Complexity Results for Single Machine Distance Constrained Scheduling Problems

MIT-LCS-TM-587

Daniel W. Engels
dragon@caa.lcs.mit.edu

David Karger
karger@theory.lcs.mit.edu

Srinivas Devadas
devadas@caa.lcs.mit.edu

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

November 1998

Abstract

Scheduling problems that involve timing constraints between tasks occur often in machine shop scheduling (e.g., job shop scheduling problems) and code scheduling during software compilation for pipelined processors (e.g., multiprocessor sequencing and scheduling problems). The timing constraints often take the form of start-start and finish-start time lags. We consider the case of non-negative finish-start time lags $l_{i,j}$, i.e., distance constraints, that require a minimum time from the completion of task T_i to the start of task T_j. New complexity results are derived for several single machine scheduling problems for both the makespan, $C_{\max}$, and mean flow time, $\sum C_j$, objective functions.

Keywords: scheduling, distance constraints, time lags, complexity theory

1 Introduction

We examine the complexity of several single machine scheduling problems in which distance constraints (non-negative finish-start time lags) are associated with the precedence constraints. Distance constraint $l_{i,j}$ requires that there exist a minimum of $l_{i,j}$ time units from the completion time C_i of task T_i to the start time s_j of task T_j, i.e., $C_i + l_{i,j} \leq s_j$. Distance constraints are a generalization of the classical precedence constrained scheduling problems where $l_{i,j} = 0$.

Distance constraints often arise in practice. For example, scheduling software on a pipelined functional unit that does not contain bypass circuitry requires that all instructions complete the last stage of the pipeline before their results are available to any other instruction. Therefore, if the pipeline has k stages, then all instructions that require the result from a previous instruction a must wait at least k cycles after instruction a enters the pipelined functional unit before entering the pipelined functional unit. This software scheduling problem may be made more difficult if some, but not full, bypass circuitry exists or different instructions require different numbers of pipeline stages to complete. Thus, the distance constraints may not all be equal.

Recent work [BHH97] has shown that scheduling distance constrained tasks on a single machine is often at least as hard as scheduling the same tasks without distance constraints on multiple machines. Although this relationship is not always true, it indicates that the addition of distance constraints can drastically increase the complexity of a single machine scheduling problem.

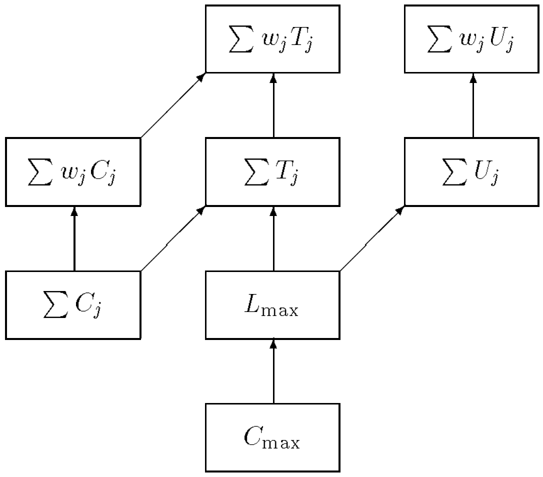


Figure 1: Complexity hierarchy for scheduling problems that differ only in their objective functions. An arrow from γ_1 to γ_2 indicates that $\alpha\ |\beta\ |\gamma_1\ \propto\ \alpha\ |\beta\ |\gamma_2$.

We use the $\alpha|\beta|\gamma$ notation introduced in [GLLR79] to specify scheduling problems. α denotes the type of scheduling problem and the number of machines to be used. In the paper we only consider $\alpha \in \{1,\ P,\ Pm\}$. The single machine environment, $\alpha = 1$, is the simplest machine environment that is a special case of all other more complicated machine environments. $\alpha = P$ indicates that the scheduling will be performed on an arbitrary number (given as input to the problem) of identical parallel machines, and $\alpha = Pm$ indicates that the scheduling will be performed on a fixed number m of identical parallel machines. β denotes the constraints on the task characteristics and on a feasible schedule. We specify β using five parameters, β_1, β_2, β_3, β_4, and β_5[1]. $\beta_1\ \in\ \{\circ$, chain, intree, outtree, tree, prec$\}$ denotes the precedence constraint topology. $\beta_2 \in \{\circ$, pmtn$\}$ denotes the allowability of preemptions in a feasible schedule. $\beta_3\ \in\ \{\circ,\ p_j = 1,\ p_j \in P_j\}$ denotes restrictions on the task processing times. $\beta_4\ \in\ \{\circ\ ,\ r_j\}$ denotes the presence of nonzero release times, and $\beta_5\ \in\ \{\circ\ ,\ l_{i,j} = L = \mathcal{O}(1),\ l_{i,j} = l,\ l_{i,j}\ \in\ \{0, l\}\}$ denotes restrictions on the distance constraints. $l_{i,j} = L$ indicates that all distance constraints are equal in value, and the value is a fixed constant. Whereas, $l_{i,j} = l$ indicates that all distance constraints are equal in value, but the value is given as an input to the problem. $l_{i,j} \in \{0, l\}$ indicates that the distance constraint values may only be one of the values in the given set, i.e., 0 or l. γ denotes the objective function to be minimized. We only consider problems with $\gamma\ \in\ \{C_{\max},\ \sum C_j\}$.

A complexity hierarchy describing the relationships between scheduling problems that differ only in their objective functions is shown in Figure 1. This hierarchy indicates that a problem with the objective function at the head of an arc reduces to a problem with the objective function at the tail of the arc, e.g., $\alpha\ |\beta\ |C_{\max}$ reduces to $\alpha\ |\beta\ |L_{\max}$. Thus, problems with an objective function of $C_{\max}$ and $\sum C_j$ are the simplest with respect to their objective functions, and determining that they are $\mathcal{NP}$-complete proves that the corresponding problem is $\mathcal{NP}$-complete for all objective functions in the complexity hierarchy.

We examine the complexity of several single machine distance-constrained scheduling problems for both the $C_{\max}$ and the $\sum C_j$ objective functions. We prove that most of these problems are

[1] The absence of constraints of type β_i, $1 \leq i \leq 5$, is denoted by a $\circ$. For clarity, these symbols are not included in the $\alpha|\beta|\gamma$ problem statement.

Problem	*Presented*
$^*1\|chain; pmtn; l_{i,j} = l\|C_{\max}$	Section 2.2
$^*1\|chain; pmtn; l_{i,j} = l\|\sum C_j$	Section 2.2
$^*1\|chain; p_j = 1; l_{i,j} \in \{0, l\}\|C_{\max}$	Section 2.3
$^*1\|chain; p_j = 1; l_{i,j} \in \{0, l\}\|\sum C_j$	Section 2.3
$^*1\|chain; p_j \in \{1, 2\}; l_{i,j} = L\ (L \geq 2)\|C_{\max}$	Section 2.4
$^*1\|chain; p_j \in \{1, 2\}; l_{i,j} = L\ (L \geq 2)\|\sum C_j$	Section 2.4
$^*Pm\|chain; p_j = 1\|\sum w_j C_j$	Section 2.5
$P\|intree; p_j = 1; l_{i,j} = l;\|\sum C_j$	Section 3.1
$P\|outtree; p_j = 1; l_{i,j} = l;\|\sum C_j$	Section 3.1
$^*1\|prec; p_j = 1; l_{i,j} = l\|C_{\max}$	Section 3.2
$^*1\|prec; p_j = 1; l_{i,j} = l\|\sum C_j$	Section 3.2

Table 1: Complexity results presented in this paper. A '*' indicates the problem to be strongly $\mathcal{NP}$-complete.

$\mathcal{NP}$-complete in the strong sense[2] for both of these objective functions; therefore, the problems are $\mathcal{NP}$-complete in the strong sense for all of the standard objective functions shown in Figure 1. Table 1 summarizes our complexity results.

These complexity results narrow the boundary between known polynomial time solvable problems and known $\mathcal{NP}$-complete problems. Figure 2 graphically illustrates the boundary with regard to possible task processing times and distance constraints on a single machine. Figure 3 graphically illustrates the boundary with regard to possible constraint topologies and distance constraints on a single machine. Each circle represents a problem. A filled-in circle represents a known $\mathcal{NP}$-complete problem. An empty circle represents a known polynomial-time solvable problem, and a dotted circle represents an unknown complexity for the problem. Problems with new complexity results obtained in this paper are marked with a square. Problems whose complexity was previously known are marked with a circle.

The remainder of this paper is organized as follows. Section 2 presents the $\mathcal{NP}$-completeness proofs for several related problems with β_1 = chain and discusses the boundary between $\mathcal{NP}$-complete problems and polynomial time solvable problems with respect to them. Section 3 presents polynomial time scheduling algorithms for a simple problem involving β_1 = intree and β_1 = outtree along with the $\mathcal{NP}$-completeness results for several related problems with β_1 = prec. The boundary between $\mathcal{NP}$-complete problems and polynomial time solvable problems with respect to these problems is also discussed. Finally, Section 4 summarizes the results of this work and discusses some related open problems.

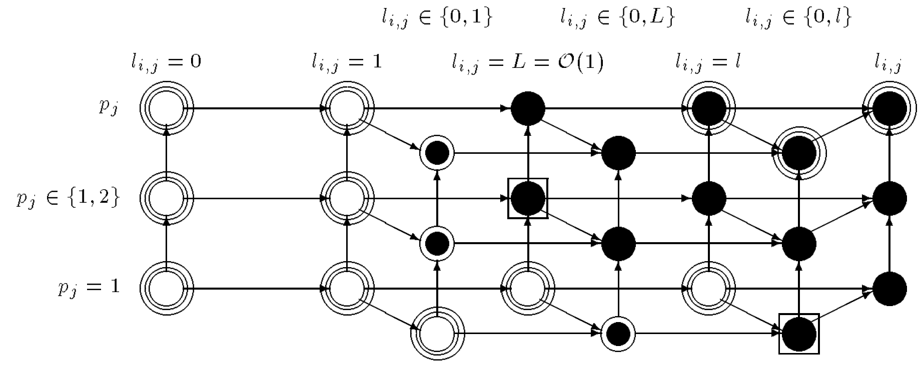
2 Chain Structured Tasks

In this section we examine the complexity of several problems involving chain structured tasks, i.e., problems in which the precedence constraints form sets of chains. We show that for the makespan, $C_{\max}$, and the mean flow time, $\sum C_j$, objective functions these problems are strongly $\mathcal{NP}$-complete; therefore, they are strongly $\mathcal{NP}$-complete for all of the objective functions shown in Figure 1.

