20. Again moving B_t left.

dent of B, and D is independent of E (because both are still unfinished). This transformation moves B_t further to the left, and hence contradicts our initial assumption.

Finally, suppose segment Y_t has a single unfinished task, say D, but that D does not finish in a single piece at the start of Z_t. As before we have that Y_t ends with a piece of D. Then for ϵ-pieces of two tasks E and F we have a schedule in the form of Figure 20(a), where E is independent of B and F is independent of D. Then, the schedule in Figure 20(a) can be transformed as shown in Figure 20(b), where again B_t is moved left. This contradiction completes the proof of Proposition 4(b). □

Notationally, let $C_t, i > 1$, denote the task, if any, that starts but does not finish in Y_t. Let C_{t1} denote the piece of C_t at the end of Y_t and let C_{t2} denote the piece at the beginning of Z_t.

(5) This step presents two key results needed in the constructive arguments of the remaining steps. In what follows, the name of a set also denotes the length of the segment spanned by the set. With one exception, this length is simply the sum of the lengths of the tasks or task pieces in the set; the exception is a set Z_t, as in Figure 15, for which this length is exactly one half the sum. For convenience we also let Y_0 denote the singleton $\{A\}$.

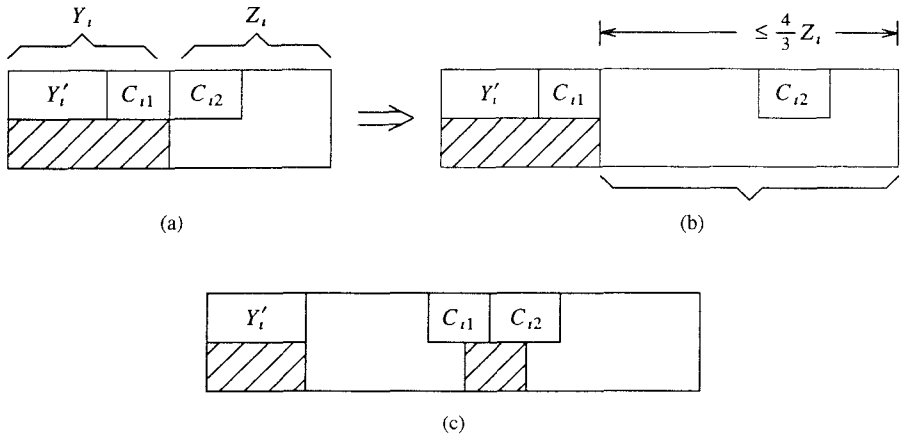


FIG. 21. A nonpreemptive schedule S_i^*. (a) S_i. (b)S_i' (c) S_i^*.

Consider the schedule in Figure 15 with B removed entirely. In this schedule, let S_i denote the segment containing $Y_i \cup Z_i$, where $\omega(S_i) = Y_i + Z_i$.

PROPOSITION 5(a). *A nonpreemptive schedule S_i^* corresponding to S_i can be constructed so that $\omega(S_i^*) \leq Y_i + (4/3)Z_i$.*

PROOF. To see this, first consider the case $i > 1$, where at most a single task C_i remains unfinished from Y_i. Figure 21(a) shows the corresponding schedule S_i. Since Z_i contains fewer tasks than the original instance I (in particular, it contains neither A nor B), we can apply the inductive hypothesis to obtain a nonpreemptive schedule Z_i^* for Z_i that has length at most $(4/3)Z_i$. Figure 21(b) shows a new schedule S_i' obtained from S_i by replacing Z_i with Z_i^*. Note that $\omega(S_i') = Y_i + Z_i^* \leq Y_i + (4/3)Z_i$.

Now, if C_i does not exist, we are done, since $S_i^* = S_i'$ is the required nonpreemptive schedule. If C_i does exist, then the second piece C_{i2} of C_i that belongs to Z_i now runs somewhere within Z_i^*, not necessarily at the beginning, but still all in one piece since Z_i^* is nonpreemptive. Thus, all we need to do to convert S_i' to a nonpreemptive schedule S_i^* for $Y_i \cup Z_i$ is to remove C_{i1}, slide all tasks in Z_i^* that finish before the end of C_{i2} earlier by the length of C_{i1}, and reinsert C_{i1} in the idle space adjacent to the start of C_{i2}. This does not increase the length of the schedule at all, so it still satisfies the required length bound. No precedence constraints are violated, because all the tasks moved are independent of C_i. Since S_i^* is now also nonpreemptive, it is our required schedule.

If $i = 1$, the only difference is that there may be more than one unfinished task remaining from Y_i. However, it is easy to continue performing the transformation described in the preceding paragraph for each of these in turn, without increasing the length of the schedule, finally obtaining, once again, the required schedule. □

In what follows, it will be convenient to refer to the idle time that remains in S_i^* under the whole tasks in Y_i', and the idle time added to Z_i^* during the

insertion of uncompleted tasks from Y_t, as the Y_t *idle time in* S_t^*. The total amount of such idle time in S_t^* is exactly equal to $Y_t = B_t$.

PROPOSITION 5(b). *Consider an arbitrary schedule S that keeps both processors busy during $[0, \omega(S)]$. Suppose S can be converted to a schedule S' in which the amount of time, say T_2, that both processors are busy is at least the amount of time, say T_1, that only one processor is busy. Then $\omega(S')/\omega(S) \leq 4/3$.*

PROOF. We have

$$\frac{\omega(S')}{\omega(S)} = \frac{T_1 + T_2}{T_1/2 + T_2} = 1 + \frac{T_1}{T_1 + 2T_2} \leq 1 + \frac{T_1}{3T_1} \leq \frac{4}{3}. \quad \square$$

In the remainder of the proof, we consider many such transformations $S \to S'$. The proofs that $T_2 \geq T_1$ holds will be called *compensation arguments*; the amount of time T_2 that both processors are busy in S' compensates for the idle time T_1 in S'. The argument will often involve pairing up or "matching" equal length segments of the two types.

We conclude this step with the following simple observation. (The proof is left to the reader.)

