Computer Science • 24(1) 2023 https://doi.org/10.7494/csci.2023.24.1.4656

Debasis Dwibedy
Rakesh Mohanty

A NOTE ON HARDNESS OF MULTIPROCESSOR SCHEDULING WITH SCHEDULING SOLUTION SPACE TREE

Abstract *We study the hardness of the non-preemptive scheduling problem of a list of independent jobs on a set of identical parallel processors with a makespan minimization objective. We make a maiden attempt to explore the combinatorial structure of the problem by introducing a scheduling solution space tree (SSST) as a novel data structure. We formally define and characterize the properties of SSST through our analytical results. We show that the multiprocessor scheduling problem is $\mathcal{NP}$-complete with an alternative technique using SSST and weighted scheduling solution space tree (WSSST) data structures. We propose a non-deterministic polynomial-time algorithm called magic scheduling (MS) based on the reduction framework. We also define a new variant of multiprocessor scheduling by including the user as an additional input parameter, which we called the multiuser multiprocessor scheduling problem (MUMPSP). We also show that MUMPSP is $\mathcal{NP}$-complete. We conclude the article by exploring several non-trivial research challenges for future research investigations.*

Keywords combinatorial structure, computational complexity, hardness, makespan, multiprocessor scheduling, multiuser, $\mathcal{NP}$-completeness, nondeterministic algorithm, reduction, scheduling solution space tree

Citation Computer Science 24(1) 2023: 53–74

1. Introduction

The non-preemptive *multiprocessor scheduling problem (MPSP)* deals with the scheduling of a list of jobs on a set of identical parallel processors in order to minimize the completion time of a job that finishes last in a schedule; i.e., makespan [19]. We study the combinatorial structure and hardness of MPSP. Our first objective is to develop a strategy to explore the exhaustive solution space of a problem for analyzing the hardness of the computation. Our second objective is to design a non-deterministic algorithmic framework for analyzing offline scheduling algorithms.

Research motivation

MPSP is a typical combinatorial minimization problem where a scheduling decision leads to exponential solution space with increases in the number of jobs and processors. For example, we have m^n feasible schedules for the job list at most in the case of a list of n jobs and $m(\geqslant 2)$ processors. In many scientific studies, authors have attempted to explore the optimal scheduling that incurs the minimum value of makespan. These studies generally have ended up proving the $\mathcal{NP}$-completeness of MPSP – providing a good indication of the hardness of the problem. Nevertheless, the current literature lacks an optimal scheduling algorithm that runs in polynomial time in the lengths of jobs or processors. The following three research questions arise naturally upon investigation into the hardness of the MPSP:

1) What is the formal representation of the combinatorial structure of MPSP?
2) Why is the problem harder to solve in polynomial time even for $m = 2$?
3) How does the solution space affect the complexity class of the problem?

In our current study, we address the above questions. The idea is clear: if a problem has yet to be solved efficiently, we must develop a strategy to explore the intrinsic details (such as the exhaustive solution space of the problem) so that we can precisely define the complexity class; in the future, we would possibly be able to design an optimal scheduling algorithm. Another perspective of our study is related to the $\mathcal{NP}$ proof of MPSP. The first problem that proved to be $\mathcal{NP}$-complete was the 3-SAT problem by Cook [9]. Cook presented a detailed proof of 3-SAT$\in \mathcal{NP}$ and 3-SAT$\in \mathcal{NP}$-hard. Subsequently, several complex combinatorial problems have been shown as $\mathcal{NP}$-complete by the method of reduction from well-known $\mathcal{NP}$-complete problems with less concern toward the $\mathcal{NP}$ proof of these problems. Although many researchers have shown the variants of MPSP $\mathcal{NP}$-complete, there is hardly any attempt to exclusively develop a proof technique for MPSP$\in \mathcal{NP}$. The design of a model with a non-deterministic polynomial-time algorithm for MPSP would possibly pave the way for the development of a deterministic polynomial-time algorithm for all $\mathcal{NP}$-complete problems.

Our contribution

We have developed a *scheduling solution space tree (SSST)* to explore the combinatorial structure with the exhaustive solution space of *MPSP*. The properties of

the SSST have been formally defined and characterized through our analytical results. We have developed a unique technique to show that MPSP $\in \mathcal{NP}$ by mapping the construction of the SSST to a non-deterministic Turing machine that can verify a given *scheduling solution* as a certificate in polynomial time. The SSST is presented as a polynomial-time verifier with its variant named *weighted scheduling solution space tree (WSSST)* by following an interactive proof method. We have shown that MPSP is $\mathcal{NP}$-hard by an alternate yet simple reduction technique. We make a maiden attempt to design a non-deterministic polynomial-time algorithm named *magic scheduling (MS)* for the problem based on the reduction framework. A variant of MPSP is defined by considering the *user* as an additional input parameter and named the problem the *multiuser multiprocessor scheduling problem (MUMPSP)*. We prove that MUMPSP is $\mathcal{NP}$-complete.

Organization

We organize the rest of the paper as follows. Section 2 discusses the foundation of complexity classes and highlights the state-of-the-art literature on the related hardness studies of the variants of the MPSP. Section 3 formally defines and characterizes the *SSST* data structure based on the MPSP. Here, we explore the combinatorial structure and derive some interesting results. Section 4 highlights the computational hardness of MPSP. Here, we propose a general reduction framework and design a non-deterministic polynomial-time algorithm called *magic scheduling (MS)* for the MPSP. We show the computational hardness of MPSP by our proposed SSST and MS algorithm with new analytical results. Section 5 formally defines the MUMPSP problem and proves its complexity class. Section 6 highlights some non-trivial research challenges for future work.

2. Background and state-of-the-art literature

In this section, we highlight and discuss the fundamental aspects of the complexity classes for a basic understanding. We also overview the state-of-the-art literature on the related hardness studies of the variants of the MPSP.

2.1. Complexity classes: preliminaries

The complexity defines the hardness of solving a computational problem. A complexity class identifies a set of problems that are similar in hardness. Cook [9] and Karp [17] were the first to define the basic complexity classes ($\mathcal{P}$ and $\mathcal{NP}$). They also established the formal relationship between $\mathcal{P}$ and $\mathcal{NP}$ in terms of language recognition by the Turing machine ($\mathcal{TM}$). We transform the fundamental idea of defining the complexity classes in Fig. 1 for simplicity in understanding.

We can express any computational problem X as a formal language L, where $L \subseteq \sum^*$ and the $\sum^*$ is a set that contains all strings over any finite alphabet [9]. Furthermore, we can transform a language into a string-recognition problem for the

$\mathcal{TM}$ [17]. Since the $\mathcal{TM}$ can either accept or reject a given $w \in L$ string, we create an instance of X and map it to an equivalent $w^{'}$ string to yield a yes/no answer [17]. The acceptance of $w^{'}$ implies a 'yes' answer. If it is realized by a deterministic $\mathcal{TM}$ in polynomially bounded time with a length of $w^{'}$, then P_1 is considered to be in the $\mathcal{P}$ class [2,9,17]; however, if $w^{'}$ is accepted only by a non-deterministic $\mathcal{TM}$ in polynomial time, then X belongs to the $\mathcal{NP}$ class [9]. By the polynomial time, we mean that the time that is taken for computation is in the form of n^c, where n is the length of the input, and $c(\geqslant 1)$ is any positive number [2]. Informally, a non-deterministic algorithm for X can be considered to be equivalent to that of the corresponding language-recognition algorithm for the non-deterministic $\mathcal{TM}$ [17].

