九州大学学術情報リポジトリ
Kyushu University Institutional Repository

# Succinct Representation of Linear Extensions via MDDs and Its Application to Scheduling Under Precedence Constraints

Miyake, Fumito
Department of Informatics, Kyushu University

Takimoto, Eiji
Department of Informatics, Kyushu University

Hatano, Kohei
Department of Informatics, Kyushu University

https://hdl.handle.net/2324/4495596

出版情報：Combinatorial algorithms : 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5-7, 2021 : proceedings, pp.365-377, 2019-07-10. Springer-Verlag
バージョン：
権利関係：

KYUSHU UNIVERSITY

# Succinct Representation of Linear Extensions via MDDs and Its Application to Scheduling under Precedence Constraints

Fumito Miyake[1][0000−0001−9008−7477], Eiji Takimoto[1][], and Kohei Hatano[1,2][0000−0002−1536−1269]

[1] Department of Informatics, Kyushu University, Fukuoka, Japan
{fumito.miyake, eiji, hatano}@inf.kyushu-u.ac.jp
[2] RIKEN AIP

**Abstract.** We consider a single machine scheduling problem to minimize total flow time under precedence constraints, which is NP-hard. Matsumoto et al. proposed an exact algorithm that consists of two phases: first construct a Multi-valued Decision Diagram (MDD) to represent feasible permutations of jobs, and then find the shortest path in the MDD which corresponds to the optimal solution. Although their algorithm performs significantly better than standard IP solvers for problems with dense constraints, the performance rapidly diminishes when the number of constraints decreases, which is due to the exponential growth of MDDs. In this paper, we introduce an equivalence relation among feasible permutations and show that it suffices to construct an MDD that maintains only one representative for each equivalence class. Experimental results show that our algorithm outperforms Matsumoto et al.'s algorithm for problems with sparse constraints, while keeping good performance for dense constraints. Moreover, we show that Matsumoto et al.'s algorithm can be extended for solving a more general problem of minimizing weighted total flow time.

**Keywords:** combinatorial optimization · job scheduling · precedence constraints · MDD

## 1 Introduction

A single machine scheduling problem to minimize total flow time under precedence constraints($1|prec|\sum c_j$) is fundamental in the scheduling literature. The problem is, given processing times of $n$ jobs and precedence constraints for pairs of jobs, to find an order of $n$ jobs (permutation) which minimizes the sum of wait times and process times of all the jobs (called flow time) among those satisfying the precedence constraints [2]. The problem is known to be NP-hard [12,13] and various 2-approximate polynomial time algorithms are proposed [4,5,10,14,20].

On the other hand, non-trivial exact algorithms for solving the problems are not known until recently, except standard methods on integer programming

formulations. Matsumoto et al. [15] proposed an exact algorithm using a variant of the Multi-valued Decision Diagrams(MDD) [16]. It can efficiently construct a MDD expressing linear extensions from given precedence constraints, and solve the problem in linear time in terms of the size of the diagram, by reducing the scheduling problem to the shortest path problem on the diagram. The algorithm of Matsumoto et al. outperformed standard IP-based methods for synthetic problems with dense precedence constraints. However, Their algorithm performs worse when constraints are sparse, where the resulting MDDs have exponentially large size.

In this paper, we propose a more efficient exact algorithm which overcomes the weakness of Matsumoto et al.'s algorithm. Our key idea is to introduce some equivalence relations between linear extensions and exploit the equivalence properties to obtain a more succinct MDD which represents the equivalent feasible solutions.

In the experiments on synthetic data sets, our method performs more than 10 times faster than standard IP-based methods and improves Matsumoto et al.'s method for sparse constraints at the same time.

Moreover, we show that Matsumoto et al.'s algorithm can be extended for solving a more general problem of minimizing weighted total flow time under precedence constraints($1|prec|\sum w_j c_j$).

### 1.1 Comparison to previous work

Succinct data structures such as BDDs (Binary Decision Diagrams) [1,3], ZDDs (Zero Suppressed BDDs) [11,17,18] and MDDs are used for counting and solving NP-hard problems. To the best of our knowledge, there are few diagram-based approaches to scheduling problems except Matsumoto et al. [15], and Ciré and van Hoeve [6, 7]. The work of Ciré and van Hoeve [6, 7] deals with a different scheduling problem which has release times and deadlines and not applicable to ours.