PROPOSITION 5(c). *Consider any segment of a preemptive schedule over an interval of length x. The tasks or pieces of tasks in this segment can be run nonpreemptively (with all pieces of each task together) on a single processor in an interval of length $2x$, without violating precedence constraints. Moreover, the schedule can be structured to start with either of the two tasks that begin the two-processor schedule; symmetrically, it can be structured to end with either of the two tasks that end the preemptive schedule.*

(6) Roughly speaking, the method used in the remainder of the proof is as follows: First, we remove from the schedule S (as shown in Figure 15) all pieces of the task B. Next, we create an idle block of length at least B by converting some set of consecutive Z_t segments from two-processor schedules to one-processor schedules, using Proposition 5(c) (the first and last such Z_t segments may be only partially converted). See Figures 22 and 23 for examples. The schedule at this point can be viewed as consisting of three segments, S_L, S_M, and S_R. S_L goes from the beginning of the schedule to the rightmost point that begins a Y_t and is at or before the start of the created idle block. S_R extends to the end of the schedule form the leftmost point that ends a Z_t and is at or after the end of the created idle block. S_M is the segment between S_L and S_R. Note that each of S_L, S_M, and S_R consists of only whole tasks, by their definitions and Proposition 4(a).

Then B is reinserted into the created idle block, and the other tasks in S_M are rearranged (in ways to be described later) to form a nonpreemptive segment S_M^*. Finally, using the inductive hypothesis and Proposition 5(a), the various Y_t, Z_t segments in S_L and S_R are converted into nonpreemptive segments of length at most $Y_t + (4/3)Z_t$. A compensation argument is then used to prove that $S^* = S_L^* S_M^* S_R^*$ satisfies $\omega(S^*) \leq (4/3)\omega(S)$, as desired.

The process of creating an idle block in an amount $x \geq B$ starting or ending at a given time is based on expanding the schedule from which B has been removed over some time interval $[t, t']$. We modify the schedule segment that goes from t to t' in such a manner that an idle period on the lower machine

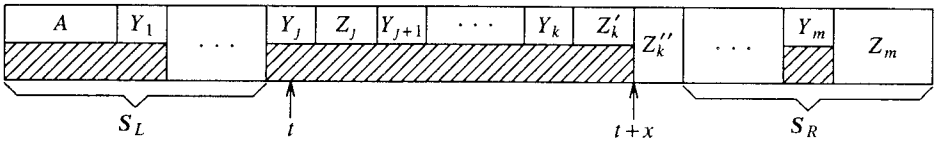


FIG. 22. Creating an idle block in S, with $t + x$ in Z_k.

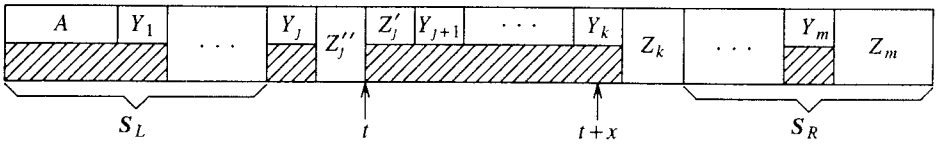


FIG. 23. Creating an idle block in S, with t in Z_j.

of total length x results. Figures 22 and 23 show examples where one end of $[t, t + x]$ in the new schedule falls in a Y_i segment, $0 \le i \le m$, and the other falls in a Z_i segment, $1 \le i < m$. We always choose t' in the original schedule, and hence $t + x > t'$ in the new schedule, to be at most the starting time of Z_m. This is because we insert B at the right end of the space created, and, since some tasks in Z_m might be successors of B, a value of $t + x$ in Z_m might lead to a precedence violation.

As shown in the figures, all of the full Z_i segments in $[t, t']$ (in the original schedule), plus partial segments, if any, at the ends of the interval, are converted to nonpreemptive single-processor schedules and combined with the Y_i segments in an alternating sequence on the upper processor, with the ordering of the new segments the same as before. By Proposition 5(c) we assume that Z_i, $j \le i \le k - 1$ in Figure 22 and Z_i, $j + 1 \le i \le k$, in Figure 23 start with C_{i2} ($j > 1$, since $B > B_0 + B_1$). Then by Proposition 4(b) the sequences $Y_j, Z_j, \ldots, Z_{k-1}, Y_k$ and $Y_{j+1}, Z_{j+1}, \ldots, Y_k$ become nonpreemptive sequences of whole tasks in Figures 22 and 23, respectively. Clearly, precedence constraints are preserved.

It is useful to go through a simple case to illustrate the kind of argument we will be using. Suppose there exist t and $x = B$ such that, after B is removed, S can be expanded into a schedule in which t and $t + B$ fall under some Y_j and Y_k, respectively, $0 \le j < k \le m$. The expansion is shown in Figure 24(a), where OK is the nonpreemptive sequence of tasks in $Z_j \cup Y_{j+1} \cup \cdots \cup Z_{k-1}$. Segment Z_k at the right end can be structured as shown in Figure 24(b), where OK' accumulates the tasks in Z_k that finish before C_{k2}, and C' is the piece of C_{k2} extending beyond OK'. X denotes a segment consisting of the earliest finishing task C' and the whole task remaining in Z_k after C_{k2} and those in OK' have been removed.

S_M^* is defined as in Figure 24(c) by running B in the created idle block, leaving the segment involving OK' unchanged, and replacing X with a nonpreemptive schedule X^* that starts with C' and that is no longer than $(4/3)X$ (using the inductive hypothesis). Then we replace all segments $S_i = Y_i \cup Z_i$ in S_L and S_R by the corresponding segments S_i^* given by Proposition 5(a).