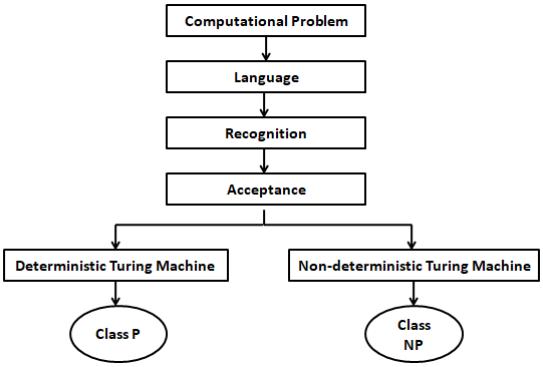

Figure 1. Foundation of Complexity Classes

Since a $\mathcal{TM}$ can either accept or reject a given string, we must represent a computational problem as a language-recognition problem that requires a yes/no answer (the answers 'yes' and 'no' correspond to the acceptance and rejection of a language, respectively). We usually refer to the *decision version* of a problem as a language-recognition problem that requires a yes/no answer [1]. Therefore, to show a problem $\mathcal{NP}$, we generally state its decision version and verify the existence of a solution with a value that is equal to some *threshold* (mostly the optimal value).

Concerning the combinatorial structure of MPSP, we do not require rigorous formal definitions of the complexity classes. In general, we can define the $\mathcal{P}$ class that consists of a set of problems for which polynomial-time algorithms exist, whereas the $\mathcal{NP}$ class covers a set of problems for which non-deterministic polynomial-time algorithms exist. Alternatively, we can define a problem that belongs to the $\mathcal{NP}$ class if the solution to a given instance of a problem is verifiable in polynomial time. We can say clearly that $\mathcal{P} \subset \mathcal{NP}$, but the question remains open, whether $\mathcal{P} = \mathcal{NP}$ [1].

In a milestone paper [9], Cook introduced the $\mathcal{NP}$-complete theory by taking the **satisfiability (SAT)** problem into account. We can formally define the SAT problem as follows.

Given a collection of n clauses $C_1, C_2, \ldots, C_n$, where each clause C_i is a disjunction of the literals from a set $U = \{u_1, u_2, \ldots, u_t, \bar{u_1}, \bar{u_2}, \ldots, \bar{u_t}\}$. Is $\bigwedge_{i=1}^{n} C_i$ satisfiable; i.e., is there a subset $U^{'} \in U$ such that there exists no C_i with complementary pair of literals $(u_j, \bar{u_j})$ in $U^{'}$, and $U^{'} \cap C_i \neq \phi$, $\forall_i$?
The SAT problem where each clause uses three literals is called the **3-SAT** problem.

Theorem 2.1 [9]. *3-SAT is $\mathcal{NP}$-complete*
Cook [9] proved the $\mathcal{NP}$-completeness of the 3-SAT problem by introducing the reduction technique that maps any instance of a problem $X \in \mathcal{NP}$ to an equivalent Boolean expression in CNF; i.e., conjunctive normal form. Thus, a polynomial-time algorithm for the 3-SAT problem could be employed to design a polynomial-time algorithm for any problem $X^{'} \in \mathcal{NP}$. This implies that $\mathcal{P} = \mathcal{NP}$ if and only if 3-SAT$\in \mathcal{P}$.

Formally, a problem X_1 is considered to be $\mathcal{NP}$-complete if $X_1 \in \mathcal{NP}$ and there exists a function f that can *reduce*$(\propto)$ in polynomial time an instance x of any problem $X_2 \in \mathcal{NP}$ to at least an instance y of X_1 such that $y = f(x)$ [13]. Subsequently, we can claim that there exists a polynomial time solution for y if and only if x has a solution in polynomial time.

If X_1, X_2, and X_3 are three $\mathcal{NP}$-complete problems, then they hold the following properties:

- Property 1. If $X_2 \propto X_1$ and $X_1 \propto X_3$, then $X_2 \propto X_3$.
- Property 2. Let problem $A^{'}$ be the special case of problem A; if $A^{'} \in \mathcal{NP}$, then $A \in \mathcal{NP}$, and if $A^{'} \in \mathcal{NP}$-complete, then $A \in \mathcal{NP}$-complete.

Karp [17] established the reducibility theory by showing several combinatorial optimization problems $\mathcal{NP}$-complete using the novel reduction technique. We highlight Karp's approach of proving a new problem $\mathcal{NP}$-complete by Lemma 2.2.

Lemma 2.2 [17]. Let X_1 and X_2 be two computational problems; if $X_1, X_2 \in \mathcal{NP}$, $X_1 \in \mathcal{NP}$-complete, and $X_1 \propto X_2$, then $X_2 \in \mathcal{NP}$-complete.

Proof. We are required to show that $X_2 \in \mathcal{NP}$-complete. Since $X_2 \in \mathcal{NP}$, we only need to show that, for every problem $X_3 \in \mathcal{NP}$, $X_3 \propto X_2$. We are given that $X_1 \in \mathcal{NP}$-complete, implies $X_3 \propto X_2$. Already we know that $X_1 \propto X_2$. Therefore, by Property 1, we have $X_3 \propto X_2$. □

The $\mathcal{NP}$-completeness theory of Karp [17] rejuvenated the non-trivial question of whether $\mathcal{P} =?\mathcal{NP}$. Despite some quality efforts from many intelligent researchers over the last five decades, no polynomial-time algorithms have been explored for $\mathcal{NP}$ class problems to date. It seems that $P = \mathcal{NP}$ is quite unlikely; therefore, we presumably consider that $\mathcal{P} \subset \mathcal{NP}$. For an exhaustive read of the rigorous mathematical definitions, properties, and in-depth analysis of the complexity classes, we refer the readers to the following outstanding articles [1, 2, 9, 13, 17].

The reducibility theory has established the foundation for locating the borderline between $\mathcal{P}$ (easy) and $\mathcal{NP}$-complete (hard) problems. The precise understanding of the borderline helps us figure out the parameters of a problem that determine its complexity class and also assists in formulating effective solution methods. For example, proving a problem $\mathcal{NP}$-complete establishes a formal argument to use enumerative solution methods such as approximation, local search, branch, and bound. Before presenting our results on the hardness of the MPSP, we first overview the related studies in the next section.

2.2. Overview of state-of-the-art related work

This section reviews and highlights the state-of-the-art literature on the hardness study of the scheduling problem for identical machine settings with optimality criteria such as makespan, the sum of completion times, the weighted sum of completion times, and the weighted mean completion time.

Karp [17] first proved that the problem of scheduling a list of jobs on a set of parallel processors to complete all jobs within a given deadline is $\mathcal{NP}$-complete. The author showed the proof by a polynomial-time reduction of an instance of the problem from an instance of the well-known *0/1-knapsack problem*. Bruno et al. [8] proved that scheduling a list of independent jobs to minimize the *weighted mean completion time (WMCT)* of the job schedule is $\mathcal{NP}$-complete. The result was shown by a polynomial-time reduction from the 0/1-knapsack problem. Garey and Johnson [12] studied the hardness of the scheduling problem with $m(\geqslant 2)$-identical processors, where each job J_i followed a partial order and required some resources for its execution. The authors proved that the problem was $\mathcal{NP}$-complete even for $m = 2$ and n jobs by a polynomial-time reduction from the well-studied *node cover* problem. The variant of the problem with $m = 5$ and $n = 8$ was shown to be $\mathcal{NP}$-complete by a polynomial-time reduction from the well-known 3-*dimensional-matching problem*. Ullman [23] studied a variant of the scheduling problem that was proposed by Garey and Johnson in [12] with $m = 2$ to minimize the makespan, where processing time $p_i = \{1, 2\}$, $\forall J_i$. The author showed the $\mathcal{NP}$-completeness of the problem by a polynomial-time reduction from the 3-SAT problem.