The 3-Partition problem is reduced to all of the various scheduling problems in their respective

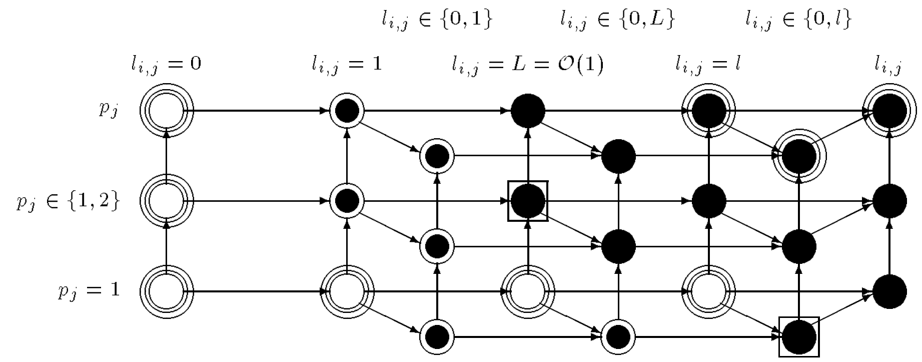
[2] A problem is $\mathcal{NP}$-complete in the strong sense if it cannot be solved by a pseudo-polynomial time algorithm. Thus, a strongly $\mathcal{NP}$-complete problem cannot be solved in pseudo-polynomial time even if its parameters are bounded by a polynomial.

$\mathbf{1|\boldsymbol{chain};\beta_2;\beta_3;\beta_5|C_{\max}}$



(a)

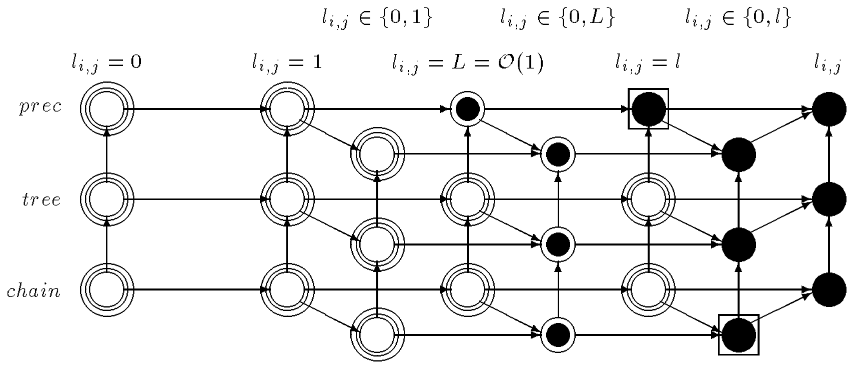
$\mathbf{1|\boldsymbol{chain};\beta_2;\beta_3;\beta_5|\sum C_j}$



(b)

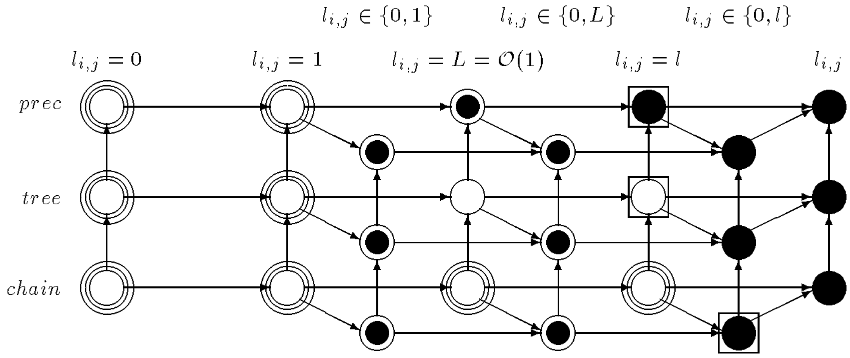
Figure 2: Graphical representation of the known boundary for single machine scheduling problems involving chain precedence constraints. Part (a) depicts the boundary for the makespan objective function. Part (b) depicts the boundary for the mean flow time objective function. Nodes corresponding to problems with new complexity bounds presented in this paper are boxed. Nodes corresponding to problems whose complexity was previously known are doubly circled. We use the representation given by [GJ79]. Problems are represented by circles, filled-in if known to be $\mathcal{NP}$-complete, empty if known to be in $\mathcal{P}$, and dotted if their complexity is unknown. An arrow from Π_1 to Π_2 signifies that Π_1 is a subproblem of Π_2.

$1|\beta_1; p_j = 1; \beta_5|C_{\max}$



(a)

$1|\beta_1; p_j = 1; \beta_5|\sum C_j$



(b)

Figure 3: Graphical representation of the known complexity boundary for unit execution time scheduling problems. Part (a) depicts the boundary for the makespan objective function. Part (b) depicts the boundary for the mean flow time objective functions.

$\mathcal{NP}$-completeness proofs. The 3-Partition problem is stated as follows.

> Given a set of $3q$ elements $A = \{a_1, a_2, \ldots, a_{3q}\}$, a bound B, and a size $s(a_j)$ for each $a_j \in A$ such that $B/4 < s(a_j) < B/2$ and $\sum_{a_j \in A} s(a_j) = qB$, can A be partitioned into q disjoint sets $A_1, A_2, \ldots, A_q$ such that for $1 \le k \le q$, $\sum_{a_j \in A_k} s(a_j) = B$?

For each scheduling problem Π in which this reduction is performed we show that for every instance of 3-Partition we can compute an instance of Π and a value z in polynomial or pseudo-polynomial time such that the instance of 3-Partition has a 'Yes' solution if and only if the instance of Π has a schedule of length at most z (for the makespan objective) or the sum of the completion times is at most z (for the mean flow time objective). The strong $\mathcal{NP}$-completeness results follow from the fact that 3-Partition is $\mathcal{NP}$-complete in the strong sense [GJ79].

2.1 $\mathbf{1|chain; l_{j,k} = l|C_{max}, \sum C_j}$

The most basic chain structured tasks scheduling problems involving distance constraints are $1|chain; l_{j,k} = l|C_{\max}, \sum C_j$. The complexity of these problems was first examined by Balas et. al.[BLV95] and Brucker and Knust[BK98] respectively. Balas et. al. were able to prove the strong $\mathcal{NP}$-completeness of $1|chain; l_{j,k} = l|C_{\max}$ with a simple reduction from 3-Partition. Brucker and Knust extended this proof in a straight forward manner to handle the mean flow time objective function. The problems examined in the remainder of this section are restricted versions of these two problems (in the sense that they contain additional β constraints compared with these problems), and their respective $\mathcal{NP}$-completeness proofs are very similar in structure to the simple $\mathcal{NP}$-completeness proofs of $1|chain; l_{j,k} = l|C_{\max}, \sum C_j$. For this reason, we repeat the $\mathcal{NP}$-completeness proofs for $1|chain; l_{j,k} = l|C_{\max}, \sum C_j$.

Theorem 2.1 ([BLV95]) $1|chain; l_{j,k} = l|C_{\max}$ *is $\mathcal{NP}$-complete in the strong sense.*

Proof This problem is clearly in $\mathcal{NP}$. To prove that it is strongly $\mathcal{NP}$-hard we reduce the 3-Partition Problem to it.

Given an instance of the 3-Partition Problem, we construct an instance of the $1|chain; l_{j,k} = l|C_{\max}$ problem as follows. There are $4q+1$ tasks. For each $a_j \in A$, there is a task T_j with processing time $p_j = s(a_j)$. For each $i \in \{0, \ldots, q\}$, there is a task X_i with processing time $p_i = B$. The tasks X_i form a chain $X = X_0 \prec X_1 \prec \cdots \prec X_q$ with all distance constraints equal to B. The tasks T_j have no precedence constraints associated with them. We define $z = 2qB + B$. Note that the precedence constraints form a single chain and that the processing times of all the tasks is equal to z. Also, note that the chain X requires z time units to complete due to the distance constraints. Thus, any schedule that completes before the deadline, z, must not have any idle time, and the tasks in X must be scheduled as soon as possible. The template formed by X is shown in Figure 4. It is easily verified that this reduction requires time polynomial in the parameters of the 3-Partition Problem.

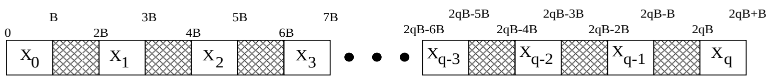


Figure 4: Template formed from the X chain in the proof of Theorem 2.1.

Suppose there exists a 'Yes' solution to the 3-Partition Problem. Since there exists a 'Yes' solution to the 3-Partition Problem, there exist q 3-element subsets S_i such that $\sum_{j \in S_i} p_j = B$. Since the chain X can be scheduled in time $2qB + B$ leaving

q intervals each containing B time units, each of the q sets S_i may be scheduled within one interval for a total schedule time of $2qB + B$.

Suppose that a schedule of length z exists. The chain X requires a minimum time of $2qB + B$ to complete. Since there are exactly B time units between each of the tasks in X in a schedule that completes in time $2qB + B$ and preemption is not allowed, each set of tasks S_i scheduled between tasks X_{i-1} and X_i must have exactly B units of execution time. Furthermore, since $B/4 < p_j < B/2$ $\forall T_j$ there must be at least three tasks in each set S_i. The fact that there are exactly q sets S_i requires that there be exactly three tasks in each set S_i. Thus, there exists a partition into q 3-element subsets S_i, such that $\sum_{T_j \in S_i} p_j = \sum_{T_j \in S_i} s(a_j) = B$ $(1 \leq i \leq q)$, and the schedule is a witness to such a partition. Thus, we have a 'Yes' instance of 3-Partition. ■

The reduction for the mean flow time objective function is essentially identical to the reduction for the makespan objective function. The main difference is that the *template chain* X has a large number of extra tasks in it. Combined with the appropriate cost z, these extra tasks force the tasks in chain X to be scheduled as soon as possible. A large number of extra tasks are required in X since the exact finish times of the tasks corresponding to the elements of A are not known. The sum of the finish times of the tasks corresponding to the elements in A may only be bounded. Thus, without the additional tasks in chain X, it is possible to have a solution with mean flow time at most z when the corresponding 3-Partition problem does not contain a 'Yes' solution.