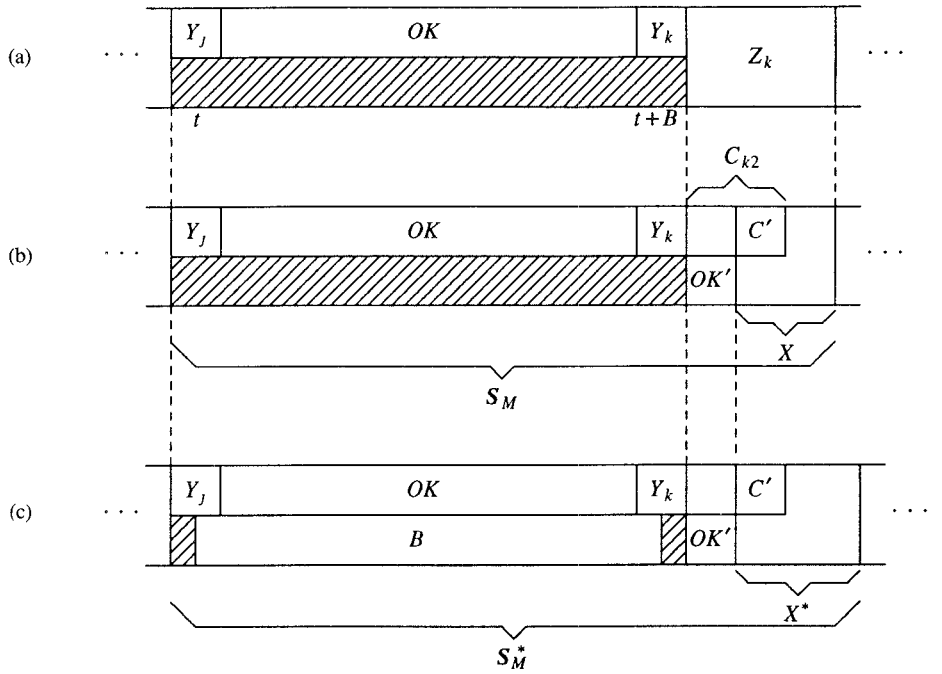


FIG. 24. An expansion/replacement.

This yields the nonpreemptive schedule $S^* = S_L^* S_M^* S_R^*$. It remains to supply the compensation argument.

The idle periods within X^*, and within the Z_t^* segments from which S_R^* and S_L^* were constructed (excluding the Y_t idle time introduced into each Z_t^* when it was formed—see Figure 21(c)), are compensated within these segments themselves, since by the inductive hypothesis $\omega(X^*) \leq (4/3)\omega(X)$ and $\omega(Z_t^*) \leq (4/3)\omega(Z_t)$. The remaining idle time is the Y_t idle time in S_R^* and S_L^* and the idle time under Y_j and Y_k. But $\sum_{t=0}^{m} Y_t = B$ by definition, so the remaining idle time is at most B and is compensated by the fact that both processors are busy in S_M^* while B is running. Thus, by Proposition 5(b), we have $\omega(S^*) \leq (4/3)\omega(S)$.

(7) It remains for us to consider cases in which, in order to expand S to make room for B, the left or right end of the space created must take the form shown in Figures 22 or 23, respectively, that is, it must start or end in a Z_t. Proposition 7(a) first prepares for the required constructions by verifying certain normal forms for Figures 22 and 23. Proposition 7(b) then takes care of another class of easy cases that follow from Proposition 7(a).

PROPOSITION 7(a). *The right-end segment containing Y_k, Z_k', and Z_k'' in Figure 22 can be reduced to one of the two forms shown in Figure 25, where D finishes no later than any task in X, D is either run nonpreemptively (case (a)) or is preempted exactly once (case (b)), and E is run nonpreemptively.*

PROOF. These forms are similar to those in Figures 4(a) and (b), and a proof of the proposition can be based on precisely the same interchange arguments proving the latter forms. In particular, by iterating the exchange

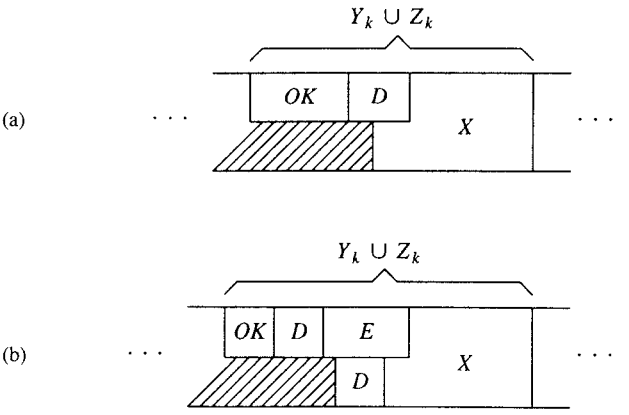


FIG. 25. Normal forms for Figure 22.

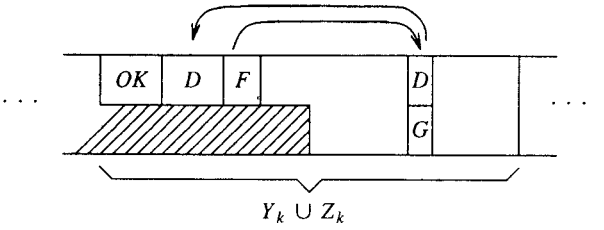


FIG. 26. Putting earliest finishing tasks over the idle period.

sketched in Figure 26, where D is the earliest finishing task past the idle period, we successively put earliest finishing tasks over the idle period, and accumulate them in OK, until we reach the end of the idle period, or until we reach the form in Figure 27. In the first case, we obtain Figure 25(a) by continuing to put the rest of D together using the exchanges in the proof of Proposition 1(c). In the second case, we proceed by using the transformations described in Figures 9–12 until we reach Figure 25(b). Note that in Figure 27, tasks D and E play the earlier roles of B and C, and the piece of the idle period to the right of the arrow plays the role of A. □

PROPOSITION 7(a′). (THE DUAL OF PROPOSITION 7(a)). *The left-end segment containing Z'_j, Z''_j in Figure* 23 *can be reduced to one of the two forms in Figure* 28, *where the starting time of D' is at least as late as that of any task in X.*

PROOF. This result follows easily from the symmetric counterparts of the arguments for Proposition 7(a). Indeed, we need only consider the reverse of the schedule for Z'_j and Z''_j in Figure 23(b), with the precedence constraints of its tasks reversed. To this schedule, we apply the transformations of the proof of Proposition 7(a), then reverse the new schedule to obtain one of the forms in Figure 28. Note that the neighboring Y_j segment is not restructured in Figure 28, as it is in Figure 25. □

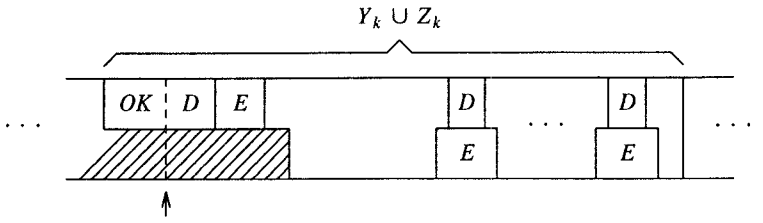


FIG. 27. A form leading to Figure 25(b).