The readers can find a comprehensive survey of the seminal contributions and early results on the computational hardness of the multiprocessor-scheduling problem and its variants in [18].

Recently, there has been increasing interest in the study of the parameterized complexity classes of variants of the MPSP problem. The idea of such a class is to consider exponential-time algorithms for solving $\mathcal{NP}$-complete problems while restricting them to a smaller parameter. Generally, an instance of any parameterized problem X consists of two input parameters; i.e., x and k. Problem X is fixed-parameter tractable (FPT) if X is solvable in time $f(k) \cdot poly(|x|)$ for any computable function f. Problem X is $W[t]$-hard if there exists a function f that can reduce $(\propto)$ in time $f(k) \cdot poly(|x|)$ an instance $(x^{'}, k^{'})$ of any problem $X^{'} \in W[t]$-hard to at least an instance (x, k) of

X such that $k \leqslant g(k^{'})$ and $(x^{'}, k^{'}) \in X^{'} \iff (x, k) \in X$, where f and g are two computable functions. There is a hierarchy of parameterized complexity classes; i.e., $FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[P]$.

Bodlaender and Fellows [7] showed the precedence-constrained deadline-based m-processor-scheduling problem $W[2]$-hard by a polynomial-time reduction from the parameterized *dominating set* problem. Mnich and Wiese [20] investigated the problem of scheduling a list $A = \{J_1, J_2, J_3, \ldots, J_n\}$ of n jobs on a single processor with the flexibility of rejecting a set $A^{'} \subseteq A$ of r jobs at most. Each job J_i is associated with a weight w_i and processing time p_i, and each rejected one incurs a rejection cost of e_i. The objective is to minimize $\sum_{J_i \in A - A'} w_i c_i + \sum_{J_i \in A'} e_i$, where c_i is the completion time of job J_i. The authors proved that the problem is $W[1]$-hard for the parameter r of the number of rejected jobs by a polynomial-time reduction from the well-known parameterized *r-subset sum* problem. Bevern et al. [5] proved that the problem is $W[2]$-hard with respect to the width of the partial order of the jobs. The authors showed their claim by a polynomial-time reduction from the parameterized *shuffle-product* problem.

Blazewicz et al. [6] proved that the problem of scheduling a list of unit-sized coupled jobs on a single processor with precedence constraints and a makespan minimization objective is $\mathcal{NP}$-hard. The coupled job consists of two operations, where the execution of the second operation starts only after the completion of the first one. The authors showed the $\mathcal{NP}$-hardness proof by a polynomial-time reduction from the well-known *balanced-coloring of graphs with partially-ordered vertices* problem. Garg et al. [14] investigated the problem of scheduling a list of jobs on a set of m-processors to minimize the sum of the job-completion times, where each job consists of components of different types with unit sizes, and each machine is capable of processing components of a single type. The authors proved the $\mathcal{NP}$-hard problem by a polynomial-time reduction from the well-studied *vertex cover* problem. Ambuhl et al. [3] studied the problem of scheduling a list of jobs on a 1-processor with precedence constraints to minimize the weighted sum of the job-completion times where each job has a non-negative weight. The authors proved that the problem is $\mathcal{NP}$-hard by a polynomial-time reduction from the vertex cover problem.

Svensson [22] studied the problem of scheduling a list of jobs on an m-processor with precedence constraints to minimize the makespan. The author established a unique relationship between the considered problem and the problem of scheduling a list of weighted jobs on a single processor with precedence constraints to minimize the sum of the weighted job-completion times. The author proved that, if the 1-processor case is computationally hard to approximate closer to a factor of $2 - \delta$, then the m-processor problem for unit-sized jobs is also harder to approximate within a factor of $2 - \epsilon$ (where $\epsilon \to 0$ as $\delta \to 0$). Bellenguez et al. [4] studied the preemptive scheduling of a list of jobs on a 3-processor to minimize the sum of job completion times, where each job has a release time. The authors showed the $\mathcal{NP}$-hardness proof by a polynomial-time reduction from the 3-Partition problem. Zhang et al. [24] investigated the problem of scheduling a list of jobs on a 1-processor to minimize the

makespan with time restrictions and the following K-constraint. Given $K = 2$ for any real t, no unit-sized time interval $[t, t + 1)$ is permitted to intersect greater than K jobs. The authors proved the $\mathcal{NP}$-hard problem by a polynomial-time reduction from the well-known *equal cardinality partition* problem. Davies et al. [11] studied the problem of scheduling a list of jobs on an m-processor with precedence constraints to minimize the makespan. For each pair of jobs that satisfy the precedence relationship, there exists a communication delay; i.e., the duration of time that the scheduler waits between the two dependent jobs if the jobs are assigned to different machines. The author showed the $\mathcal{NP}$-hardness proof of the problem by a reduction from the well-known *unique machines precedence-constrained scheduling* problem.

It is evident from the literature that the hardness proofs of all $\mathcal{NP}$-complete scheduling problems offer a polynomial-time reduction from well-known $\mathcal{NP}$-complete problems by following *Property 1* of the complexity classes. However, the existing literature lacks a formal method for explicitly showing that *MPSP* is $\mathcal{NP}$. The $\mathcal{NP}$-hardness of *MPSP* was shown in [17] and [23] by a reduction from the *partition problem*; however, the shown result was intuitive and required refinement for the ease of understanding from a beginner's perspective. On this note, we present a systematic study to explore the hardness of the MPSP problem. We also prove the complexity class of the MUMPSP problem. We highlight a summary of well-known related hardness studies in Table 1.

Table 1

Well-known Studies on $\mathcal{NP}$-completeness of Multiprocessor Scheduling Problem

Year, Author(s)	Variants of MPSP	Reduction
1972, Karp [17]	2-processor scheduling with deadline	0/1-Knapsack
1974, Bruno et al. [8]	m-processor scheduling with weighted mean completion time	0/1-Knapsack
1974, Garey, Johnson [12]	2-processor resource-constrained scheduling	Node Cover
1975, Ullman [23]	2-processor resource-constrained scheduling	3-SAT
1977, Lenstra et al. [18]	2-processor scheduling with makespan minimization	2-Partition
1995, Bodlaender, Fellows [7]	Precedence-constrained m-processor scheduling problem is $W[2]$-hard for parameterized complexity class	Dominating Set
2001, Blazewicz et al. [6]	Scheduling of unit-sized coupled jobs on 1-processor with precedence-constrained and makespan minimization	Balanced-coloring of graphs with partially-ordered vertices
2007, Garg et al. [14]	m-processor scheduling to minimize sum of completion times	Vertex Cover

Table 1 cont.