Theorem 2.2 ([BK98]) $1|chain; l_{j,k} = l|\sum C_j$ *is $\mathcal{NP}$-complete in the strong sense.*

Proof This problem is clearly in $\mathcal{NP}$. To prove that it is strongly $\mathcal{NP}$-hard we reduce the 3-Partition Problem to it.

Given an instance of the 3-Partition Problem, we construct an instance of $1|chain; l_{j,k} = l|\sum C_j$ as follows. There are $n = 3q + M$ tasks where $M = 2(3qB + B)q(q+1) + q$. For each $a_j \in A$, there is a task T_j with processing time $p_j = s(a_j)$. For each $i \in \{1, \ldots, M\}$, there is a task X_i with processing time $p_i = 3qB$. The tasks X_i form a chain $X = X_1 \prec X_2 \prec \cdots \prec X_M$ with all distance constraints equal to B. The tasks T_j have no precedence constraints associated with them. We define

$$z = \sum_{i=1}^{M}[3qBi + B(i-1)] + \sum_{j=1}^{q}[3j(3qB + B)].$$

Suppose that there exists a 'Yes' solution to the 3-Partition Problem. Since there exists a 'Yes' solution to the 3-Partition Problem, there exist q 3-element subsets S_i such that $\sum_{T_j \in S_i} p_j = B$. Since the chain X can be scheduled such that there exists $M - 1$ 'holes' each containing B time units, the tasks in X are scheduled as soon as possible. Thus, each of the q sets S_i may be scheduled within one hole. We schedule the sets S_i in the first q holes. The resulting mean flow time of the schedule satisfies

$$\begin{aligned}
\textstyle\sum_{j=1}^{n} C_j &= \textstyle\sum_{j=1}^{M} C_j + \sum_{i=1}^{3q} C_{M+i} \\
&\leq \textstyle\sum_{i=1}^{M}[3qBi + B(i-1)] + \sum_{j=1}^{q}[3j(3qB + b)] \\
&= z.
\end{aligned}$$

Suppose that a schedule with mean flow time at most z exists. The first q tasks of chain X must be scheduled as early as possible. Assume that this is not the case.

Then,

$$\begin{array}{rcl}\sum_{j=1}^{n} C_j & \geq & \sum_{j=1}^{M} C_j \\ & \geq & \sum_{j=1}^{q}[j3qB + (j-1)B] + \sum_{j=q+1}^{M}[j3qB + (j-1)B + 1] \\ & = & z + (M-q) - \sum_{j=1}^{q}[3(3qB + B)j \\ & \geq & z + (M-q) - \frac{3}{2}(3qB + B)q(q+1) \\ & = & z + (M-q) - \frac{3}{4}(M-q) \\ & > & z,\end{array}$$

which is a contradiction. Thus, the tasks of the chain X must be scheduled as early as possible.

In order for the mean flow time of the schedule to be at most z the tasks T_j, $1 \leq j \leq 3q$, must be scheduled before task X_{q+1}. Since there are exactly B time units between each of the tasks in X and preemption is not allowed, each set of tasks S_i scheduled between tasks X_{i-1} and T_i, $2 \leq i \leq q+1$, must have exactly B units of execution time. Furthermore, since $B/4 < p_j < B/2$ $\forall T_j$ there must be at least three tasks in each set S_i. The fact that there are exactly q sets S_i requires that there be exactly three tasks in each set S_i. Thus, there exists a partition into q 3-element subsets S_i, such that $\sum_{j \in S_i} p_j = \sum_{j \in S_i} s(a_j) = B$ $(1 \leq j \leq q)$, and the schedule is witness to such a partition. Thus, we have a 'Yes' instance of 3-Partition. ■

2.2 $1|chain; pmnt; l_{j,k} = l|C_{\max}, \sum C_j$

A slightly 'easier' version of the above problems allows preemption in a feasible schedule. The $\mathcal{NP}$-completeness proofs for the non-preemptive problems do not hold when preemption is allowed. Therefore, we must reexamine their complexity in light of preemption.

Balas et. al.[BLV95] were able to prove that when preemption is allowed and the distance constraints are restricted to a set of two values that are inputs to the problem, the problem with the makespan objective function is strongly $\mathcal{NP}$-complete. We improve upon this result by showing that preemption does not improve the complexity of the problem when all distance constraints are equal to the same value, i.e., $1|chain;pmnt;l_{j,k} = l|C_{\max}, \sum C_j$ are strongly $\mathcal{NP}$-complete.

The $\mathcal{NP}$-completeness proofs are very similar to those for the non-preemptive problems. The main differences are that each element of A corresponds to a chain of tasks instead of a single task and the template chains form $2q$ intervals within which these tasks may be scheduled. The key idea is that the schedule from the first q intervals (corresponding roughly to the schedule in the non-preemptive proofs) constrains the possible schedules in the second q intervals (the mirror schedule).

Theorem 2.3 *$1|chain;pmnt;l_{j,k} = l|C_{\max}$ is $\mathcal{NP}$-complete in the strong sense.*

Proof $1|chain;pmnt;l_{j,k} = l|C_{\max}$ is clearly in $\mathcal{NP}$. To prove that it is also strongly $\mathcal{NP}$-hard, we reduce the 3-Partition Problem to it.

Without loss of generality we assume that $B \gg 3$ and that $s(a_j) \gg 3$, $\forall\ a_j \in A$. If this is not the case, then we may scale B and $s(a_j)$, $\forall\ a_j \in A$, by a constant without affecting the 3-Partition Problem. This constraint is required to ensure the validity of the transformation.

Given an instance of the 3-Partition Problem, we construct the following instance of $1|chain;pmnt;l_{j,k} = l|C_{\max}$. There are $3q^2 + 7q + 1$ tasks arranged in $3q + 2$ chains. For each $a_j \in A$ there is one chain C_j containing $q+1$ tasks, $C_{(j,1)}$, $C_{(j,2)}$, $\ldots$, $C_{(j,(q+1))}$. The processing times for $C_{(j,1)}$ and $C_{(j,(q+1))}$ are $s(a_j)$. All other processing times for the tasks in C_j are equal to 1.

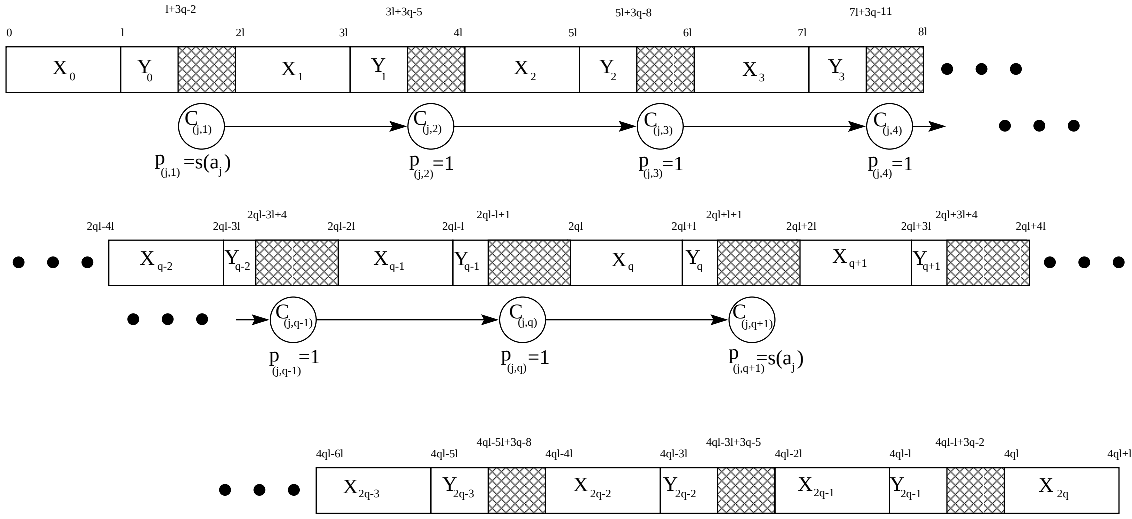


Figure 5: Template formed from the X chain and the Y chain in the proof of Theorem 2.3. This template limits the scheduling structure of the C_j chains.

Two additional chains, X and Y, are created. X contains $2q+1$ tasks each with processing time l (defined below). Y contains $2q$ tasks with processing times of

$$p(Y_i) = \begin{cases} 3q - 3i + 1 \text{ for } 1 \leq i \leq q \\ 3i - 3q - 2 \text{ for } q+1 \leq i \leq 2q \end{cases}$$

All distance constraints are equal. We define the non-zero distance constraint to be

$$l = B + 3q - 2.$$

The deadline for the schedule is $z = 4ql + l$. Note that the processing times of all the tasks is equal to $4ql + l$; thus, any schedule that completes before the deadline must not have any idle time. Figure 5 illustrates how the template chain X and the enforcer chain Y create a template within which the C_j chains must be scheduled. It is easy to verify that this reduction requires time polynomial in the parameters of the 3-Partition problem.

Suppose we have a 'Yes' instance of 3-Partition. A schedule of length z is constructed as follows. Start the tasks of chain X as soon as possible. $X_{(2q+1)}$ finishes at time z. We have $2q$ intervals, $I_1, I_2, \ldots, I_{2q}$, each of length l within which the remaining tasks must be scheduled. Chain Y has $2q$ tasks; therefore, one task of chain Y must be executed during each interval. Furthermore, the distance constraints require that no more than one task of Y may be executed during any one interval. Thus, schedule Y_i in interval I_i. After scheduling chain Y, the empty time slots in each interval I_i are characterized by the following function.

$$empty(I_i) = \begin{cases} B + 3(i-1) & 1 \leq i \leq q \\ B + 3(2q - i) & q+1 \leq i \leq 2q \end{cases}$$

By assumption of a 'Yes' instance of 3-Partition, there exists q disjoint 3-task sets, $H_1, H_2, \ldots, H_q$, with processing time of B and comprised of the first task from the chains C_j, $1 \leq j \leq 3q$. Schedule H_k in interval $I_k, 1 \leq k \leq q$. Consider task $C_{(j,1)}$ scheduled in interval I_i ($i \leq q$ due to how we scheduled the sets H_k). The tasks $C_{(j,2)}, C_{(j,3)}, \ldots, C_{(j,q)}$ can be scheduled during the next $q-1$ intervals. The additional empty time slots left by chain Y allow these 'pass-through' tasks to be scheduled in these intervals. By scheduling them thus, $C_{(j,(q+1))}$ can be scheduled in the interval I_{i+q}. The resulting schedule is feasible with a makespan of z.