A notable advantage of our method and other diagram-based methods over standard IP-based ones is that once the feasible set is represented by a diagram, we can reuse the diagram for different objectives, which save much computation time. This is also advantageous for the online setting, e.g. [8, 19].

## 2 Preliminaries

Let $[n] = \{1, 2, \ldots, n\}$ denote the set of jobs and $S_n$ denote the set of all permutations over $[n]$, where each permutation is represented by a vector $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$. For example, $S_3 = \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$. For a permutation $\boldsymbol{\pi} \in S_n$, we define the inverse permutation of $\boldsymbol{\pi}$ as $\boldsymbol{\pi}^{-1} = (\pi_1^{-1}, \ldots, \pi_n^{-1})$, where $\pi_j^{-1} = i$ if and only if $\pi_i = j$. A permutation $\boldsymbol{\pi} \in S_n$ specifies a job scheduling, i.e., an order of jobs to be processed, in the way that jobs $i$ should be processed in the decreasing order of $\pi_i$. Here, $\pi_i$ can

be interpreted as the priority of job $i$. In other words, the order of jobs specified by a permutation $\boldsymbol{\pi}$ is $\pi_n^{-1} \to \pi_{n-1}^{-1} \to \cdots \to \pi_1^{-1}$.

For each job $i \in [n]$, let $p_i \in \mathbb{R}$ be the processing time, and define the completion time $c_i = \sum_{j:\pi_j \geq \pi_i} p_j$ which is sum of process times and wait times. Then the flow time of a job scheduling $\boldsymbol{\pi} \in S_n$ is defined as $\sum_{i=1}^n c_i$. The flow time of $\boldsymbol{\pi}$ is the sum of completion times over all jobs under the order specified by $\boldsymbol{\pi}$. For example, a permutation $\boldsymbol{\pi} = (4, 2, 1, 3)$ specifies the order of jobs $1 \to 4 \to 2 \to 3$, and thus the sum of process times and wait times of all the jobs is $p_1 + (p_1 + p_4) + (p_1 + p_4 + p_2) + (p_1 + p_4 + p_2 + p_3)$, which amounts to $4p_1 + 3p_4 + 2p_2 + p_3 = \boldsymbol{\pi} \cdot \boldsymbol{p}$.

Precedence constraints over the jobs are represented as a directed acyclic graph (DAG) $G = ([n], E)$, where a directed edge $(i, j) \in E$ represents the constraint that job $i$ has to be done before job $j$. We call $G$ a constraint graph. We denote by $S_G$ the set of permutations satisfying the precedence constraints specified by $G$ (i.e., linear extensions of $G$). More specifically,

$$S_G = \{\boldsymbol{\pi} \in S_n \mid \forall (i, j) \in E, \pi_i > \pi_j\}.$$

Similarly, the set $S_G^{-1}$ of inverse permutations in $S_G$ is defined as

$$S_G^{-1} = \{\boldsymbol{\pi}^{-1} \mid \boldsymbol{\pi} \in S_G\}.$$

An example is given in Figure 1.

Now, we are ready to define a single machine scheduling problem to minimize total flow time under precedence constraintsas follows:

$$\begin{aligned} &\text{Input: DAG } G = ([n], E), \text{ process time vector } \boldsymbol{p} \in \mathbb{R}^n \\ &\text{Output: } \boldsymbol{\pi} = \operatorname*{argmin}_{\boldsymbol{\pi} \in S_G} \sum_{i=1}^n c_i = \operatorname*{argmin}_{\boldsymbol{\pi} \in S_G} \boldsymbol{\pi} \cdot \boldsymbol{p} \\ &\text{where, } c_i = \sum_{j:\pi_j \geq \pi_i} p_j \end{aligned} \tag{1}$$

## 3 Previous Work of Matsumoto et al.

We review the previous work of Matsumoto et al. [15], where they propose an algorithm of finding an optimal solution of single machine scheduling problem to minimize total flow time under precedence constraints by using a data structure called a $\pi$-MDD.