2011, Ambuhl et al. [3]	1-processor scheduling with precedence constraints to minimize the weighted sum of completion times	Vertex Cover
2011, Svensson [22]	Precedence constrained m-processor scheduling with makespan minimization	Precedence-constrained 1-processor scheduling with sum of weighted completion times
2015, Mnich, Wiese [20]	Scheduling on 1-processor with job rejection to minimize the weighted sum of completion times is $W[1]$-hard	r-Subset Sum
2015, Bellenguez et al. [4]	3-processor preemptive scheduling with a release date to minimize the sum of completion time	3-Partition
2016, Bevern et al. [5]	Precedence-constrained scheduling is $W[2]$-hard w.r.t. width of partial order	Shuffle-product
2018, Zhang et al. [24]	1-processor scheduling with time restriction to minimize makespan	Equal Cardinality Partition
2021, Davies et al. [11]	m-processor scheduling with non-uniform delay	Unique Machines Precedence-constrained Scheduling

3. Combinatorial structure of multiprocessor scheduling problem

In this section, we first formally define the MPSP problem and then present its combinatorial structure by proposing the SSST data structure, followed by a discussion of our analytical results.

Multiprocessor Scheduling Problem (MPSP)

- Inputs:
  - Given, $M = \{M_1, M_2, M_3, \ldots, M_m\}$ is the set of m identical processors, and $J = \{J_1, J_2, J_3, \ldots, J_n\}$ is the list of n jobs, where $n \ggg m$.
  - Processing time p_i for job J_i, where $p_i \geqslant 1$ and $1 \leqslant i \leqslant n$.
- Output: A schedule $S = \{J^1, J^2, \ldots, J^m\}$ is the collection of m disjoint partitions of set J such that $\bigcup_{j=1}^{m} J^j = J$, and for each pair of partitions $(J^a, J^b) \in S$, we have $J^a \cap J^b = \phi$, where $a \neq b$ and each $J^j (\subseteq J)$ represents a set of jobs assigned to respective machine M_j. Each M_j has a load l_j, where $l_j = \sum_{J_i \in M_j} p_i$. The output parameter is *makespan*, which is represented as C_{max}, where $C_{max} = \max_{1 \leqslant j \leqslant m} l_j$.

- Objective: To minimize C_{max}.
- Assumptions:
  - Independent jobs: Set J is free from the partial order ($\prec$) relationship. Thus, jobs can execute in parallel.
  - Non-preemption: If a job J_i with processing time p_i starts its execution on any M_j at time t, then it continues on M_j until time $t + p_i$.

We define two new characterizations of *Schedule S* such as *Partial Schedule* (S_p) and *Essential Schedule* (S_e) as follows.

Definition 3.1. Partial Schedule (S_p) is the collection of m disjoint partitions of a set $J' \subset J$ such that $\bigcup_{j=1}^{m} J'^j = J'$, where $J'^j \subseteq J'$, and for each pair of partitions $(J^a, J^b) \in S_p$, we have $J^a \cap J^b = \phi$, where $1 \leqslant a, b \leqslant m$, and $a \neq b$.

Definition 3.2. Essential Schedule (S_e) is a Schedule S, where $|J^j| \geqslant 1, \ \forall j$.

3.1. Combinatorial structure of MPSP as scheduling solution space tree

Before presenting the combinatorial structure of MPSP, we define the following basic terms.

Definition 3.3. The level of a tree defines the positioning of the parent node and its children in the sense that, if the parent is at level b ($\geqslant 0$), then its child nodes are at level $b + 1$ (assuming that the root node is at level 0).

Definition 3.4. The leaf node of a tree is a node with no children, whereas a non-leaf node has at least one child node.

Definition 3.5. The depth or height (h) of a tree can be defined as the maximum level of the tree where there exists no non-leaf nodes.

The combinatorial structure represents all of the possible solutions of a computational problem [16]. In the case of MPSP, we have m possibilities to schedule the job if we consider one job and m processors. Thus, for scheduling n jobs on m processors ($n > m$), we can have m^n possible schedules at most, which is an exponential solution space.
For a basic understanding, we explore the combinatorial structure for the restricted case of the MPSP problem, where $m = 2$. We present the assignments of jobs to processors as configurations through the construction of a *scheduling solution space tree (SSST)* 2 (as shown in Figure 2).

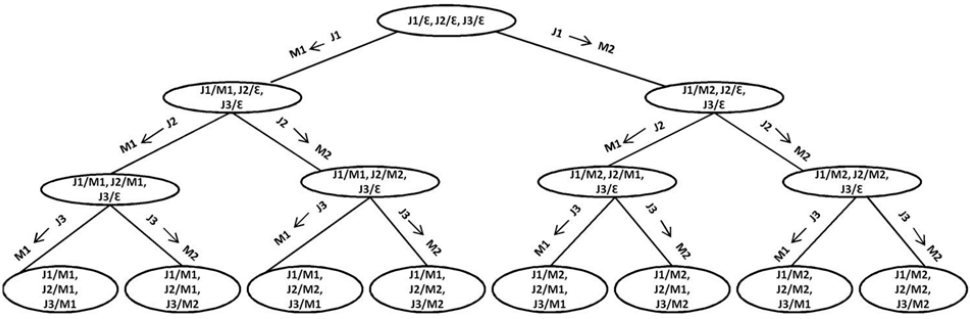

Figure 2. Scheduling Solution Space Tree for Three Jobs and Two Processors

In an SSST, each node N represents an n-tuple of the J_i/M_j type that shows the assignment of job J_i to processor M_j. Each tuple in the root node represents *initial configuration* J_i/ϵ, where none of the jobs have been assigned to any of the machines. Tuple J_i/ϵ changes to tuple J_i/M_j through an arc with an action that is represented by $J_i \rightarrow M_j$ when job J_i is assigned to processor M_j. The leaf nodes are represented by tuples of the J_i/M_j type that show the assignments of all jobs to some machines. Let us consider the case where $m = 2$ with $M = \{M_1, M_2\}$ and $n = 3$ with $J = \{J_1, J_2, J_3\}$. Let the initial configuration be represented by root node N_0 as a 3-tuple, where $N_0(J_1/\epsilon, J_2/\epsilon, J_3/\epsilon)$. For the scheduling of job J_1 on either the M_1 or M_2 processor, two child nodes (N_1 and N_2) are created through arcs $J_1 \rightarrow M_1$ and $J_1 \rightarrow M_2$ in the SSST to capture two possible new configurations, where $N_1 : (J_1/M_1, J_2/\epsilon, J_3/\epsilon)$ and $N_2 : (J_1/M_2, J_2/\epsilon, J_3/\epsilon)$, respectively. Thus, two child nodes are inserted into the SSST to capture two possible assignments. Recursively, new nodes are created from each internal node by exploring two alternative schedules for each unscheduled job. At level i of the SSST where $1 \leqslant i \leqslant n$, all possible assignments of the i^{th} job are represented in 2^i nodes (of course, we do not consider the root node that resides at level 0). Each leaf node represents a schedule S with configuration $(J_1/M_j, J_2/M_j, J_3/M_j)$, where $j \in \{1, 2\}$.