Conversely, suppose that we have a schedule of length z. As before, the tasks in chain X must be scheduled as soon as possible, and exactly one task from chain Y must be scheduled in each interval I_i, $1 \leq i \leq 2q$. The tasks from the C_j chains are scheduled without idle times in the remaining time slots of the intervals. Furthermore, due to the distance constraints and lengths of the chains C_j, the first tasks from these chains, $C_{(j,1)}$, $1 \leq j \leq 3q$, must be scheduled in the first q intervals I_i, $1 \leq i \leq q$, and the last tasks from these chains, $C_{(j,(q+1))}$, $1 \leq j \leq 3q$, must be scheduled in the last q intervals I_i, $q+1 \leq i \leq 2q$.

Consider the first interval I_1 and only the first tasks in the chains C_j. Let S_1 be the set of $C_{(j,1)}$ tasks that are started and finished in I_1. Task Y_1 requires $3q-2$ time units in I_1 leaving B time units to schedule other tasks. Assume that $\sum_{C_{(j,1)} \in S_1} p_j = B - c$ for some $c > 0$. Consider the interval I_{q+1}. Y_{q+1} has an execution time of 1. There are at most $3q$ unit execution time tasks from the chains C_j that may be scheduled in this interval. The only additional tasks that may be scheduled in this time interval are the final tasks of the chains C_j whose first task was scheduled in the first interval I_1 such that $C_{(j,1)} \in S_1$. $|S_1| \leq 3$ by the constraints in the 3-Partition Problem, and c is characterized by the following function due to our assumptions on B and $s(a_j), a_j \in A$.

$$c \geq \begin{cases} B & \text{if } |S_1| = 0 \\ B/2+1 & \text{if } |S_1| = 1 \\ 2 & \text{if } |S_1| = 2 \\ 3 & \text{if } |S_1| = 3 \end{cases}$$

Therefore, there are $c - 3 + |S_1| > 0$ idle time slots in interval I_{q+1}. It follows that our assumption is incorrect, and $\sum_{C_{(j,1)} \in S_1} p_j = B$. An iterative application of this argument leads to the identification of sets S_i with $\sum_{C_{(j,1)} \in S_i} p_j = B$ for $1 \leq i \leq q$ and to the conclusion that we have a 'Yes' instance of 3-Partition. ■

As in the previous section, the proof for $1|chain; pmnt; l_{j,k} = l|\sum C_j$ follows the proof for $1|chain; pmnt; l_{j,k} = l|C_{\max}$. This is achieved by making the template chain X and the enforcer chain Y much longer than the chains corresponding to the elements of A. The increased lengths of X and Y ensures that they are scheduled as soon as possible except possibly at the end. The chains corresponding to the elements of A must then be scheduled without idle times in the first $2q$ intervals caused by X if there is a 'Yes' instance of 3-Partition. Scheduling a task from these chains after all tasks in X (or most tasks in X) causes the mean flow time of the resultant schedule to be greater than the target value z.

Theorem 2.4 $1|chain; pmnt; l_{j,k} = l|\sum C_j$ *is* $\mathcal{NP}$*-complete in the strong sense.*

Proof $1|chain; pmnt; l_{j,k} = l|\sum C_j$ is clearly in $\mathcal{NP}$. To prove that it is also strongly $\mathcal{NP}$-hard, we reduce the 3-Partition Problem to it.

Without loss of generality we assume that $B \gg 3$ and that $s(a_j) \gg 3$, $\forall\ a_j \in A$. If this is not the case, then we may scale B and $s(a_j)$, $\forall\ a_j \in A$, by a constant without affecting the 3-Partition Problem. Given an instance of the 3-Partition Problem, we construct the following instance of $1|chain; pmnt; l_{j,k} = l|\sum C_j$. There are $3q+2$ chains.

For each $a_j \in A$ there is one chain C_j containing $q+1$ tasks, $C_{(j,1)}, C_{(j,2)}, \ldots, C_{(j,(q+1))}$. The processing times for $C_{(j,1)}$ and $C_{(j,(q+1))}$ are $s(a_j)$. All other processing times for the tasks in C_j are equal to 1.

Two additional chains, X and Y, are created. X contains l^5+2q+1 tasks each with processing time l. Y contains l^5+2q tasks with processing times of

$$p(Y_i) = \begin{cases} 3q-3i+1 & \text{for } 1 \leq i \leq q \\ 3i-3q-2 & \text{for } q+1 \leq i \leq 2q \\ l & \text{for } 2q+1 \leq i \leq l^5+2q \end{cases}$$

All distance constraints are equal. We define the non-zero distance constraint to be

$$l = B + 3q - 2.$$

The target mean flow time for the schedule is

$$z = \sum_{i=1}^{l^5+2q+1} [l(2i-1)] + \sum_{i=2q+1}^{l^5+2q} [2li] + \sum_{i=1}^{2q} [2lif(i)],$$

where $f(i)$ is defined as

$$f(i) = \begin{cases} 4 + [(3q-2) - (3q-3i+1)] & 1 \leq i \leq q \\ 4 + [(3q-2) - (3i-3q-2)] & q+1 \leq i \leq 2q \end{cases}$$

It is easy to verify that this reduction requires time polynomial in the parameters of the 3-Partition problem.

Suppose we have a 'Yes' instance of 3-Partition. A schedule with mean flow time $\leq z$ is constructed as follows. Start the tasks of chain X as soon as possible. Thus,

$$\sum_{i=1}^{l^5+2q+1} C_{X_i} = \sum_{i=1}^{l^5+2q+1} [l(2i-1)].$$

We have l^5+2q intervals, $I_1, I_2, \ldots, I_{l^5+2q}$, each of length l within which the remaining tasks must be scheduled. Chain Y has l^5+2q tasks; therefore, one task of chain Y must be executed during each interval. Furthermore, the distance constraints require that no more than one task of Y may be executed during any one interval. Thus, schedule Y_i in interval I_i. It is easily seen that if the first $2q$ tasks of X and Y are not scheduled as above, then the schedule will have a mean flow time $> z$. After scheduling chain Y, the empty time slots in each interval I_i are characterized by the following function.

$$empty(I_i) = \begin{cases} B+3(i-1) & 1 \leq i \leq q \\ B+3(2q-i) & q+1 \leq i \leq 2q \\ 0 & 2q+1 \leq i \leq l^5+2q \end{cases}$$

By assumption of a 'Yes' instance of 3-Partition, there exists q disjoint 3-task sets, $H_1, H_2, \ldots, H_q$, with processing time of B and comprised of the first task from the chains C_j, $1 \leq j \leq 3q$. Schedule H_k in interval I_k, $1 \leq k \leq q$. Consider task $C_{(j,1)}$ scheduled in interval I_i ($i \leq q$ due to how we scheduled the sets H_k). The tasks $C_{(j,2)}$, $C_{(j,3)}$, $\ldots$, $C_{(j,q)}$ can be scheduled during the next $q-1$ intervals. The additional empty time slots left by chain Y allow these 'pass-through' tasks to be scheduled in these intervals. By scheduling them thus, $C_{(j,(q+1))}$ can be scheduled in the interval I_{i+q}. The resulting schedule is feasible with a mean flow time $\leq z$.

Conversely, suppose that we have a schedule with a mean flow time $\leq z$. As before, the tasks in chain X must be scheduled as soon as possible, and exactly one task from chain Y must be scheduled in each interval I_i, $1 \leq i \leq l^5 + 2q$. The tasks from the C_j chains are scheduled without idle times in the remaining time slots of the intervals. If this is not the case, then there must be at least one task, T_k, with completion time $C_k \geq (l^5 + 2q + 1)B \geq \sum_{i=1}^{2q}[2lif(i)]$ contradicting our assumption on the mean flow time of the schedule. Furthermore, due to the distance constraints and lengths of the chains C_j, the first tasks from these chains, $C_{(j,1)}$, $1 \leq j \leq 3q$, must be scheduled in the first q intervals I_i, $1 \leq i \leq q$, and the last tasks from these chains, $C_{(j,(q+1))}$, $1 \leq j \leq 3q$, must be scheduled in the last q intervals I_i, $q+1 \leq i \leq 2q$.

Consider the first interval I_1 and only the first tasks in the chains C_j. Let S_1 be the set of $C_{(j,1)}$ tasks that are started and finished in I_1. Task Y_1 requires $3q-2$ time units in I_1 leaving B time units to schedule other tasks. Assume that $\sum_{C_{(j,1)} \in S_1} p_j = B - c$ for some $c > 0$. Consider the interval I_{q+1}. Y_{q+1} has an execution time of 1. There are at most $3q$ unit execution time tasks from the chains C_j that may be scheduled in this interval. The only additional tasks that may be scheduled in this time interval are the final tasks of the chains C_j whose first task was scheduled in the first interval I_1 such that $C_{(j,1)} \in S_1$. $|S_1| \leq 3$ by the constraints in the 3-Partition Problem, and c is characterized by the following function due to our assumptions on B and $s(a_j), a_j \in A$.

$$c \geq \begin{cases} B & \text{if } |S_1| = 0 \\ B/2 + 1 & \text{if } |S_1| = 1 \\ 2 & \text{if } |S_1| = 2 \\ 3 & \text{if } |S_1| = 3 \end{cases}$$

Therefore, there are $c - 3 + |S_1| > 0$ idle time slots in interval I_{q+1}. It follows that our assumption is incorrect, and $\sum_{C_{(j,1)} \in S_1} p_j = B$. An iterative application of this argument leads to the identification of sets S_j with $\sum_{C_{(j,1)} \in S_i} p_j = B$ for $1 \leq i \leq q$ and to the conclusion that we have a 'Yes' instance of 3-Partition. ■

2.3 $\mathbf{1|chain; p_j = 1; l_{j,k} \in \{0, l\}|C_{max}, \sum C_j}$

The strong $\mathcal{NP}$-completeness results for the preemptive version of the problem suggests that if the problem is restricted to have integral processing times and distance constraints, then preemptions may occur at integral boundaries only without affecting the complexity of the problem. The following lemma formalizes this observation.