In what follows, we identify a permutation $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ with the string $\pi_1 \pi_2 \cdots \pi_n$ of length $n$ by concatenating all components of $\boldsymbol{\pi}$. Then, $S_G^{-1}$ can be regarded as a set of strings. A $\pi$-MDD for a constraint graph $G$ is defined as the smallest DFA[3] that only accepts the strings in $S_G^{-1}$. Figure 2 shows an

[3] More precisely, we omit non-accepting states and transitions to non-accepting states in the automaton.

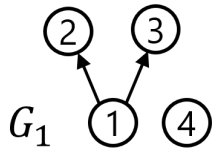


**Fig. 1.** DAG $G_1$ representing precedence constraints
$S_{G_1}^{-1} = \{(2,3,1,4),(2,3,4,1),$
$(2,4,3,1),(3,2,1,4),(3,2,4,1),$
$(3,4,2,1),(4,2,3,1),(4,3,2,1)\}$.

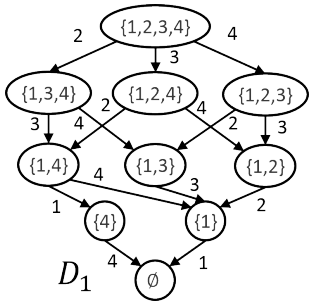


**Fig. 2.** A $\pi$-MDD that represents linear extensions in $S_{G_1}$ satisfying precedence constraints $G_1$ in Figure 1

example of $\pi$-MDDs. We note that (i) a $\pi$-MDD is a DAG with the root (initial state) and a single leaf (unique accepting state) and every edge $e$ is labeled with a job $l(e) \in [n]$. (ii) each path $(e_1, e_2, \ldots, e_n)$ from the root to the leaf corresponds to the permutation $(l(e_1), l(e_2), \ldots, l(e_n))$ that is accepted by the $\pi$-MDD. For a $\pi$-MDD $D$, we denote by $L(D)$ the set of all strings that $D$ accepts. Clearly, a $\pi$-MDD $D$ for $G$ should satisfy $L(D) = S_G^{-1}$. We define the size of $D$, denoted by $|D|$, as the number of edges in $D$.

Matsumoto et al.'s method for solving problem (1) consists of the following two steps.

**Step 1** Construct a $\pi$-MDD $D = (V_D, E_D)$ for a given DAG $G = ([n], E)$. See algorithm MakePiMDD in Algorithm 1. Note that $G|_{V'}$ is the subgraph of $G$ induced by $V'$.

**Step 2** (i) Assign weights $a_e = dp_{l(e)}$ to each edge $e \in E_D$ of $\pi$-MDD $D$, where $d$ is the depth of edge $e$ from the root of $D$. (ii) Find the shortest path $(e_1, e_2, \ldots, e_n)$ from the root to the leaf in the weighted graph obtained in (i). (iii) Output $\boldsymbol{\pi}^*$ where $(\boldsymbol{\pi}^*)^{-1} = (l(e_1), l(e_2), \ldots, l(e_n)) \in S_G^{-1}$.

Below we describe an outline of Algorithm MakePiMDD for step 1.

Given a constraint graph $G = (V, E)$ with $V \subseteq [n]$, MakePiMDD recursively constructs a $\pi$-MDD $D_G$ for $G$. Every node of $D_G$ is associated with a subset of $V$ and thus is identified with the subset. The root and the leaf of $D_G$ correspond to the whole set $V$ and the empty set $\emptyset$, respectively. The root $V$ has outgoing edges to a node $V' \subset V$ if and only if there exists a node $v \in V$ such that (i) $v = V \setminus V'$ and (ii) the out-degree of $v$ is zero in $G$.

The following fact is shown for Step 1.