Weighted Scheduling Solution Space Tree (WSSST)

This is an extension of the SSST data structure. While the SSST showcases the assignments of jobs to processors, the WSSST data structure represents the intermediate and final loads of the processors in all possible schedules. We present the job assignments and updated loads of the processors as configurations through the construction of the WSSST by considering an instance of MPSP (as shown in Figure 3). We consider 2 processors and a $J = \{J_1/1, J_2/1, J_3/3\}$ set of 3 jobs, where each job J_i is characterized by its processing time p_i. In a WSSST, each node N represents a m-tuple of the M_j/l_j type that shows the value of the load l_j of the corresponding machine M_j, where $l_j = \sum_{J_i \in M_j} p_i$, the sum of the processing times of the jobs assigned to M_j. Each tuple in the root node represents the *initial configuration* $M_j/0$,

which indicates that none of the jobs have been assigned to any of the machines. Tuple $M_j/0$ changes to tuple M_j/l_j through an arc with an action represented by $J_i \rightarrow M_j$ when job J_i is assigned to processor M_j. The leaf nodes are represented by tuples of the M_j/l_j type, which reflect the final load of machine M_j, and the makespan C_{max} of the job schedule can be computed as $C_{max} = \max\{l_j \mid 1 \leqslant j \leqslant m\}$.

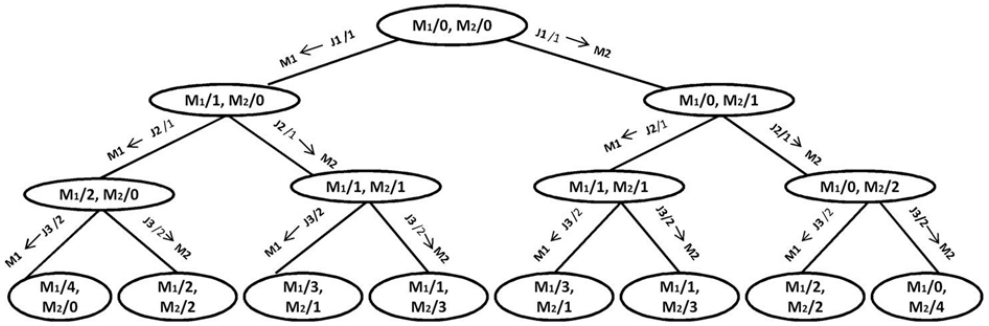

Figure 3. Weighted Scheduling Solution Space Tree for Three Jobs and Two Processors

3.2. Our Analytical Results Based on Characterization of SSST

Here, we explore and present some of the important properties of an *SSST* and show a non-trivial characterization of the combinatorial solution space of MPSP. We begin with the following definitions.

Definition 3.6. A **path** [16] in a tree can be defined as the collection of nodes and edges along the way from the root node to one of the leaf nodes. The first node in the collection is the root node, and the other nodes and edges are chosen as follows: at each level b where $1 \leqslant b \leqslant h$, exactly one node is chosen whose parent has already been chosen at level $b-1$ along with the edge that joins the parent at level $b-1$ to its child at level b.

Definition 3.7. A **perfect m-ary tree** is an m-ary tree where each node has exactly m child nodes, and all leaf nodes are at the same level.

Lemma 3.8. *$N(h) = \frac{m^{h+1}-1}{m-1}$, where $N(h)$ is the total number of nodes of the perfect m-ary tree of height h.*

Proof. Let N^b be the number of nodes at height b where $0 \leqslant b \leqslant h$. Clearly, $N^0 = 1 = m^0$, as only a single node resides at the root. By *Definition 3.3*, we have $N^1{=}m$, $N^2{=}m^2$, and so on. In general, we can write $N^h{=}m^h$; therefore, we have $N(h){=}m^0 + m^1 + m^2 + ... + m^{h-1} + m^h{=}\frac{m^{h+1}-1}{m-1}$.
We prove *Lemma 3.8* by the method of induction on height h of the SSST as follows.

Induction Basis: If $h = 0$, then we have $N(h) = \frac{m^{0+1}-1}{m-1} = 1 = m^0$; thus, $N(0)$ holds true for height $h = 0$.
Induction Hypothesis: Let us assume that $N(k) = \frac{m^{k+1}-1}{m-1}$ is true for height $h = k$.
Inductive Step: We have to show that $N(k+1)$ is true for $h = k+1$; i.e., $N(k+1) = \frac{m^{k+2}-1}{m-1}$. We can construct a *perfect m-ary tree* of height $k+1$ by adding m^{k+1} nodes to $N(k)$ at level $k+1$; hence, $N(k+1) = N(k) + m^{k+1} = \frac{m^{k+2}-1}{m-1}$. Therefore, $N(k+1)$ holds true for $h = k+1$, and $N(h)$ is true for all $h \geqslant 0$. □

Theorem 3.9. *If an SSST for MPSP with m processors and n jobs is a perfect m-ary tree, then*

a) $h{=}n$ and $|S|{=}m^n$;
b) $|S_p|{=}\frac{m^n-m}{m-1}$;
c) $|S_e| = m^n - m$.

Proof. Let us consider the $J = \{J_1, J_2, J_3, \ldots, J_n\}$ set of n jobs to be scheduled on m processors.
(a) To represent all possible options of the assignment of each job J_i to one of the m processors, the SSST creates m child nodes for each node at level $i-1$, where $i \geqslant 1$. Let N^i be the number of nodes at level i. Clearly, $N^0 = 1$, $N^1 = 1 \cdot m = m$, $N^2 = m \cdot m = m^2$, and so on. Without a loss of generality, we can write $N^i = m \cdot N^{i-1}$, where each node represents a distinct schedule for jobs $J_1, J_2, \ldots, J_i$. Thus, at level $i = n$, the *SSST* represents distinct schedules for the J set of jobs by m^n leaf nodes. This implies $|S| = m^n$ and $N(n) = \frac{m^{n+1}-1}{m-1}$. Therefore, by *Lemma 3.8*, we can conclude that the *SSST* for *MPSP* with m-processors and n jobs is a *perfect m-ary tree* of height $h = n$.

(b) It is clear from the construction of the SSST that each level i represents m^i *partial schedules* for a $J' = \{J_1, J_2, \ldots, J_i\}$ set of jobs where $1 \leqslant i \leqslant n-1$ and $J' \subset J$. Therefore, we have $|S_p| = m^1 + m^2 + \ldots + m^{n-1} = \frac{m^n-m}{m-1}$.

(c) Let us consider a *path* in the *SSST* from the root to one of the leaf nodes such that, at each level i where $1 \leqslant i \leqslant n-1$, job J_i is assigned to the same machine M_j. Clearly, for an *m-ary SSST*, there are m leaf nodes that represent the assignments of all jobs to any one of the m machines. This implies that the remaining leaf nodes represent those schedules where at least one job is assigned to all M_j where $1 \leqslant j \leqslant m$. Therefore, we have $|S_e| = m^n - m$.
This completes the proof of Theorem 3.9. □

4. Our results on hardness of MPSP

In this section, we prove that MPSP is $\mathcal{NP}$-complete by simulating a nondeterministic Turing machine ($\mathcal{NTM}$) through our proposed SSST and reducing an

instance of MPSP from an instance of the well-known partition problem. Our objective is to present exhaustive and refined proof sketches of $\mathcal{NP}$ and $\mathcal{NP}$-hardness for a better understanding of the hardness of the MPSP.

4.1. MPSP $\in \mathcal{NP}$

Here, we first prove $MPSP \in \mathcal{NP}$ by mapping the construction of an SSST as a $\mathcal{NTM}$. Alternatively, we develop a polynomial-time verifier by using SSST and WSSST data structures.