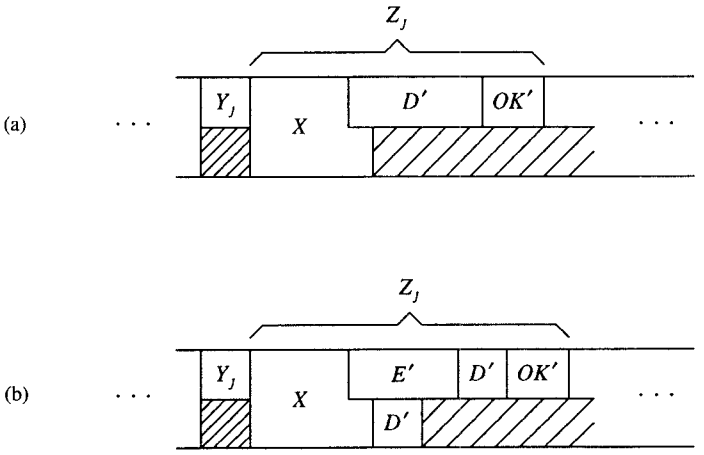


FIG. 28. Normal forms for Figure 23.

The normal forms in Figures 25(a) and 28(a) allow us to dispose of the following additional easy cases:

PROPOSITION 7(b). *Suppose we can create by expansion an idle block of length B such that*

—*it begins under some Y_j, $j \geq 0$, or in some Z_j, $1 \leq j < m$, such that the segment containing Z'_j and Z''_j in Figure 23 can be put in the form of Figure 28(a), and*
—*it ends under some Y_k, $2 \leq k \leq m$, or in some Z_k, $1 \leq k < m$, such that the segment containing Y_k, Z'_k, and Z''_k in Figure 22 can be put in the form of Figure 25(a).*

Then we can transform S into a nonpreemptive schedule S^* no longer than $(4/3)\omega(S)$.

PROOF. Consider first the case above where the left and right ends are in some Z_j and Z_k, $1 \leq j < k < m$. This case can be illustrated as in Figure 29, where D and D' have been split into the consecutive pieces shown. D'_1 is a latest starting task in X_L and D_2 is an earliest finishing task in X_R.

Comparing Figures 24(b) and 29, we see that the schedules at the right end of the length-B idle block can be made structurally the same by taking OK' empty in Figure 24, and identifying C' with D_2. Thus, as in Step (6), we insert B into the idle block in S_M, replace X_R by an optimal nonpreemptive

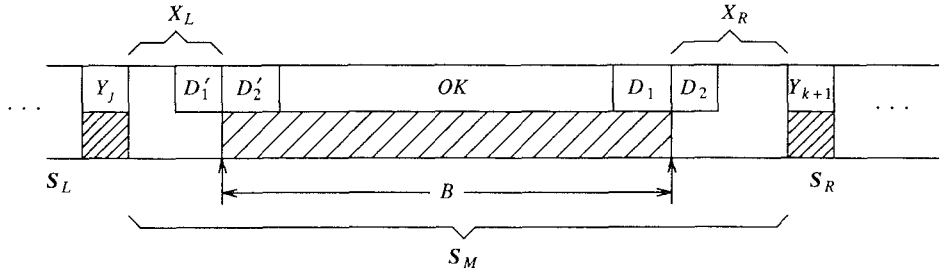


FIG. 29. An easy case based on Figures 25(a) and 28(a).

schedule starting with D_2, and replace the segments S_i to the right of X_R and to the left of Y_j by the corresponding S_i^* from Proposition 5(a).

Using the dual inductive hypothesis we next replace X_L by an optimal nonpreemptive schedule X_L^* ending with D_1'. This may leave one or more (if $j = 1$) preemptions at the end of Y_j; if so, the insertion operation of the proof of Proposition 5(a) is applied to eliminate them. It is easy to verify that the insertions leave D_1' as a latest finishing task of this segment, so it still matches up with D_2'.

The compensation argument for S^* works as before, combining the Y_i idle time in S_L^* with the Y_j idle time in X_L^*. Thus, we have a nonpreemptive schedule S^* starting with A such that $\omega(S^*) \leq (4/3)\omega(S)$.

It remains to observe that if either end of the space created falls under a Y_i, while the other can be put in the form of Figures 25(a) or 28(a), then the argument above can be coupled easily with that of Step (6) to again obtain the desired nonpreemptive schedule S^*. □

(8) We are left with the most difficult case, where any idle block of length B created by expansion must begin or end with the form in Figures 25(b) or 28(b), respectively. In this step, we deal with this case, handling two remaining exceptional cases in Steps (9) and (10). The transformation involves two stages.

For the first stage, define the index $h \geq 0$ by the inequality

$$\sum_{i=0}^{h-1} Y_i < \frac{B}{2} \leq \sum_{i=0}^{h} Y_i;$$

that is, B becomes half finished in S during Y_h. The first stage creates by the usual expansion an idle block starting at the left end of Y_h and ending at a point in some Z_k, $k < m$, assuming that this is possible, that is, that

$$\sum_{i=h}^{m-1} (Y_i + 2Z_i) + Y_m \geq B.$$

The case where this inequality does not hold will be covered in Step (9); the analysis in Steps (9) and (10) (the last two steps) will also make clear why we have chosen here to create space starting from the left end of the segment in which B becomes half finished in S.

By Steps (6) and (7), the normalized form of the schedule at the right end of the created idle block must be as shown in Figure 25(b). We observe that this

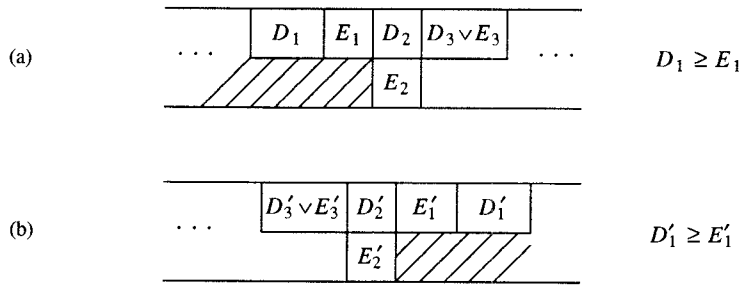


FIG. 30. Structuring the normal forms in Figures 25(b) and 28(b).

form can be expressed as in Figure 30(a), where $D_1 \geq E_1$ and where $D_3 \vee E_3$ denotes either a third and final piece of D or a similar piece of E. (Note that, if $D_1 < E_1$ initially, we reverse the roles of D and E to obtain this form.) We will also be using the dual of this observation, which is shown in Figure 30(b) and applies to Figure 28(b).