Lemma 2.1 *If all input parameters are integral valued, then there exists an optimal solution to* $1|\beta_1;\, pmtn;\, \beta_3;\, \beta_5 \mid C_{\max}, \sum C_j$ *such that all preemptions occur at integral time boundaries.*

Proof We will prove this lemma by constructing an optimal solution S' containing preemptions only at integral time boundaries from an optimal solution S that may have preemptions at non-integral time boundaries. The solution S' will be constructed by iteratively 'fixing' a unit of execution time from a single task into a time slot. Once a unit of execution time is 'fixed' at a particular time slot it will remain at that time slot for the remainder of the iterations.

Let S be an optimal solution to a problem in $1|\beta_1;\, pmtn;\, \beta_3;\, \beta_5 \mid C_{\max}, \sum C_j$. Let t_i be the first time slot in S that has more than one task scheduled in it. Fix the schedule through time slot t_{i-1}. Considering only non-fixed task segments, let task C

be the first task to complete an integral amount of processing after time t_i in S. Let $t_j > t_i$ be the time slot in which this unit of C completes. We may move this unit of processing time of C to time slot t_k where $t_i \leq t_k \leq t_j$ and t_k is the earliest time slot not already containing a fixed unit of processing time in which this unit of C may be scheduled. Fix this unit of C in time slot t_k. Note that due to distance constraints t_k may not be equal to t_i.

All non C tasks that were scheduled in the interval $[t_k, t_j]$ in S may be 'compressed' into the interval $[t_k+1, t_j]$ where the compression does not change the scheduled time of fixed task segments. The compression maintains the ordering of the compressed tasks in the schedule, and it does not change the completion times of any task in the schedule S except possibly decreasing the completion time of task C. Thus, the modified schedule is still optimal.

By iteratively finding a time slot t_i and a task C, we construct the schedule S' that contains preemptions only at integral boundaries, and the lemma is proven. ■

The complexity results for the makespan scheduling problem containing chains, unit execution time tasks, and all distance constraints equal to either zero of l for some l input to the problem follow directly from Theorem 2.3 and Lemma 2.1.

Theorem 2.5 $1|chain; p_j = 1; l_{j,k} \in \{0, l\}|C_{\max}$ *is $\mathcal{NP}$-complete in the strong sense.*

Proof Follows from Theorem 2.3 and Lemma 2.1. ■

Likewise, the complexity for the mean weighted flow time objective follows from Theorem 2.4 and Lemma 2.1.

Theorem 2.6 $1|chain; p_j = 1; l_{j,k} \in \{0, l\} \mid \sum w_j C_j$ *is $\mathcal{NP}$-complete in the strong sense.*

Proof Follows from Theorem 2.4 and Lemma 2.1. ■

However, we can do better than this. The complexity for the mean flow time objective function follows from the complexity for the makespan when all processing times are equal to one.

Theorem 2.7 $1|chain; p_j = 1; l_{j,k} \in \{0, l\}|\sum C_j$ *is $\mathcal{NP}$-complete in the strong sense.*

Proof We restrict the problem to instances where the optimum schedule S_{opt} contains no idle time. The sum of the completion times for S_{opt} is $\sum_{i=1}^{C_{\max}} i = \sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$. Solving for the optimum sum of completion times is equivalent to solving for the optimum makespan. The theorem then follows from Theorem 2.5. ■

2.4 $\mathbf{1|chain; p_j \in \{1,2\}; l_{j,k} = L\ (L \geq 2)|C_{max}, \sum C_j}$

We now turn our attention to the case when the values of the distance constraints are no longer part of the input to the problem, but instead are a fixed value. We consider the problem when all distance constraints are equal to a fixed value L. When $L = 0$, $1|\beta_1; \beta_2; \beta_3; l_{i,j} = 0|C_{\max}, \sum C_j$ are solvable in polynomial time. When $L = 1$, $1|\beta_1; \beta_2; \beta_3; l_{i,j} = 1|C_{\max}$ are solvable in polynomial time; however, the complexity of many of these problems is unknown for the sum of completion time objective function $\sum C_j$. We consider problems for which $L \geq 2$ and $\beta_1 = chain$. We further restrict the problem to contain only tasks with processing times of either 1 or 2. In other words, we examine the complexity for $1|chain; p_j \in \{1,2\}; l_{j,k} = L\ (L \geq 2)|C_{\max}, \sum C_j$. We prove that $1|chain; p_j \in \{1,2\}; l_{j,k} = L\ (L \geq 2)|C_{\max}, \sum C_j$ are strongly $\mathcal{NP}$-complete problems. These results are stronger than the results from Section 2.1.

Theorem 2.8 $1|chain; p_j \in \{1,2\}; l_{j,k} = L\ (L \geq 2)|C_{\max}$ *is $\mathcal{NP}$-complete in the strong sense.*

Proof This problem is clearly in $\mathcal{NP}$. To prove that it is strongly $\mathcal{NP}$-hard we reduce the 3-Partition Problem to it.

Given an instance of the 3-Partition Problem, we construct an instance of the $1|chain; p_j \in \{1,2\}; l_{j,k} = L\ (L \geq 2)|C_{\max}$ problem as follows. There are $[(6+B)q+1](L-2)+3q(3+B)+1$ tasks. For each $a_j \in A$, there is a chain C_j consisting of $s(a_j)+1$ tasks, $C_{j,1} \prec C_{j,2} \prec \cdots \prec C_{j,s(a_j)+1}$, with processing times $p_{j,k} = 1$, $1 \leq k \leq s(a_j)$, and $p_{j,s(a_j)+1} = 2$.

L additional chains, X_i, $1 \leq i \leq L-1$, and Y, are created. X_i, $1 \leq i \leq L-1$, contains $(6+B)q+2$ tasks each with processing time 1. Y contains $(3+B)q+1$ tasks with processing times of

$$p(Y_i) = \begin{cases} 2 & i = 0 \\ 1 & j(3+B)+1 \leq i \leq j(3+B)+B, 0 \leq j \leq q-1 \\ 2 & j(3+B)+B+1 \leq i \leq j(3+B)+B+3, 0 \leq j \leq q-1. \end{cases}$$

All distance constraints are equal to a constant $L \geq 2$.

We define $z = [(6+B)q+1]L + L - 1$. Note that the sum of the processing times of all tasks is equal to z; therefore, any schedule that completes by time z must not have any idle time. Figure 6 illustrates how the template chains X_i and the enforcer chain Y create a template within which the C_j chains must be scheduled. It is easy to verify that this reduction requires time polynomial in the parameters of the 3-Partition problem.

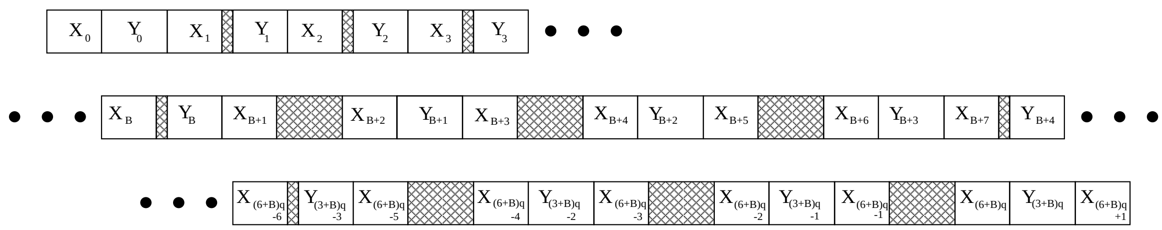


Figure 6: Template formed from the X_i chains and the Y chain in the proof of Theorem 2.8. X_k in the figure corresponds to the set of tasks $X_{i,k}$, $1 \leq i \leq L-1$. The shaded regions indicate unused processing times after chains X_i, $1 \leq i \leq L-1$, and Y have been scheduled.

Suppose that there exists a 'Yes' solution to the 3-Partition Problem. Schedule the tasks of chains X_i, $1 \leq i \leq L-1$, as soon as possible. $X_{i,(6+B)q+2}$ finish at times $[z-L+1, z]$. We have $(6+B)q+1$ intervals, $I_1, I_2, \ldots, I_{(6+B)q+1}$, each of length 2 within which the remaining tasks can be scheduled. Chain Y has $(3+B)q+1$ tasks; however, due to the processing times of the tasks in X_i and the distance constraints there are $3q$ intervals during which a task from Y cannot be scheduled, e.g., the interval between tasks $X_{i,B+6}$ and $X_{i,B+7}$ $\forall i$. Therefore, we must schedule the tasks of Y as early as possible. After scheduling chain Y the empty time slots in each interval I_i are characterized by the following function

$$
empty(I_i) = \begin{cases} 0 & i = \begin{cases} 1 \\ j(B+6)+B+3 & 0 \le j \le q-1 \\ j(B+6)+B+5 & 0 \le j \le q-1 \\ j(B+6)+B+7 & 0 \le j \le q-1 \end{cases} \\ \\ 1 & i = k + j(B+6)+1, \ 1 \le k \le B, \ 0 \le j \le q-1 \\ \\ 2 & i = \begin{cases} j(B+6)+B+2 & 0 \le j \le q-1 \\ j(B+6)+B+4 & 0 \le j \le q-1 \\ j(B+6)+B+6 & 0 \le j \le q-1 \end{cases} \end{cases}
$$

By assumption of a 'Yes' instance of 3-Partition, there exists q disjoint 3-element sets, $H_1, H_2, \ldots, H_q$, with each element a_j corresponding to chain C_j. Schedule the corresponding chains of the elements in H_1 during the first $3+B$ non-full intervals. Note that of these $3+B$ intervals, each of the first B intervals has one idle time unit and each of the last three intervals has 2 idle time units. The tasks corresponding to H_1 consist of B tasks with $p_j = 1$ and three tasks with $p_j = 2$. The precedence constraints are such that the tasks corresponding to H_1 may be scheduled in the first $3+B$ non-full intervals. (A non-full interval is an interval that contains a non-zero amount of unused processing time after all chains X_i and Y are scheduled.) Similarly, the tasks corresponding to H_2 may be scheduled in the next $3+B$ non-full intervals, and so on. The resulting schedule is feasible and has a makespan of z.