Theorem 4.1. $MPSP \in \mathcal{NP}$

Proof. We consider an instance of MPSP where $m = 2$. Let us assume there exists an encoding that can represent MPSP with at least n symbols. Without a loss of generality, let set $J = \{J_1, J_2, J_3, \ldots, J_n\}$ denote the input symbols and set $M = \{M_1, M_2\}$ represent the output alphabet of the $\mathcal{NTM}$. Let set $N = \{\{N_0\} \cup \{N_1, N_2, \ldots, N_{2^{n+1}-2}\}\}$ represent the nodes of the SSST where N_0 is the root node. We now can transform the SSST as a state transition diagram that is generated by the $\mathcal{NTM}$, where the root node represents the initial state of the 1-tape $\mathcal{NTM}$, and each node represents a state with a configuration that results due to the transition function, which is defined as $\delta : NXJ \rightarrow NXM_jXR$ (where $j \in \{1, 2\}$ and R represents the right-hand-side move off the tape header).

It is clear by *Theorem 3.9* that any scheduling certificate that represents the assignments of jobs can be verified by traversing the SSST from its root to one of the leaf nodes in polynomial time bounded by the height of the tree.

After representing all of the schedules at the leaf nodes of the SSST, we now consider a WSSST where each node (N_x) is assigned with a *weight* to represent the current loads of the machines. Therefore, each leaf node now represents a scheduling certificate with a makespan value. It is assumed that the value of the optimum makespan $C_{OPT} = \frac{\sum_{i=1}^{n} p_i}{m}$. We consider C_{OPT} to be a reference to compare with the weight of any leaf node. Let W be a weight function from N to non-negative integers that represents the value of the makespan at any N_x. We consider $W(N_x) = \max_{1 \leqslant j \leqslant 2} l_j$ where $l_j = \sum_{J_i \in M_j} p_i$ and $1 \leqslant x \leqslant 2^{n+1} - 2$. Let us assume $W(N_0) = 0$. The optimality of any scheduling certificate can be verified in polynomial time by comparing the weight of the leaf node with the value of C_{OPT}.

Therefore, it has been proven that $MPSP \in \mathcal{NP}$. □

4.2. SSST as polynomial-time verifier

The $\mathcal{NP}$ class covers all decision problems for which the given solutions are verifiable in polynomial time [13]. According to the interactive proof procedures of Goldwasser et al. [15], the verification process considers a *prover* and a *verifier*. The *prover* provides a yes/no answer to the decision version of a computational problem with an explanation in the form of a solution certificate. The verifier validates the answer

in polynomial time and concludes whether the provided answer is correct or not. For a proper validation, the verifier must know all feasible solutions as well as the optimal one. Let us consider an instance of the MPSP problem with n jobs, corresponding processing times, and the value of the optimum makespan C_{OPT}. We use the SSST and WSSST together as a verifier for the MPSP problem.

Decision version of MPSP. Is there a schedule S for a given list of n jobs on m identical processors such that $C_{max} \leqslant C_{OPT}$?
Suppose the true answer is "yes".
Prover. This provides a schedule S with a value of C_{max}.
Verifier. Given schedule S and value C_{max}, the verifier must validate the following in the polynomial time.

1. *Given S is valid schedule of list of n jobs on m processors.*
   A valid schedule can be verified by traversing the SSST from the root to a leaf node, and the time that is required to complete the traversal is bounded in polynomial time by the height of the SSST.
2. *The $C_{max} \leqslant C_{OPT}$.*
   For a given instance of $MPSP$, we can know the actual value of the optimum makespan by comparing the values of the leaf nodes of the WSSST. After knowing the exact value of C_{OPT}, the verification of $C_{max} \leqslant C_{OPT}$ requires a constant time.

The theoretical lower bound of the optimum makespan, i.e., $C_{OPT} \geqslant \frac{1}{m} \cdot \sum_{i=1}^{n} p_i$ does not reflect the actual value of C_{OPT} in all instances of the problem. The theoretical value of C_{OPT} is impractical for real-life instances; therefore, the WSSST would be handy in deriving the actual value of C_{OPT} in practical scenarios.

The SSST is an implicit representation, meaning that the nodes and branches are considered to be implicit objects in a computer's memory. One can algorithmically generate the components of an SSST for a given instance of the problem. A typical SSST is a larger structure to represent in the memory. The nodes of the SSST can be generated while exploring a path from the root to a leaf node to verify a scheduling-solution certificate and are subsequently discarded. Therefore, the SSST data structure could be a global polynomial-time verifier for any instance with a solution certificate of the MPSP.

4.3. MPSP $\in \mathcal{NP}$-hard

Here, we first show the $\mathcal{NP}$-hardness of the MPSP problem by a polynomial-time reduction from the well-known partition problem. Then, we develop a new reduction framework to design the first non-deterministic polynomial-time algorithm for the MPSP problem.

Theorem 4.3. *$MPSP \in \mathcal{NP}$-hard*

Proof. We start by defining the *partition problem (PP)* and show the polynomial time reduction of an instance of *MPSP* from an instance of *PP*.
Partition Problem: Given, a set $A = \{a_i \| 1 \leqslant i \leqslant n\}$ of n elements, where each element a_i has a non-negative weight w_i, and the total weight of set A is $W(A) = \sum_{a_i \in A} w_i$.
Decision Version of PP: Can we have partitions of set A into r disjoint sets such that $W(A_1) = W(A_2) = \ldots = W(A_r) = \frac{W(A)}{r}$?
Instance of PP: We have an instance of *PP* by considering $r = 2$.
We now define the *decision version* of 2-PP as follows:
Can we have a subset $A^{'} \subset A$ such that $\sum_{a_i \in A^{'}} w_i = \sum_{a_i \in A - A^{'}} w_i = \frac{W(A)}{2}$?
Instance of MPSP: We consider an instance of *MPSP* with $m = 2$.
Polynomial Transformation of 2-PP to 2-processor scheduling problem (2-PSP): Let us consider $D = W(A)$ and $C_{OPT} = \frac{\sum_{i=1}^{n} p_i}{2}$. We now show the equivalence between the input parameters of 2-*PP* and 2-*PSP* as follows: $J_i = a_i$ and $p_i = w_i$ for $1 \leqslant i \leqslant n$, $m = 2$ jobs assigned to machine M_1 are equivalent to the elements that belong to set $A^{'}$, and the jobs assigned to machine M_2 are equivalent to the elements that belong to set $A - A^{'}$; we consider $C_{max} = \max\{W(A^{'}), W(A - A^{'})\}$.
Clearly, the transformation is bounded by a polynomial in the length of the inputs that are represented as the set elements or jobs.
We now can claim that 2-*PSP* has a schedule S such that $C_{max} \leqslant C_{OPT} = \frac{D}{2}$ if and only if there exists a subset $A^{'} \subset A$ such that $W(A^{'}) = W(A - A^{'}) = \frac{D}{2}$ for positive number D. □

4.4. Framework for designing non-deterministic polynomial-time algorithm

The $\mathcal{NP}$ complexity class defines a set of problems for which we do not have deterministic polynomial-time algorithms, but the class accepts the existence of non-deterministic polynomial-time algorithms for these problems [13]. We develop a general framework for designing a non-deterministic polynomial-time algorithm for the MPSP using the principle of *reduction* that was discussed in Section 2.1. Let us consider that we have a computational problem B and a well-known $\mathcal{NP}$-hard problem A. If an instance of problem A is reduced ($\propto$) to an instance of problem B, we can solve problem A in polynomial time if and only if there exists a polynomial-time solution for problem B and vice-versa. Moreover, we can develop a polynomial-time algorithm Alg_A for A by using the polynomial-time algorithm that was designed for problem B as a subroutine for Alg_A [13]. By considering this reduction principle, we present sample subroutines for problems A and B in Figures 4(a) and 4(b). The function Success() in the subroutine for problem B indicates the existence of the optimal output, and the Failure() function represents the non-existence of the desired outcome. If the subroutine for B runs in polynomial time, then the subroutine for A will also run in polynomial time. Next, we develop a non-deterministic polynomial-time algorithm for the MPSP using the general reduction framework.