**Lemma 1 (Matsumoto et al. [15]).** *Given a constraint graph $G = ([n], E)$, MakePiMDD outputs a $\pi$-MDD $D$ such that $L(D) = S_G^{-1}$.*

The correctness of Step 2 can be easily verified from the following observation. Assignment of weights in Step 2 (ii) ensures that for each path $(e_1, e_2, \ldots, e_n)$

**Algorithm 1** MakePiMDD

**Input:** DAG $G = (V, E)$, where $V \subseteq [n]$
1: **if** $V = \emptyset$ **then**
2: **Output:** node $\emptyset$
3: **end if**
4: $D \leftarrow (V_D, E_D)$ with $V_D = \{V\}$ and $E_D = \emptyset$.
5: **for** each $v \in V$ whose outdegree is 0 in $G = (V, E)$ **do**
6: $V' \leftarrow V \setminus \{v\}$
7: **if** have never memorized the $\pi$-MDD $D_G$ for $G$ **then**
8: $D' = (V_{D'}, E_{D'}) \leftarrow$ MakePiMDD$(G|_{V'})$
9: **end if**
10: $V_D \leftarrow V_D \cup V_{D'}$
11: $E_D \leftarrow E_D \cup E_{D'} \cup \{(V, V')\}$
12: **end for**
13: memorize $D$ as $D_G$
14: **Output:** $D_G$

from the root to the leaf and the corresponding permutation $\boldsymbol{\pi}^{-1} = (l(e_1), l(e_2), \ldots, l(e_n))$, the weighted length of the path is exactly the flow time of $\boldsymbol{\pi}$: $\sum_{d=1}^{n} a_{e_d} = \sum_{d=1}^{n} dp_{l(e_d)} = \sum_{d=1}^{n} dp_{\pi_d^{-1}} = \sum_{i=1}^{n} \pi_i p_i = \boldsymbol{\pi} \cdot \boldsymbol{p}$.

Now we give a characterization of $\pi$-MDDs, which is not explicitly given in [15]. Let $<_G$ be the partial order naturally defined by a constraing graph $G = ([n], E)$. That is, for any $u, v \in [n]$, $v \leq_G u$ if and only if there exists a directed path from $u$ to $v$ in $G$. A set $L \subseteq [n]$ is called a lower set of $G$ if for any $u, v \in [n]$, $u \in L$ and $v <_G u$ imply $v \in L$. The family of lower sets of $G$ is denoted as $\mathcal{L}_G \subseteq 2^{[n]}$. The following relationship between the family of lower sets $\mathcal{L}_G$ and linear extensions $S_G$ of $G$ is well-known and easily verified.

**Proposition 1.** *The partially ordered set $(\mathcal{L}_G, \supseteq)$ is a distributive lattice with the minimum element $\emptyset$ and the maximum element $[n]$ and for each maximal chain $L_1 = [n] \supseteq L_2 \supseteq \cdots \supseteq L_{n+1} = \emptyset$, there exists a linear extension $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n) \in S_G$ such that $L_i = L_{i+1} \cup \{\pi_i^{-1}\}$.*

From the proposition above, we immediately get the characterization as described in the following corollary.

**Corollary 1.** *The $\pi$-MDD $D = (V_D, E_D)$ for a constraint graph $G$ is isomorphic to the Hasse diagram of the partially ordered set $(\mathcal{L}_G, \supseteq)$. That is, there exsits a one-to-one correspondence between $V_D$ and the lower sets in $\mathcal{L}_G$ (the root and the leaf correspond to $[n]$ and $\emptyset$, respectively) and for any $L, L' \in \mathcal{L}$ such that $L = L' \cup \{u\}$, there exists a directed edge $e = (L, L')$ with $l(e) = u$ in $D$.*

Using the new characterization, we give a performance bound of Matsumoto et al.'s method in terms of the size of $\mathcal{L}_G$.

**Theorem 1 ( [15]).** *For any constraint graph $G = ([n], E)$, we can find an optimal solution of problem (1) in $O(n|\mathcal{L}_G|)$ time.*

## 4 Main result

Intuitively, when given fewer constraints in $G$ we would have an exponentially many lower sets in $\mathcal{L}_G$ and thus Matsumoto et al.'s method would take exponential time as suggested in Theorem 1. In particular, the worst case occurs when no constraints are given (i.e., no edges in the constraint graph $G$). In this case, any subset in $[n]$ is a lower set and thus we have $\mathcal{L}_G = 2^{[n]}$. On the other hand, if no constraints are given, then we can easily obtain an optimal solution: process jobs $i$ in the increasing order of the process time $p_i$. In other words, we can solve the problem in $O(n \log n)$ time by just sorting $(p_1, p_2, \ldots, p_n)$.