Conversely, suppose that we have a schedule of length z. As before, the tasks in chains X_i, $1 \le i \le L-1$, must be scheduled as soon as possible, and the tasks from chain Y must be scheduled as soon as possible within the $(6+B)q+1$ intervals. The tasks from the C_j chains are scheduled without idle times in the remaining time slots of the schedule. Furthermore, tasks with an execution time of 1 are only scheduled in intervals containing a task of Y with an execution time of 1, and tasks with an execution time of 2 are only scheduled in intervals that do not contain a task from Y.

Consider the first $3+B$ non-full intervals. Each of the first B intervals contains a unit execution time task in a chain. By assumption on a schedule of length z, intervals I_{B+2}, I_{B+4}, and I_{B+6} each contain a non-Y task with execution time of 2. Since all non-Y tasks with an execution time of 2 are the final tasks in the C chains, there must be three chains C_i, C_j, and C_k, that have their respective first $s(a_i)$, $s(a_j)$, and $s(a_k)$, tasks scheduled in the first B intervals. Therefore, chains C_i, C_j, and C_k are completely scheduled during the first $3+B$ non-full intervals. C_i, C_j, and C_k correspond to set H_1 containing the elements a_i, a_j, and a_k such that $|H_1| = s(a_i)+s(a_j)+s(a_k) = B$. An iterative application of this argument over each of the q sets of $3+B$ non-full intervals leads to the identification of sets H_j with $|H_j| = B$ for $1 \le j \le q$ and to the conclusion that we have a 'Yes' instance of 3-Partition. ■

The complexity proof for the mean flow time objective function is identical to the complexity proof for the makespan objective function. This is because the reduction forces the tasks to have a known sum of completion times. There is no variability in the sum of completion times if there is a 'Yes' solution to the 3-Partition problem. Therefore, by replacing z in the complexity proof for the makespan objective function with the sum of the completion times of chains $X_i, 1 \le i \le L-1$, Y, and C_j, $\forall\ a_j \in A$, the proof remains valid for the mean flow time objective function.

Theorem 2.9 $1|chain; p_j \in \{1,2\}; l_{j,k} = L\ (L \ge 2)|\sum C_j$ *is* $\mathcal{NP}$*-complete in the strong sense.*

p	$1\|chain; p_j = 1; l_{i,j} = l\|C_{\max}$	[Li77]
p	$1\|chain; p_j = 1; l_{i,j} = l\|\sum C_j$	[BK98]
p	$1\|chain; p_j = 1; l_{i,j} \in \{0,1\}\|C_{\max}$	[BG89]
?	$1\|chain; p_j = 1; l_{i,j} \in \{0,1\}\|\sum C_j$	
?	$1\|chain; p_j = 1; l_{i,j} \in \{0,L\}\ (L \geq 2)\|C_{\max}$	
?	$1\|chain; p_j = 1; l_{i,j} \in \{0,L\}\ (L \geq 2)\|\sum C_j$	
$*$	$1\|chain; p_j = 1; l_{i,j} \in \{0,l\}\|C_{\max}$	Section 2.3
$*$	$1\|chain; p_j = 1; l_{i,j} \in \{0,l\}\|\sum C_j$	Section 2.3

(a)

p	$1\|chain; l_{i,j} = 1\|C_{\max}$	[FL96]
?	$1\|chain; l_{i,j} = 1\|\sum C_j$	
?	$1\|chain; pmtn; l_{i,j} = L\ (L \geq 2)\|C_{\max}$	
?	$1\|chain; pmtn; l_{i,j} = L\ (L \geq 2)\|\sum C_j$	
$*$	$1\|chain; pmtn; l_{i,j} = l\|C_{\max}$	Section 2.2
$*$	$1\|chain; pmtn; l_{i,j} = l\|\sum C_j$	Section 2.2
$*$	$1\|chain; p_j \in \{1,2\}; l_{i,j} = L\ (L \geq 2)\|C_{\max}$	Section 2.4
$*$	$1\|chain; p_j \in \{1,2\}; l_{i,j} = L\ (L \geq 2)\|\sum C_j$	Section 2.4

(b)

Table 2: Complexity boundary involving single machine chain scheduling problems and the makespan and mean flow time objective functions.

2.5 Complexity Boundary Analysis involving Chains

We have proven the strong $\mathcal{NP}$-completeness of several single machine problems involving chain structured tasks. By extension, the multiple machine versions of these problems are also strongly $\mathcal{NP}$-complete. These results lead naturally to the question of where is the boundary between polynomial time solvable problems and $\mathcal{NP}$-complete problems.

Table 2 presents the known boundary for the single machine case with respect to the makespan and mean flow time objective functions. Polynomial time solvable problems are denoted by a 'p.' Strongly $\mathcal{NP}$-complete problems are denoted by a '$*$.' And, problems with an unknown complexity are denoted by a '?.' Note that the complexity results obtained in this section are all new minimal $\mathcal{NP}$-complete results.

A graphical representation of this boundary with respect to allowable task processing times and distance constraints is shown in Figure 2. The graph in Part (a) of the figure depicts the boundary for the makespan objective function, and the graph in Part (b) of the figure depicts the boundary for the mean flow time objective function.

We now examine the complexity boundary involving multiple machine problems and problems that do not involve distance constraints. In order to more fully delineate the boundary we present one additional $\mathcal{NP}$-completeness proof for a parallel machine scheduling problem that does not involve distance constraints, namely $Pm|chain; p_i = 1|\sum w_j C_j$ for any fixed m with $m \geq 2$. Du, Leung, and Young proved that $Pm|chain; pmtn|\sum w_j C_j$ is strongly $\mathcal{NP}$-complete by showing that preemption can not improve the mean weighted flow time to $Pm|chain|\sum w_j C_j$ [DLY91]. This result suggests that requiring all processing times to be equal to one time unit, i.e., β_3 is set to $p_j = 1$, does not reduce the complexity of the problem. We prove this to be the case after stating the main results obtained in [DLY91].

p	$P\|chain;p_j = 1\|\sum C_j$	[Hu61]
p	$P\|chain;p_j = 1;l_{i,j} = l\|C_{\max}$	[Li77]
p	$P\|chain;p_j = 1;l_{i,j} = l\|\sum C_j$	Section 3.1
p	$1\|chain;p_j = 1;l_{i,j} \in \{0,1\}\|C_{\max}$	[BRG89]
p	$1\|chain;p_j = 1;\|\sum w_j C_j$	[Law78]
$*$	$1\|chain;p_j = 1;l_{i,j} \in \{0,l\}\|C_{\max}$	Section 2.3
$*$	$1\|chain;p_j = 1;l_{i,j} \in \{0,l\}\|\sum C_j$	Section 2.3
$*$	$1\|chain;p_j = 1;l_{i,j} = 1\|\sum w_j C_j$	[TGS94]
$*$	$P2\|chain;p_j = 1\|\sum w_j C_j$	Section 2.5

Table 3: Complexity boundary involving multiple machine chain scheduling problems and the makespan and mean (weighted) flow time objective functions.

Theorem 2.10 ([DLY91]) *$Pm|chain|\sum w_j C_j$ is strongly $\mathcal{NP}$-complete.*

Theorem 2.11 ([DLY91]) *Preemption cannot reduce the mean weighted flow time for a set of chains.*

Theorem 2.12 ([DLY91]) *$Pm|chain;pmtn|\sum w_j C_j$ is strongly $\mathcal{NP}$-complete.*

We now show that $Pm|\text{chain};p_j = 1|\sum w_j C_j$ is strongly $\mathcal{NP}$-complete.

Theorem 2.13 *$Pm|chain;p_j = 1|\sum w_j C_j$ is strongly $\mathcal{NP}$-complete.*

Proof Consider the problem $Pm|\text{chain;pmtn}|\sum w_j C_j$. Without loss of generality we assume that all tasks in $Pm|\text{chain;pmtn}|\sum w_j C_j$ have integral processing times, p_j. We represent each task $T_j \in \mathcal{T}$ as a chain of p_j unit execution time tasks $C_{j,1} \prec \cdots \prec C_{j,p_j}$. To preserve the original precedence constraints, we add precedence constraints $C_{i,p_i} \prec C_{j,1}\ \forall\ T_i \prec T_j$ in the original problem.

With each task C_{j,p_j} we associate the weight w_j. The weight for all other tasks is set to zero.

This reduction is performed in pseudo-polynomial time since the number of tasks in the reduced problem is equal to $\sum p_j$.

Solving $Pm|\text{chain};p_j = 1|\sum w_j C_j$ clearly finds a feasible schedule to $Pm|\text{chain;pmtn}|\sum w_j C_j$. By Theorem 2.11 the optimum schedule to $Pm|\text{chain};p_j = 1|\sum w_j C_j$ is an optimum schedule to $Pm|\text{chain;pmtn}|\sum w_j C_j$. Therefore, solving $Pm|\text{chain};p_j = 1|\sum w_j C_j$ solves $Pm|\text{chain;pmtn}|\sum w_j C_j$. Thus, $Pm|\text{chain};p_j = 1|\sum w_j C_j$ is strongly $\mathcal{NP}$-complete. ■

Table 3 presents the known boundary involving multiple machine unit execution time chain scheduling problems and makespan, mean flow time, and weighted mean flow time objective functions.

3 Arbitrary Precedence Structured Tasks

We now examine the complexity of several problems involving `tree` and `prec` precedence constraint topologies. We show that for the makespan, $C_{\max}$, and mean flow time, $\sum C_j$, objective functions the problems involving `tree` structured precedence constraint topologies are polynomial-time solvable, and for the problems involving `prec` structured precedence constraint topologies are strongly $\mathcal{NP}$-complete; therefore, they are strongly $\mathcal{NP}$-complete for all of the objective functions shown in Figure 1.

3.1 $1|tree; p_j = 1; l_{j,k} = l|C_{max}, \sum C_j$

We first consider the unit execution time scheduling problem where all distance constraints are equal and the precedence constraints form a `tree`. When the precedence constraint topology is an intree, the makespan problem $P|intree; p_j = 1; l_{j,k} = l|C_{\max}$ is solvable in polynomial time by Hu's algorithm [Li77]. When the precedence constraint topology is an outtree, the makespan problem $P|outtree; p_j = 1; l_{j,k} = l|C_{\max}$ is solvable in polynomial time by using a critical path based algorithm [BJS80].

We show that Li's results [Li77] for the makespan objective function are easily extended to the mean flow time problem $P|intree; p_j = 1; l_{j,k} = l|\sum C_j$. We also show that the proof of optimality for the makespan problem for outtrees given by Bruno et. al. [BJS80] is valid for the mean flow time problem $P|outtree; p_j = 1; l_{j,k} = l|\sum C_j$.