With E_1 as defined by the first stage, the second stage creates by expansion an additional idle block of length $2E_1$ just to the left of Y_h, assuming that this can be done without extending to the left of A, that is, that

$$A + \sum_{i=1}^{h-1} (Y_i + 2Z_i) \geq 2E_1.$$

Step (10) concludes the proof by taking care of the case in which this inequality does not hold.

After creating the total of $B + 2E_1$ time units of space in the two stages described above, we must have one of the three schedules given in Figure 31, which use the normal forms (or their duals) from Figures 25(a), 28(a), or 30 on the two ends. The cases in Figure 31 depend on whether the left end of the created idle block lies in a Y_j (case (a)) or a Z_j, and in the latter case, on whether it takes the normal form of Figure 28(a) (case (b)) or Figure 30(b) (case (c)). Then $D_1 \geq E_1$ and $D_2 = E_2$ in all figures, and $D_1' \geq E_1'$, $D_2' = E_2'$ in Figure 31(c). S_M is the segment having X_L and X_R as its left and right end portions, and S_L and S_R are the segments to the left and right of S_M. By the procedure given in Step (6), $X_R \cup S_R$ can be replaced by a nonpreemptive schedule $X_R^* \cup S_R^*$ beginning with the task $D_3 \vee E_3$. The dual of this procedure converts $S_L \cup X_L$ to a nonpreemptive schedule $S_L^* \cup X_L^*$ ending in $D_3' \vee E_3'$ (or an appropriate part of D'). It remains to find a nonpreemptive schedule for B and the remaining parts of S_M such that S_M^* begins and ends with pieces of tasks matching those that begin and end the schedules just constructed, and such that a compensation argument yields 4/3 bound, accounting for idle time within S_M^* itself as well as for the Y_i idle time in X_L^*, S_L^*, and S_R^*.

We give the construction for Figure 31(c) below; we point out later how the remaining constructions for Figures 31(a) and (b) follow as special cases. If $E_1' + E_2' \geq E_1 + E_2$ and X_R^* begins with a piece of E, then we choose Figure 32(a) or Figure 32(b); the choice depends on whether or not D_2 extends beyond B. If $E_1' + E_2' \geq E_1 + E_2$, but X_R^* begins with a piece of D, then

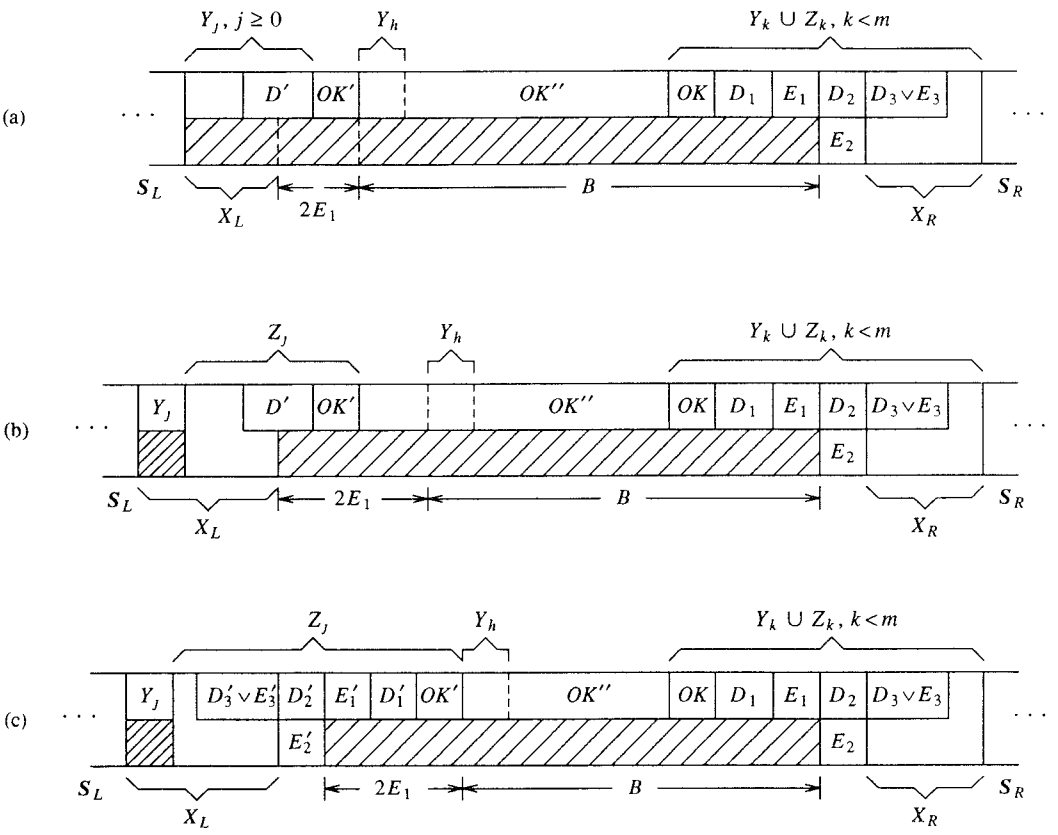


FIG. 31. Possible schedules after creating an idle block of length $B + 2E_1$ with exactly B time units of the idle block to the right of the start of Y_h.