Our method is based on the observation above. Suppose we are given a subset $A$ of jobs that can be somehow regarded as a no constraint set with respect to the linear extensions $S_G$, then we can find an optimal (partial) solution over $A$ by sorting. So when constructing $\pi$-MDD for $G$, we do not need to consider all $|A|!$ permutations over $A$ but it suffices to fix a representative permutation over $A$, which reduces the number of paths by a factor of $1/|A|!$. In this way, we construct a succinct $\pi$-MDD that accepts only representative permutations.

### 4.1 Equivalence relation

**Definition 1 (Input and output sets).** *For a DAG $G = ([n], E)$, and each node $v \in [n]$, input set $I_v$ and output set $O_v$ of $v$ are defined respectively as follows:*

$$I_v = \{v' \in [n] \mid (v', v) \in E\}, O_v = \{v' \in [n] \mid (v, v') \in E\}.$$

**Definition 2 (Equivalence relations between jobs).** *Given a DAG $G = ([n], E)$, job $i$ and $j$ are equivalent if and only if $I_i = I_j$ and $O_i = O_j$ and denoted as $i \simeq_G j$. If it is clear from the context, we abbreviate $i \simeq j$.*

The equivalence relation $\simeq_G$ implies a partition over $[n]$

$$[n] = A_1 \cup A_2 \cup \cdots \cup A_N,$$

where each $A_j$ $(j = 1, \ldots, N)$ is an equivalence set defined by $\simeq_G$. We call each $A_j$ a job equivalence set. Figure 3 shows an illustration of job equivalence sets.

For a permutation $\boldsymbol{\pi} \in S_n$, a set $A \subseteq [n]$, and a bijection $\delta : A \to A$, let $\boldsymbol{\pi} \circ \delta$ is defined as

$$(\boldsymbol{\pi} \circ \delta)_i = \begin{cases} \pi_{\delta(i)} & i \in A, \\ \pi_i & \text{otherwise.} \end{cases}$$

Then we define a equivalence relation between linear extensions in $S_G$.

**Definition 3 (Equivalence relations between linear extensions).** *Permutations $\boldsymbol{\pi}, \boldsymbol{\pi}' \in S_n$ are equivalent w.t.t. a DAG $G = ([n], E)$ if and only if for each job equivalence set $A_j$, there exists a bijection $\delta_{A_j} : A_j \to A_j$ such that*

$$\boldsymbol{\pi'} = \boldsymbol{\pi} \circ \delta_{A_1} \circ \delta_{A_2} \circ \cdots \circ \delta_{A_N}.$$

*We denote $\boldsymbol{\pi} \simeq_G \boldsymbol{\pi}'$ if $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$ are equivalent w.r.t. $G$. When clear from the context, we abbreviate as $\boldsymbol{\pi} \simeq \boldsymbol{\pi'}$.*

Figure 4 shows an illustration of equivalence relation between permutations. For clarity, we used expression of inverse permutation.

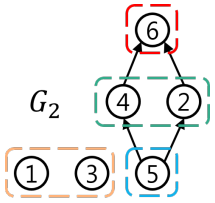


**Fig. 3.** Ilustration of equivalence sets defined by a DAG $G_2$

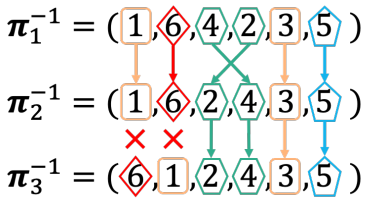


**Fig. 4.** For a DAG $G_2$, $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$ are equivalent, but $\boldsymbol{\pi}_2$ and $\boldsymbol{\pi}_3$ are not.

The following proposition holds for the permutations which are equivalence each other.

**Proposition 2.** *For any permutations* $\boldsymbol{\pi}, \boldsymbol{\pi'} \in S_G$, $\boldsymbol{\pi} \simeq \boldsymbol{\pi'} \iff \forall j \in [n], \pi_j^{-1} \simeq \pi'^{-1}_j$ .

The following lemma holds for the equivalence relation on $S_G$.