3.1.1 $P|intree; p_j = 1; l_{j,k} = l|\sum C_j$

We prove that the problem $P|intree; p_j = 1; l_{j,k} = l|\sum C_j$ is solvable in polynomial-time by Hu's algorithm [Hu61]. Our proof of optimality closely follows that of Li [Li77] for the makespan objective function.

Hu's algorithm may be simply stated as: at each time step schedule those ready tasks with the largest label. The label of task T_j, $l(T_j)$, is defined to be $l(T_j) = N_j + 1$, where N_j is the label of the immediate successor of task T_j. The label of a task with no immediate successor is 1.

We first make an observation on the structure of an optimum schedule.

Lemma 3.1 *In an optimum schedule to $P|tree; p_j = 1; l_{j,k} = l|\sum C_j$, no machine is left idle during a time slot in which a task is ready to be scheduled.*

Proof By contradiction. Assume that we have an optimum schedule that contains an idle time slot, t, during which a task T_j was ready to execute. Since T_j is ready during time slot t but scheduled at some time greater than t, we may reschedule T_j in time slot t without affecting the feasibility of the schedule and without rescheduling any other tasks. The rescheduled completion time of T_j is less than its original value. Thus, we have found a schedule with a smaller mean flow time contradicting our assumption of an optimum schedule. ∎

The proof of the following lemma is identical to the one provided by Li [Li77]. It is repeated here for completeness.

Lemma 3.2 *At any time t, in generating a schedule for $P|tree; p_j = 1; l_{j,k} = l|\sum C_j$, for the remaining tree of n nodes to be optimally scheduled, those tasks with the highest labels should be scheduled first.*

Proof Let $s(j)$ be the scheduled time of task T_j. $E(t)$ is defined to be the set of tasks scheduled at time t. $s_{j,1}$ is the immediate successor of task T_j, and $s_{j,i}$ is the immediate successor of task $s_{j,i-1}$.

By induction. For $n = 1$, the lemma trivially holds. Assume that the lemma is true for $n \leq k - 1$. Then, for $n = k$ suppose there exists an optimum schedule where there is a ready task at time t, $T_j \notin E(t)$, but $l(T_j) > l(T_i)$ for some $T_i \in E(t)$. In case there is more than one such T_i, choose the one with the lowest label value.

By our induction hypothesis $T_j \in E(t+1)$. If $s_{i,1} \in E(t+\delta)$ for some $\delta > l$, then T_j and T_i may be interchanged in the schedule without affecting feasibility or optimality.

If $s_{i,1} \in E(t+l)$, then $s_{j,1} \in E(t+l+\delta)$ for some $\delta \geq 1$. Then $\exists\, \gamma \in E(t+l+1)$ such that γ is either not dependent on any node $\in E(t+1)$ or $s_{j,1}$ where $s_{j,1}$ is not dependent on any node $\in E(t+1)$ except T_j. This is because at l time units away, there is one slot corresponding to each slot at time $t+1$. Note that γ can not be an empty time slot. If it were, then we could interchange T_j and T_i and interchange $s_{i,1}$ and γ. The resulting schedule would have a lower mean flow time contradicting our assumption on the optimality of the original schedule.

We may now interchange T_j and T_i and interchange $s_{i,1}$ and γ. We repeat this analysis for $s_{i,2}, s_{i,3}, \ldots$ until no interchange is required. Since $l(T_j) > l(T_i)$, an interchange is always feasible until $s_{i,a} \in E(t+al+\delta)$ for some a and $\delta \geq 1$. Then, no further interchange is required. Observe that $s_{i,x} = s_{j,y}$ for some x and some y with $y > x$. The interchange will always terminate at some point because $s(s_{i,x-1}) < s(s_{j,y-1})$. ■

Theorem 3.1 *Hu's algorithm optimally schedules instances of the* $P|intree; p_j = 1; l_{j,k} = l|\sum C_j$ *problem.*

Proof Follows from Lemma 3.1 and Lemma 3.2. ■

3.1.2 $\mathbf{P|outtree; p_j = 1; l_{j,k} = l|\sum C_j}$

Bruno et. al. [BJS80] proved that a critical path based algorithm optimally solves the problem $P|outtree; p_j = 1; l_{j,k} = l|C_{\max}$. The critical path algorithm is similar to Hu's algorithm and may be simply stated as: at each time step schedule those ready tasks with the largest label. The label of task T_j, $l(T_j)$, is defined to be $l(T_j) = \max\{N_{j_1}, \ldots, N_{j_k}\} + 1$, where $N_{j_1}, \ldots, N_{j_k}$ are the labels of the immediate successors of task T_j. The label of a task with no immediate successors is 1.

The proof of optimality relies on the analysis of congestion regions, i.e., regions where more than m tasks are ready to execute during a time slot, where m is the number of machines. Bruno et. al. were able to prove that tasks that are not scheduled in a congestion region are scheduled as early as possible, i.e., given an infinite number of processors, these tasks could not be scheduled any earlier. They also proved that congestion regions do not contain any idle time slots, except possibly during the last time slot of the region. Finally, they proved that a critical path schedule during one congestion region does not induce idle time slots in a critical path schedule during another congestion region. Consequently, the proof given in [BJS80] is valid for the following theorem and is not repeated here.

Theorem 3.2 *The critical path algorithm optimally schedules instances of the* $P|outtree; p_j = 1; l_{j,k} = l|\sum C_j$ *problem.*

3.2 $\mathbf{1|prec; p_j = 1; l_{j,k} = l|C_{max}, \sum C_j}$

We now consider the problems in the previous section but with arbitrary precedence constraints, namely $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ and $1|prec; p_j = 1; l_{j,k} = l|\sum C_j$. Unfortunately, allowing arbitrary precedence constraints makes the problems strongly $\mathcal{NP}$-complete.

We reduce the 3-Partition problem to $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ in a manner similar to the reductions used for chain structured tasks. Precedence constraints are used to create a specific template structure in a similar manner to the way the zero distance constraints and arbitrary execution times are used in the reductions involving chain structured tasks.

Theorem 3.3 $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ *is* $\mathcal{NP}$*-complete in the strong sense.*

Proof $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ is clearly in $\mathcal{NP}$. To prove that it is also strongly $\mathcal{NP}$-hard, we reduce the 3-Partition Problem to it.

Given an instance of the 3-Partition Problem, we construct the following instance of $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$. We define the non-zero distance constraint to be

$$l = B + 3q - 3.$$

There are $l(4q+1) + 4q + 2$ tasks arranged in $3q+1$ separate dags. For each $a_i \in A$ there is one dag D_i containing $2s(a_i) + q - 1$ tasks, $D_{(i,1)}$, $D_{(i,2)}$, $\ldots$, $D_{(i,(2s(a_i)+q-1))}$. The precedence constraints within D_i are defined as follows. $D_{(i,j)} \prec D_{(i,s(a_i)+1)}$, $1 \leq j \leq s(a_i)$. $D_{(i,j)} \prec D_{(i,+1)}$, $s(a_i) + 1 \leq j \leq s(a_i) + q - 1$. $D_{(i,s(a_i)+q-1)} \prec D_{(i,j)}$, $s(a_i) + q \leq j \leq 2s(a_i) + q - 1$.

One additional dag, X, is created. X contains $l(2q+1)+4q+2+2\sum_{i=1}^{q}(l-B-3(i-1))$ tasks. The central feature of X is a chain of $4q+2$ tasks, $X_1, X_2, \ldots, X_{4q+2}$. All other tasks of X are connected to at least one of these chain tasks and most are connected to two of the chain tasks. A non-chain task has precedence relations with chain tasks only. For simplicity in the definition of the remaining precedence constraints we note a non-chain task of X as Y_j. We also use the following invariant. If it is stated that chain task X_i precedes non-chain task Y_j, $X_i \prec Y_j$, then it is also true that Y_j precedes X_{i+3}, $Y_j \prec X_{i+3}$, if X_{i+3} exists.

The remaining precedence constraints in X are defined as follows. $Y_{0,j} \prec X_3$, $1 \leq j \leq l$. $X_i \prec Y_{i,j}$, $1 \leq j \leq l$, $1 \leq i \leq 4q$, and i is even. $X_i \prec Y_{i,j}$, $1 \leq j \leq l - B - 3(o(i) - 1)$, $1 \leq i \leq 2q$, i is odd, and $o(i)$ returns the index of i in the list of odd numbers greater than zero, e.g., $o(1) = 1$, $o(3) = 2$, $o(5) = 3$, $o(7) = 4$. $X_i \prec Y_{i,j}$, $1 \leq j \leq l - B - 3(q - o(i - 2q))$, $2q + 1 \leq i \leq 4q$, i is odd, and $o(i)$ is defined as above.

All distance constraints are equal to l.

The deadline for the schedule is $z = l(4q+1)+4q+2$. Note that the processing times of all the tasks is equal to z; thus, any schedule that completes before the deadline must not have any idle time. It is easy to verify that this reduction requires time polynomial in the parameters of the 3-Partition problem.

Suppose we have a 'Yes' instance of 3-Partition. A schedule of length z is constructed as follows. Start the tasks of chain X as soon as possible. $X_{(4q+2)}$ finishes at time z. The remaining $Y_{i,j}$ tasks are constrained to be scheduled in the time interval between task X_{i+1} and task X_{i+2}. The resultant template has $2q$ non-full intervals, $I_1, I_2, \ldots, I_{2q}$, within which the remaining tasks must be scheduled. These intervals occur between tasks X_i and X_{i+1} for $1 \leq i \leq 4q$ and i even. The number of empty time slots in each interval I_i are characterized by the following function.

$$empty(I_i) = \begin{cases} B + 3(i-1) & 1 \leq i \leq q \\ B + 3(2q - i) & q + 1 \leq i \leq 2q \end{cases}$$

By assumption of a 'Yes' instance of 3-Partition, there exists q disjoint B-task sets, H_1, H_2, $\ldots$, H_q, with processing time of B and comprised of the first $s(a_i)$ tasks from the dags D_i, $1 \leq i \leq 3q$. Schedule H_k in interval $I_k, 1 \leq k \leq q$. Consider tasks $D_{(i,1)}$, $D_{(i,2)}$, $\ldots$, $D_{(i,s(a_i))}$ scheduled in interval I_j ($j \leq q$ due to how we scheduled the sets H_k). The tasks $D_{(i,s(a_i)+1)}$, $D_{(i,s(a_i)+2)}$, $\ldots$, $D_{(i,s(a_i)+q-1)}$ can be scheduled during the next $q-1$ intervals. The additional empty time slots above the B needed to schedule H_j allow these 'pass-through' tasks to be scheduled in these intervals. By scheduling them thus, The tasks $D_{(i,s(a_i)+q)}$, $D_{(i,s(a_i)+q+1)}$, $\ldots$, $D_{(i,2s(a_i)+q-1)}$ can be scheduled in the interval I_{j+q}. The resulting schedule is feasible with a makespan of z.