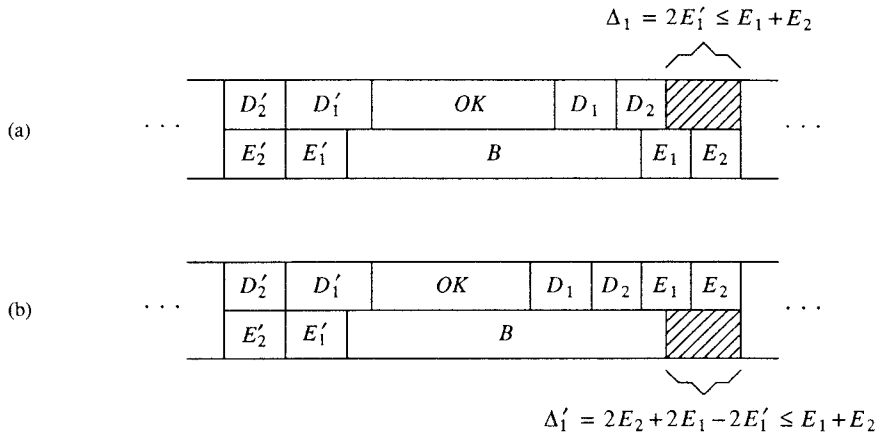


FIG. 32. Replacing B when $E_1' + E_2' \geq E_1 + E_2$.

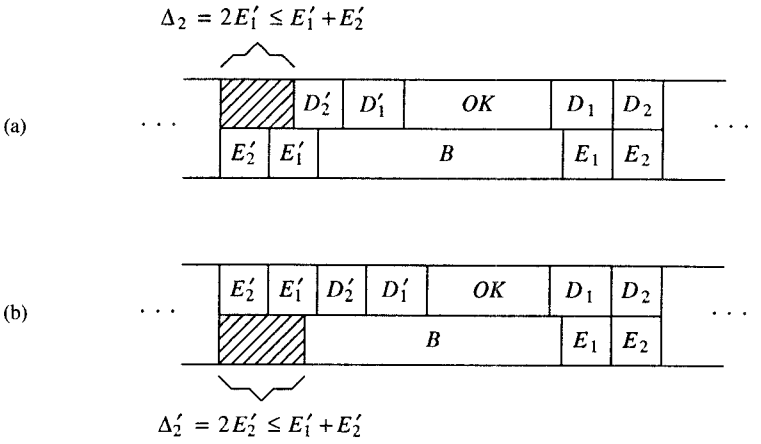


FIG. 33. Replacing B when $E_1 + E_2 \ge E_1' + E_2'$.

we choose Figure 32(a) with D_1 and D_2 right justified, or Figure 32(b) with D_1 and D_2 exchanged with E_1 and E_2; again the choice depends on whether or not D_2 (when D_1 and D_2 are left justified) finishes beyond B. If $E_1' + E_2' < E_1 + E_2$, then one of the symmetric choices is taken from Figure 33, with D_1' and D_2' left justified in the case of Figure 33(a). In all four figures, OK is the union of OK, OK', and OK'' from Figure 31. The following observations verify that this schedule has the necessary properties:

(a) As reflected in all figures $D_1 \ge E_1$, $D_1' \ge E_1'$, $D_2 = E_2$, and $D_2' = E_2'$. Trivially, after right-justifying in Figure 32(a) and left-justifying in Figure 33(a), this part of the schedule begins and ends with the appropriate tasks $D_2' \vee E_2'$ and $D_2 \vee E_2$, respectively.

(b) A comparison of Figure 31(c) with Figures 32 and 33 gives the expressions for the idle times shown. Trivially, Δ_1, Δ_2, and Δ_2' are non-negative. From Figure 31(c), we have $2E_1 \ge D_1' + E_1' \ge 2E_1'$. Then $E_1 \ge E_1'$, so $\Delta_1' \ge 0$ holds as well. Thus, the schedule ends in E_2 in Figure 32, begins with E_2' in Figure 33, and keeps the tasks in OK over B in all figures. The upper bounds on the idle times can be seen by inspection.

(c) B, D, and E are given as mutually independent in Figure 31(c), as are B, D', and E'. Since OK is over B, the precedence relations between $\{D', E'\}$ and OK are respected, as are those between OK and $\{D, E\}$.

The compensation argument is straightforward. For example, if the schedule is chosen as in Figure 32(a), then since $E_1' + E_2' \ge E_1 + E_2$, the idle time of at most $E_1 + E_2$ in the central portion of S_M^* is compensated by the fact that both processors are busy while E_1' and E_2' are running. Both processors are also busy while B is running, so the Y_t idle time in $S_L^* \cup X_L^*$ and S_R^* is compensated as in Steps (6) and (7). A similar argument applies to the remaining possibilities in Figures 32 and 33.

It remains to observe that the cases corresponding to Figures 31(a) and (b) can be treated in the same way, with E_1', E_2', and D_2' set to 0, and with D' broken into D_1' and D_3' in the obvious way. Then the middle of the new schedule looks like Figure 33(a), but with no idle time at all, so the compensation argument is trivial.

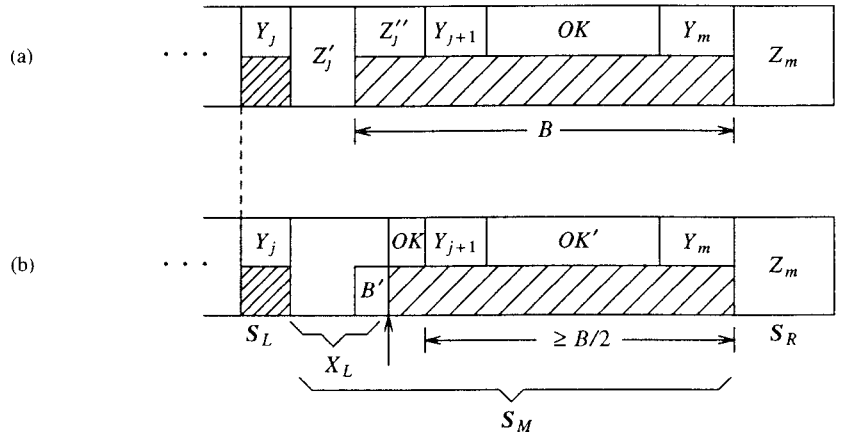


FIG. 34. The case $\sum_{i=h}^{m-1}(Y_i + 2Z_i) + Y_m < B$.

In the final two steps, we return to the special cases identified earlier where, for one of two reasons, the idle block of length $B + 2E_1$, could not be created.