a)

```
FUNCTION (Problem A)
  {
    Statements;
    CALL Problem B;
    Statements;
  }
```

b)

```
FUNCTION (Problem B)
 {
   Statements;
   If (Condition)
    {
      Success ();
     RETURN Solution;
    }
   ELSE
    {
      Failure ();
       RETURN 0;
    }
 }
```

Figure 4. Subroutine for Problem A (a) and Problem B (b)

4.5. Non-deterministic polynomial-time algorithm: magic scheduling

A non-deterministic algorithm makes random choices for each input in different runs, which results in different outputs [10]. In a non-deterministic algorithm, we may have some statements or functions whose operations are not defined clearly in the context of the problem. However, these statements can guide us when exploring and verifying the optimal solution for the problem. For example, if a solution certificate to an instance of the $\mathcal{NP}$-complete problem is given, the algorithm can verify whether the given solution is an optimal one or not. Hence, a non-deterministic algorithm works more like a verifier than a solver. We design the non-deterministic algorithm named *magic scheduling (MS)* by using our proposed reduction framework. We present the pseudocode of the MS algorithm in **Algorithm** 1.

The MS algorithm runs in polynomial time at the current state, and its worst-case time complexity depends on the SELECT_PARTITION () function. One can observe that, in the MS algorithm, the SELECT_PARTITION function is a non-deterministic function that partitions a list of n jobs into two subsets (A and $A^{'}$). The jobs that belong to subset A are assigned to machine M_1, and the jobs that belong to set $A^{'}$ are assigned to machine M_2. On the contrary, we do not know how the SELECT_PARTITION function selects the jobs for each subset; however, we can obtain an optimal solution for MPSP if we can determine the exact working principle. By using the same principle, we can also solve the partition problem.

Nevertheless, if a solution to an instance of MPSP is given, then the SELECT_PARTITION function can use the SSST to select an appropriate partition from the leaf nodes to verify whether the solution certificate is the optimal one or not. Therefore, the SSST data structure with our proposed non-deterministic polynomial-time MS algorithm constitutes a computational framework to show MPSP to be $\mathcal{NP}$-complete.

Algorithm 1 MS

Initially $l_1 = l_2 = 0$
PARTITION(X, w_i) // set X with n elements, each having corresponding weight w_i
  BEGIN
    $P(A, A^{'})$ ← SCHEDULE $(2, n, p_i, C_{OPT})$ // $C_{OPT} \geqslant \frac{1}{2} \cdot \sum_{i=1}^{n} p_i$
  END
SCHEDULE(m, n, p_i, C_{OPT})
  BEGIN
    $P(a, a^{'})$ ← SELECT_PARTITION() // non-deterministic function
    $a = \{J_i \mid J_i \in M_1\}$ // assigns jobs to machine M_1
    $a^{'} = \{J_i \mid J_i \in M_2\}$ // assigns jobs to machine M_2
    $l_1 = \sum_{J_i \in M_1} p_i$, and $l_2 = \sum_{J_i \in M_2} p_i$
    $C_{max} \leftarrow \max\{l_1, l_2\}$
    IF $(C_{max} = C_{OPT})$
      Success () // optimum schedule exists
      Return $P(a, a^{'})$
    ELSE
      Failure () // optimum schedule does not exist
      Return 0
  END

5. Hardness of multiuser multiprocessor scheduling problem

In MPSP, we do not explicitly consider the sources of job requests and assume that all jobs arrive from a single source only. MUMPSP is a practically significant variant of MPSP where *user* is considered to be an additional input parameter to represent the source of job requests. Here, each *user* submits a set of jobs to the scheduler and aims to obtain the optimal makespan. In the MUMPSP problem with $k(\geqslant 1)$ users, the set J of jobs that are considered in MPSP can be partitioned into k disjoint subsets, where each subset represents the jobs of a single user, and the scheduler must meet the objective of each user. This scheduling problem often arises in high-performance computing systems such as supercomputers, cloud servers, robotic controllers, clusters, and grids [21].

5.1. Multiuser multiprocessor scheduling problem

We formally define MUMPSP by presenting the inputs, output, objective, and assumptions of the problem as follows.

- Inputs:
  - $M = \{M_1, M_2, \ldots, M_m\}$ is a set of m identical processors;
  - let U_r be *user*, where $1 \leqslant r \leqslant k$ and L_r denote distinct set of jobs of respective U_r, where $L_r = \{J_i^r | 1 \leqslant i \leqslant n_r\}$ such that $\sum_{r=1}^{k} n_r = n$;
  - let p_i^r be processing time of job J_i^r, where $p_i^r \geqslant 1$.

- Output: generation of a schedule(S) that represents *makespan* $C^r_{max} = \max\{C^r_i | 1 \leqslant i \leqslant n_r\}$ for each *user* U_r, where C^r_i is completion time of job J^r_i such that $(t(S) + p^r_i) \leqslant C^r_i \leqslant C^r_{max}$ (where $t(S)$ is starting time of schedule S).
- Objective: minimization of $C^r_{max}, \quad \forall U_r$.
- Assumptions:
  - jobs are submitted from k users;
  - jobs of each user are independent of each other as well as with jobs of other users;
  - no job can be split up and preempted in middle of its execution.

5.2. Our results on hardness of MUMPSP

We have proven the MUMPSP $\mathcal{NP}$-complete problem by a polynomial-time reduction from the MPSP problem.

Theorem 5.2. *MUMPSP* $\in \mathcal{NP}$*-complete*

Proof. We consider the 2-*Processor Scheduling Problem (2-PSP)* as an instance of MPSP. We first show 2-PSP $\propto$ MUMPSP.

Problem: 2-PSP.
Instance of 2-*PSP:* Given, $m = 2$ and a list $J = \{J_1/p_1, J_2/p_2, J_3/p_3, \ldots, J_n/p_n\}$ of n jobs, where $p_i \geqslant 1$.
Decision version: Is there schedule S of list J of n jobs on 2 processors such that $l_1 = l_2 = \frac{1}{2} \cdot \sum_{i=1}^{n} p_i = D$?

We now have to construct an $I' \in$ instance MUMPSP from an $I \in$ 2-PSP instance.

Suppose the number of users is $k = 1$ for MUMPSP, and we denote the instance as 1-UMPSP. Then, I' can be constructed by letting $n_1 = n$, $p^1_i = p_i$ for $1 \leqslant i \leqslant n$, and $C^1_{OPT} = \frac{1}{2} \cdot \sum_{i=1}^{n} p^1_i = D$.

Clearly, the construction of I' requires a total of $n + 2$ assignments and n additions to compute C^1_{OPT}. Therefore, the mapping of instance I into I' requires $O(n)$ time at most.