Conversely, suppose that we have a schedule of length z. As before, the tasks in chain X must be scheduled as soon as possible, and the remaining $Y_{i,j}$ tasks are constrained to be scheduled in the time interval between task X_{i+1} and task X_{i+2}. The resultant template has $2q$ non-full intervals, $I_1, I_2, \ldots, I_{2q}$, within which the remaining tasks are scheduled. The tasks from the D_i dags are scheduled without idle times in the remaining time slots of the intervals. Furthermore, due to the distance constraints and lengths of the chains $D_{(i,s(a_i)+1)} \prec D_{(i,s(a_i)+2)} \prec \cdots \prec D_{(i,s(a_i)+q-1)}$, the first $s(a_i)$ tasks from these dags, D_i, $1 \leq i \leq 3q$, must be scheduled in the first q intervals I_j, $1 \leq j \leq q$, and the last $s(a_i)$ tasks from these dags must be scheduled in the last q intervals I_j, $q+1 \leq j \leq 2q$.

Consider the first interval I_1 and only the first $s(a_i)$ tasks in the dags D_i. Let S_1 be the set of dags D_i that have their first $s(a_i)$ tasks, $D_{(i,1)}, D_{(i,2)}, \ldots, D_{(i,s(a_i))}$, scheduled in I_1. The template dag leaves B time units to schedule other tasks in I_1. Assume that $\sum_{D_i \in S_1} s(a_i) = B - c$ for some $c > 0$. Consider the interval I_{q+1}. The template dag leaves $B + 3q - 3$ time units to schedule other tasks in I_{q+1}. There are at most $3q$ tasks, $D_{(i,j)}$, $s(a_i)+1 \leq j \leq s(a_i)+q-1$, from the dags D_i that may be scheduled in this interval. The only additional tasks that may be scheduled in this time interval are the final $s(a_i)$ tasks of the dags D_i whose first $s(a_i)$ tasks were scheduled in the first interval I_1 such that $D_i \in S_1$. $|S_1| \leq 3$ by the constraints in the 3-Partition Problem, and c is characterized by the following function due to our assumptions on B and $s(a_i)$, $a_i \in A$.

$$c \geq \begin{cases} B & \text{if } |S_1| = 0 \\ B/2+1 & \text{if } |S_1| = 1 \\ 2 & \text{if } |S_1| = 2 \\ 3 & \text{if } |S_1| = 3 \end{cases}$$

Therefore, there are $c - 3 + |S_1| > 0$ idle time slots in interval I_{q+1}. It follows that our assumption is incorrect, and $\sum_{D_i \in S_1} s(a_i) = B$. An iterative application of this argument leads to the identification of sets S_j with $\sum_{D_i \in S_j} s(a_i) = B$ for $1 \leq j \leq q$ and to the conclusion that we have a 'Yes' instance of 3-Partition. ■

We may assume without loss of generality that the optimum solution to $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ does not contain any idle times. Since this problem involves unit execution time tasks, the mean flow time to the optimum schedule for $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ is $\sum C_j = \sum_{j=1}^{C_{\max}} j = \sum_{j=1}^{n} j$. This is the smallest mean flow time that any schedule may have. Therefore, finding a schedule to $1|prec; p_j = 1; l_{j,k} = l|\sum C_j$ with mean flow time of $\sum_{j=1}^{n} j$ finds an optimum schedule to $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$, and $1|prec; p_j = 1; l_{j,k} = l|\sum C_j$ is strongly $\mathcal{NP}$-complete. This result is formalized in the following theorem.

Theorem 3.4 *$1|prec; p_j = 1; l_{j,k} = l|\sum C_j$ is $\mathcal{NP}$-complete in the strong sense.*

Proof We reduce the known strongly $\mathcal{NP}$-complete problem $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ to $1|prec; p_j = 1; l_{j,k} = l|\sum C_j$. Without loss of generality we assume that the optimum schedule, S_{opt}, to $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ does not contain any idle time slots. The mean flow time for S_{opt} is $\frac{1}{n}\sum_{j=1}^{C_{\max}} j = \frac{1}{n}\sum_{j=1}^{n} j$.

Given an instance of $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$, we assign a weight of one to all tasks. Let $y = \sum_{j=1}^{n} j$. The optimum solution to $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$ is a solution to $1|prec; p_j = 1; l_{j,k} = l|\sum C_j$ such that $\sum_{j=1}^{n} C_j \leq y$. Conversely, any solution to $1|prec; p_j = 1; l_{j,k} = l|\sum C_j$ such that $\sum_{j=1}^{n} C_j \leq y$ is clearly an optimum solution to $1|prec; p_j = 1; l_{j,k} = l|C_{\max}$. The theorem follows. ■

p	$P\|intree; p_j = 1; l_{i,j} = l\|C_{\max}$	[Li77]
p	$P\|outtree; p_j = 1; l_{i,j} = l\|C_{\max}$	[BJS80]
p	$P\|intree; p_j = 1; l_{i,j} = l\|\sum C_j$	Section 3.1
p	$P\|outtree; p_j = 1; l_{i,j} = l\|\sum C_j$	Section 3.1
p	$1\|prec; p_j = 1; l_{i,j} \in \{0, 1\}\ \|C_{\max}$	[BRG89]
?	$1\|prec; p_j = 1; l_{i,j} \in \{0, 1\}\ \|\sum C_j$	
?	$1\|prec; p_j = 1; l_{i,j} = L(L \geq 2)\ \|C_{\max}$	
?	$1\|prec; p_j = 1; l_{i,j} = L(L \geq 2)\ \|\sum C_j$	
*	$1\|prec; p_j = 1; l_{i,j} = l\|C_{\max}$	Section 3.2
*	$1\|prec; p_j = 1; l_{i,j} = l\|\sum C_j$	Section 3.2

(a)

p	$1\|prec; l_{i,j} = 1\|C_{\max}$	[FL96]
?	$1\|prec; l_{i,j} = 1\|\sum C_j$	
*	$1\|chain; p_j \in \{1, 2\}; l_{i,j} = L\ (L \geq 2)\|C_{\max}$	Section 2.4
*	$1\|chain; p_j \in \{1, 2\}; l_{i,j} = L\ (L \geq 2)\|\sum C_j$	Section 2.4

(b)

Table 4: Complexity boundary involving scheduling problems and the makespan and mean flow time objective functions.

3.3 Complexity Boundary Analysis

In this section we have examined unit execution time problems involving non-chain structured tasks and constant distance constraints. Although allowing more general precedence constraints than simple chains does, in general, make the scheduling problems more difficult, the more general precedence constraints do not necessarily make the problem $\mathcal{NP}$-complete as we saw in Section 3.1.

Table 4 presents the known boundary for unit execution time scheduling problems involving various precedence constraints with respect to the makespan and mean flow time objective functions. Note that the results obtained in this section are either minimal $\mathcal{NP}$-complete results or maximal polynomial-time solvable results. Figure 3 depicts this boundary graphically.

4 Conclusions

We have presented several new complexity results for scheduling problems involving distance constraints. This work sharpens the boundary between known polynomial time solvable scheduling problems and known $\mathcal{NP}$-complete scheduling problems; however, there are still several open problems of practical interest. The $1|\beta_1; p_j = 1; l_{i,j} \in \{0, L\}|C_{\max}$ problem corresponds to code scheduling on a single pipelined processor where some instructions have a fixed latency L and others have zero latency (corresponding to full bypass circuitry for some, but not all, instructions). Even though bypass circuitry can decrease the complexity of the code scheduling problem, it is not always practical to add this extra hardware to the processor design. This is particularly true in Digital Signal Processors (DSPs). DSPs typically do not contain bypass logic; thus, code scheduling for these DSPs corresponds to solving the $1|prec; p_j = 1; l_{i,j} = L\ (L \geq 2)|C_{\max}$ scheduling problem.

References

[BG89] David Bernstein and Izidor Gertner. Scheduling expressions on a pipelined processor with a maximal delay of one cycle. *ACM Transactions on Programming Languages and Systems*, 11(1):57–66, January 1989.

[BHH97] Peter Brucker, Thomas Hilbig, and Johann Hurink. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. Technical Report Reihe P, Nr. 179, Universitat Osnabruck, Osnabrucker Schriften zur Mathematik, 1997.

[BJS80] John Bruno, John W. Jones, III, and Kimming So. Deterministic scheduling with pipelined processors. *IEEE Transactions on Computers*, C-29(4):308–316, April 1980.

[BK98] Peter Brucker and Sigrid Knust. Complexity results for single-machine problems with positive finish-start time-lags. Technical Report Reihe P, Heft 202, Universitat Osnabruck, Osnabrucker Schriften zur Mathematik, 1998.

[BLV95] Egon Balas, Jan Karel Lenstra, and Alkis Vazacopoulos. The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41(1):94–109, 1995.

[BRG89] David Bernstein, Michael Rodeh, and Izidor Gertner. Approximation algorithms for scheduling arithmetic expressions on pipelined machines. *Journal of Algorithms*, 10:120–139, 1989.

[DLY91] Jianzhong Du, Joseph Y-T. Leung, and Gilbert H. Young. Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92:219–236, 1991.

[FL96] L. Finta and Z. Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics*, 70:247–266, 1996.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[GLLR79] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[Hu61] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9:841–848, 1961.

[Law78] Eugene L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75–90, 1978.

[Li77] Hon F. Li. Scheduling trees in parallel/pipelined processing environments. *IEEE Transactions on Computers*, C-26(11):1101–1112, 1977.

[TGS94] V.S. Tanaev, V.S. Gordon, and Y.M. Shafransky. *Scheduling Theory. Single-Stage Systems.* Kluwer Academic Publishers, Boston, USA, 1994.