(9) Suppose that, after removing B from S, we cannot create an idle block of length B starting at the left end of Y_h (in which B becomes half finished in S) and ending before the start of Z_m, that is, suppose

$$\sum_{i=h}^{m-1}(Y_i + 2Z_i) + Y_m < B.$$

In this case, we create an idle block of length B that ends at the beginning of Z_m. Since the sum of the lengths of the Y_i's is B, this must be possible without extending to the left of A. Moreover, we may suppose that the newly created idle block does not begin under some Y_i, for this would give us the easy case in Step (6). Figure 34(a) shows an example where the idle block begins in a Z_j. By the definition of Y_h, we must have $j < h$, so that the sum of the Y_i's before Z_j is at most $B/2$ and the sum of the Y_i's that are over the length-B idle block is at least $B/2$.

Using the dual of Proposition 1(b), we rearrange Z''_j so that tasks completely within Z''_j go at its end (in OK). Then we restructure the schedule as shown in Figure 34(b), where B is divided into two parts, B' and B'', with $B' = Z''_j - OK$ inserted as shown in the figure. Note that B' is a latest starting task in X_L.

S^* is formed simply by placing B'', the remainder of B, under the tasks in S_M in the corresponding idle block, taking care of $Z_m = S_R$ as in Step (6), and taking care of $S_L \cup X_L$ as in Step (7) to obtain a nonpreemptive schedule ending in B'.

The Y_i idle time needing compensation is only in $S^*_L \cup X^*_L$ and is at most $B/2$. Since $B'' \geq B/2$ (see Figure 34), this idle time is compensated by the fact that both processors are busy while running B''.

(10) In this final step, we assume that

$$2E_1 > A + \sum_{i=1}^{h-1}(Y_i + 2Z_i),$$

that is, in Step (8) it was not possible to extend the idle period to the left by $2E_1$ time units with E_1 as shown in Figure 30(a). After removing B from S, we now create an idle block of length B starting at time 0. Assuming that this does not give us the easy case in Step (6), this space must end in a Z_k. Since

$$\sum_{i=0}^{m} Y_i = B,$$

we have $k < m$. Also, we observe that $k \geq h$ and hence

$$Y^* \equiv \sum_{i=k+1}^{m} Y_i \leq \frac{B}{2},$$

because

$$B \geq D_1 + E_1 \geq 2E_1 > A + \sum_{i=1}^{h-1} (Y_i + 2Z_i)$$

and because the idle block ends in a Z_k. According to Proposition 7 and the normal form in Figure 30(a), the schedule can be put into one of the two forms shown in Figure 35, where $A = Y_0$ still starts at time 0. Note that the D and E tasks of Figure 35 are not those in the above inequalities, that is, those in Figures 29 and 30 of Steps (7) and (8). This should cause no confusion; the remainder of the proof uses D and E only to refer to the tasks identified in Figure 35.

Figure 35(a) just specializes the easy case in Proposition 7(b) by taking S_L and X_L empty. For Figure 35(b) we consider the following subcases:

(a) $B \geq D_2 + E_2 + Y^*$ ($Y^* = \sum_{i=k+1}^{m} Y_i \leq B/2$).

In this case, S_L is empty and S_M is formed as in the schedule of Figure 36, or this schedule with the ordering of D and E reversed (so as to match $D_3 \vee E_3$). Comparing Figures 35(b) and 36 shows that the idle time in S_M^* is given by $\Delta = D_2 + E_2$. Next, to the right of S_M^*, S_R is replaced by S_R^* as in Step (7) to yield the desired nonpreemptive schedule S^*. Since $B \geq D_2 + E_2 + Y^* = \Delta + Y^*$, the idle time in S_M^* as well as the Y_i idle time in S_R^* is compensated by the time during which both processors are busy in S_M^*.

(b) $OK + D_1 + D_2 \geq 2E_1 + Y^*$.

In this case, we produce the schedule for S^* given in Figure 37, where the value $\Delta = 2E_1$ is obtained by comparing Figures 35(b) and 37, and where S_R^* is obtained from S_R as before. The compensation argument again succeeds, because both processors are busy in S_M^* for time $OK + D_1 + D_2 \geq 2E_1 + Y^*$, and the idle time that needs to be compensated is at most $2E_1 + Y^*$.

(c) $2(E_1 + E_2 + Y^*) \geq OK + D_1 + D_2 + B$.

This is the last subcase that needs to be covered, because $D_2 = E_2$ implies that the sum of the right-hand sides of the inequalities in the three subcases is equal to the sum of the left-hand sides. With OK, D_1, D_2, E_1, E_2, $D_3 \vee E_3$, and Y^* as defined by Figure 35(b), we now change to the construction in Step (9). After removing B from S, we create an idle period of length B that ends at the start of Z_m. Assuming that B is small enough, it must be possible to

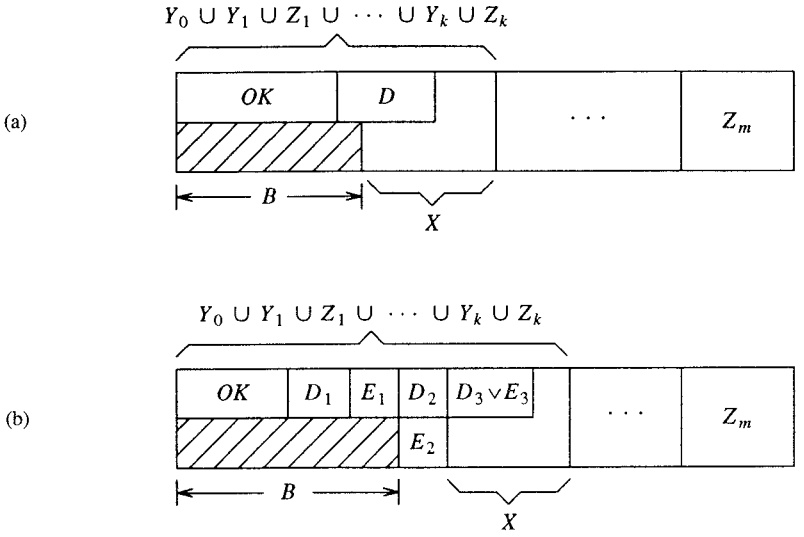


FIG. 35. Examples for Step (10).