To ensure completeness, we must show that the answer to the decision version of instance $I \in$ 2-PSP is 'yes' if and only if the answer is 'yes' for instance $I' \in$ 1-UMPSP. Let S be the optimal schedule for I such that $l_1 = l_2 = D$. Let $A_1 = \{J_i \mid \textit{jobs assigned to } M_1\}$ and $A_2 = \{J_i \mid \textit{jobs assigned to } M_2\}$. We now can construct a solution for I' by scheduling the corresponding job $J^1_i = J_i \in A_1$ on M_1 and job $J^1_i = J_i \in A_2$ on M_2. Let S' be the resulting schedule for I'. By the definition of optimal schedule S, we have $\sum_{J^1_i \in M_1} p^1_i = \sum_{J^1_i \in M_2} p^1_i = \frac{1}{2} \cdot \sum_{i=1}^{n} p^1_i = D$.

Clearly, schedule $S^{'}$ is constructed in $\mathcal{O}(n)$ time given S. Similarly, we can obtain an optimal schedule S in polynomial time by mapping the same assignment procedure given the optimal schedule $S^{'}$ for 1-UMPSP. This completes the reduction part to show that MUMPSP is $\mathcal{NP}$-hard. Now, we establish an analogy to show the $\mathcal{NP}$-completeness of the problem.

As we know, MPSP is a special case of MUMPSP with $k = 1$ number of users. Thus, by *Property 2* of the complexity classes that were discussed in Section 2.1, we conclude that MUMPSP is $\mathcal{NP}$-complete for any $k \geqslant 2$. □

6. Concluding remarks

We formally explored and presented the combinatorial structure of MPSP by developing SSST and WSSST data structures with well-defined properties. The computational hardness of the problem was shown by our refined analytical proof techniques. We developed a non-trivial method by considering a combined SSST and WSSST as a polynomial-time verifier to exclusively show that $MPSP \in \mathcal{NP}$. We designed the MS optimal scheduling algorithm by our proposed non-deterministic algorithmic framework. We formally defined MUMPSP as a standard variant of MPSP and proved that $MUMPSP \in \mathcal{NP}$-complete. To further extend our studies on the exploration of the combinatorial structure and hardness of MPSP and MUMPSP, we present the following non-trivial research challenges.

Research challenges

- Can the *SSST* and WSSST data structures be used to explore the combinatorial structure of a similar class of $\mathcal{NP}$-complete problems?
- Can we have an alternate and simple $\mathcal{NP}$-hard reduction of the MUMPSP problem from any of the well-known $\mathcal{NP}$-complete problems other than the MPSP problem?
- What is the mathematical formulation for achieving the practically significant optimum value of makespan C_{OPT} for all instances of MPSP in general?
- Can SSST be extended as a *disjoint-set data structure* to explore the combinatorial structure of MUMPSP?
- Can we design the PTAS algorithms for MUMPSP for $k = m$?

Acknowledgements

We remain grateful to the Veer Surendra Sai University of Technology for providing the adequate facilities for carrying out our current research work and considering this contribution as a part of the Ph.D. work of Debasis Dwibedy. We express our profound gratitude to the competent anonymous reviewers for their constructive suggestions for improving the scientific rigor of the article.

References

[1] Aaronson S.: $P \stackrel{?}{=} \mathcal{NP}$. In: J. Nash, Jr., M. Rassias (eds.), *Open Problems in Mathematics*, pp. 1–122, Springer, Cham, 2016.

[2] Aho A.V., Hopcroft J.E., Ullman J.D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] Ambühl C., Mastrolilli M., Mutsanas M., Svensson O.: On the Approximability of Single-Machine Scheduling with Precedence Constraints, *Mathematics of Operations Research*, pp. 653–669, 2011.

[4] Bellenguez-Morineau O., Chrobak M., Dürr C., Prot D.: A note on NP-Hardness of preemptive mean flow-time scheduling for parallel machines, *Journal of Scheduling*, vol. 18, pp. 299–304, 2015.

[5] Bevern van R., Bredereck R., Bulteau L., Komusiewicz C., Talmon N., Woeginger G.J.: Precedence-Constrained Scheduling Problems Parameterized by Partial Order Width. In: *Discrete Optimization and Operations Research. 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19–23, 2016, Proceedings*, pp. 105–120, 2016.

[6] Blazewicz J., Ecker K., Kis T., Tanas M.: A note on the complexity of scheduling coupled tasks on a single processor, *Journal of Brazilian Computer Society*, pp. 23–26, 2001.

[7] Bodlaender H.L., Fellows M.R.: $W[2]$-hardness of precedence constrained K-processor scheduling, *Operations Research Letters*, vol. 18, pp. 93–97, 1995.

[8] Bruno J., Coffman E.G., Sethi R.: Scheduling independent tasks to reduce mean finishing time, *Communications of the ACM*, pp. 382–387, 1974.

[9] Cook S.A.: The complexity of theorem-proving procedures. In: *STOC'71: Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, 1971.

[10] Cormen T.H., Leiserson C.E., Rivest R.L., Stein C.: *Introduction to Algorithms*, MIT Press, 3rd ed., 2009.

[11] Davies S., Kulkarni J., Rothvoss T., Sandeep S., Tarnawski J., Zhang Y.: On the Hardness of Scheduling With Non-Uniform Communication Delays, *CoRR abs/210901064*, 2021.

[12] Garey M.R., Johnson D.S.: Complexity results for multiprocessors scheduling under resource constraints. In: *Proceedings of the 8th Annual Princeton Conference on Information Sciences and Systems*, pp. 384–393, 1974.

[13] Garey M.R., Johnson D.S.: *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman & Co., New York, USA, 2nd ed., 1990.

[14] Garg N., Kumar A., Pandit V.: Order scheduling models: Hardness and algorithms. In: *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science. 27th International Conference, New Delhi, India, December 12–14, 2007, Proceedings*, pp. 96–107, 2007.

[15] Goldwasser S., Micali S., Rackoff C.: The knowledge complexity of interactive proof systems, *SIAM Journal of Computing*, pp. 186–208, 1989.

[16] Horowitz E., Sahni S., Anderson-Freed S.: *Fundamentals of Data Structures in C*, Universities Press, 2nd ed., 2008.

[17] Karp R.M.: Reducibility among combinatorial problems. In: *Complexity of Computer Computations. Proceedings of a symposium on the Complexity of Computer Computations, held March 2022, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, Department of Computer Science, University of California at Berkeley, 1972.

[18] Lenstra J.K., Rinnooy Kan A.H.G., Brucker P.: Complexity of machine scheduling problems, *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[19] McNaughton R.: Scheduling with deadlines and loss functions, *Management Science*, vol. 6, pp. 1–12, 1959.

[20] Mnich M., Wiese A.: Scheduling and fixed-parameter tractability, *Mathematical Programming*, vol. 154, pp. 533–562, 2015.

[21] Saule E., Trystram D.: Multi-users scheduling in parallel systems. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–9, Washington DC, USA, 2009.

[22] Svensson O.: Hardness of precedence constrained scheduling on identical machines, *SIAM Journal of Computing*, vol. 40, pp. 1258–1274, 2011.

[23] Ullman J.D.: $\mathcal{NP}$-Complete Scheduling Problems, *Journal of Computer and System Sciences*, vol. 10, pp. 384–393, 1975.

[24] Zhang A., Chen Y., Chen L., Chen G.: On the NP-hardness of scheduling with time restrictions, *Discrete Optimization*, pp. 54–62, 2018.

Affiliations

Debasis Dwibedy
Veer Surendra Sai University of Technology, Burla, Odisha, India, debasis.dwibedy@gmail.com

Rakesh Mohanty
Veer Surendra Sai University of Technology, Burla, Odisha, India, rakesh.iitmphd@gmail.com

Received: 22.01.2022
Revised: 19.09.2022
Accepted: 28.10.2022