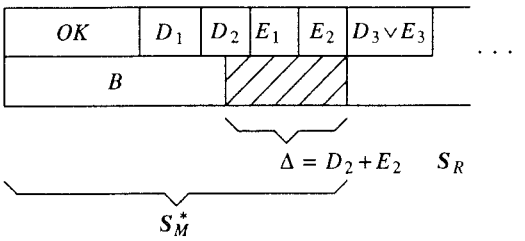


FIG. 36. The case $B \geq D_2 + E_2 + Y^*$.

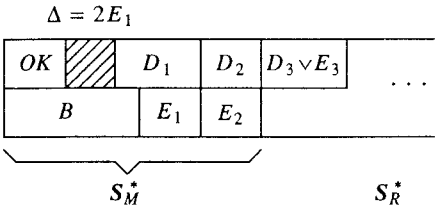


FIG. 37. The case $OK + D_1 + D_2 \geq 2E_1 + Y^*$.

construct the (preemptive) schedule in Figure 38 where S_R contains all of B, starts with $D_3 \vee E_3$ and has no idle time. To be precise, we need that

$$2X + \sum_{i=k+1}^{m-1} (Y_i + 2Z_i) + Y_m \geq B,$$

in order to do this.

By the inductive hypothesis S_R can be replaced by a nonpreemptive schedule S_R^* starting with $D_3 \vee E_3$ such that $\omega(S_R^*) \leq (4/3)\omega(S_R)$; that is, the idle time in S_R^* is compensated within S_R^*. But $Y^* \leq B/2$ and $2(E_1 + E_2 + Y^*) \geq OK + D_1 + D_2 + B$ imply that $2(E_1 + E_2) \geq OK + D_1 + D_2$. It follows that

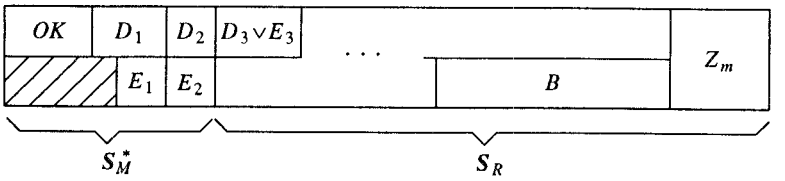


FIG. 38. The case $2(E_1 + E_2 + Y^*) \geq OK + D_1 + D_2 + B$.

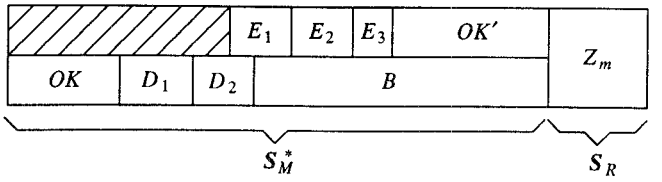


FIG. 39. B is large.

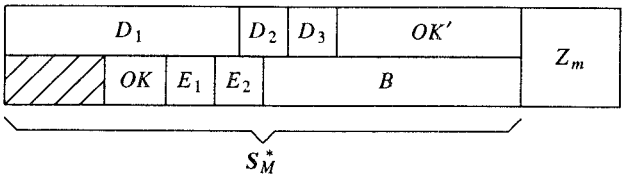


FIG. 40. D_1 extends to the left of OK.

the idle time in S_M^* is compensated by the amount of time that both processors are busy in S_M^*.

Now suppose that B is too large to allow the schedule in Figure 38. If $D_3 \vee E_3 = E_3$, that is, if E_3 extends into X in Figure 35(b), then we create the schedule in Figure 39, where OK' contains only whole tasks, except possibly for the portion C_{m1} of a task C_m that runs nonpreemptively and finishes in Z_m (see Figure 21(a)). Since $OK' \geq Y^*$, the condition $2(E_1 + E_2 + Y^*) \geq OK + D_1 + D_2 + B$ implies that $2(E_1 + E_2 + OK') \geq OK + D_1 + D_2 + B$. Thus, the idle time in S_M^* is compensated within S_M^*. The expansion of S_R and the remainder of the argument follow as in Step (6); all idle time in S_R^* is compensated within S_R^* itself.

Finally, suppose $D_3 \vee E_3 = D_3$. Then, after exchanging D_1 and D_2 with E_1 and E_2, we adopt the schedule of Figure 39, where D_1 may or may not extend to the left of OK. If it does not, the same compensation argument again applies since $D_1 \geq E_1$ and $D_2 = E_2$. On the other hand, if D_1 does extend to the left, as in Figure 40, then $D_2 = E_2$ and $OK' \leq B$ imply that the idle time is at most D_1. (For S^* we need to left-justify OK so that A begins the schedule.) The compensation argument within S_M^* is trivial since $B \geq D_1$. As before, the argument concludes by dealing with $S_R = Z_m$ as in Step (6).

3. *Concluding Remarks*

It is unpleasant to have such a complex proof for such a simply stated theorem. We hope that the existence of this first proof will stimulate and assist others

in finding a significantly shorter and simpler proof. There does not seem to be much hope that this proof can be extended to prove the corresponding conjecture for an *arbitrary* number m of processors, that $\omega_{NP}(I)/\omega_P(I)$ is never more than $2m/(m + 1)$ [4]. Here, also, a simplified proof of Theorem 1 is preferred as a better starting point for generalization.

ACKNOWLEDGMENT. We are indebted to Tomas Feder for carefully listening to all the details of the proof, pointing out several gaps that needed fixing, and preparing the first comprehensive set of notes on the proof, which served us well in writing this paper. We are also pleased to acknowledge the careful reading by a referee who pointed out several flaws and areas for improvement.

REFERENCES

1. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, Calif, 1979.
2. GOYAL, D. K. Non-preemptive scheduling of unequal execution time tasks on two processors. Tech. Rep. CS-77-039. Computer Science Dept., Washington State Univ., Pullman, Wash., 1977.
3. HONG, K. S., AND LEUNG, J. Y-T. Some results on Liu's conjecture. *SIAM J Comput.*, to appear.
4. LIU, C. L. Optimal scheduling on multiprocessor computer systems. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*. IEEE Computer Society, New York, 1972, pp. 155–160.
5. ULLMAN, J. D. Complexity of sequencing problems. In *Computer and Job-Shop Scheduling Theory*, E. G. Coffman, Jr., ed. Wiley, New York, 1976.

RECEIVED MARCH 1991; REVISED MAY 1992; ACCEPTED MAY 1992