**Lemma 2.** *For any permutations* $\boldsymbol{\pi}, \boldsymbol{\pi'} \in S_n$, $\boldsymbol{\pi} \in S_G$ *and* $\boldsymbol{\pi'} \simeq \boldsymbol{\pi}$ *implies* $\boldsymbol{\pi'} \in S_G$.

It can be proved easily by the symmetry between equivalence jobs on $G$.

For each equivalence class $[\boldsymbol{\pi}]$, we define the representative $\tilde{\boldsymbol{\pi}}$ as the permutation satisfying (i) $\tilde{\boldsymbol{\pi}} \simeq \boldsymbol{\pi}$, and (ii) for any $k, l \in A_j (k < l)$, $\tilde{\pi}_k > \tilde{\pi}_l$. In other words, $\tilde{\boldsymbol{\pi}}$ is the particular representative of $[\boldsymbol{\pi}]$ satisfying additional precedence constraints defined with job id numbers.

### 4.2 Our algorithm

We propose an exact algorithm for solving problem (1). Our algorithm consists of the following steps.

1. Compute job equivalence sets $A_1, \ldots, A_N$.
2. Construct a DAG $\tilde{G}$ satisfying $S_{\tilde{G}} = \{\tilde{\boldsymbol{\pi}} \mid \boldsymbol{\pi} \in S_G\}$
3. Run the algorithm of Matsumoto et al. [15] with input $\tilde{G}$ and obtain a $\pi$-MDD $\tilde{D}$ representing $S_{\tilde{G}}$.
4. For some appropriate weights over edges in $\tilde{D}$, solve the shortest path problem and construct an optimal solution of (1) from it.

For step 1, we compute a adjacency matrix of $G$, $X \in \mathbb{R}^{n\times n}$, that is, $X(i,j) = 1$ when there is a directed edge from job $i$ to job $j$, otherwise $X(i,j) = 0$. Then, we define matrix $Z \in \mathbb{R}^{n\times 2n}$ as follows

$$Z(i,j) = \begin{cases} X(i,j) & (j \le n) \\ X^\top(i, j-n) & (j > n). \end{cases}$$

Now, we make matrix $Z' \in \mathbb{R}^{n\times 2n}$ by regarding each rows of $Z$ as individual strings, and sorting with radix-sort. Since $v \simeq v' \Leftrightarrow \forall i \in [n]\ Z'(v,i) = Z'(v',i)$, strings corresponding to job-equivalent nodes are the same. By checking the equivalence, we can find the equivalent classes. These procedures can be done in time $O(n^2)$.

In Step 2, for each equivalence class $A_j$ ($j \in [N]$), sort its elements $v_1 < v_2 < \cdots < v_{|A_j|}$ and add edges $(v_i, v_{i+1})$ for $1 \le i \le |A_j| - 1$ to the DAG $G$, which we denote as $\tilde{G}$ (Figure 5) shows an illustration of $\tilde{G}$. Computation time for Step 2 is $O(n)$.

Step 3 takes $O(n|\mathcal{L}_{\tilde{G}}|)$ time to obtain a $\pi$-MDD $\tilde{D}$ by using Matsumoto et al.'s algorithm.

In Step 4, we first sort the weight vector $\boldsymbol{p}$ for each job equivalent set. To do so, we prepare a $n$-dimensional array $C$ and sort $\boldsymbol{p}$ in the following way.

1. For any $i \in [n]$, $i \simeq C[i]$.
2. For any $i, j \in [n]$ such that $i \simeq j$, $i < j$ implies $p_{C[i]} \le p_{C[j]}$.

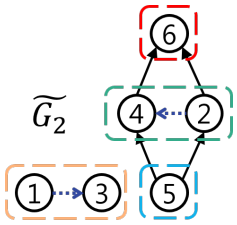


**Fig. 5.** DAG $\tilde{G_2}$ obtained by adding edges to the DAG $G_2$ in Figure 3

Then, for each edge $e \in E_{\tilde{D}}$, we assign a weight $dp_{C[l(e)]}$, where $d$ is the depth of the edge $e$ from the root node and $l(e)$ is label of $e$, respectively. For the assigned weights, we solve the shortest path problem over $\tilde{D}$ and obtain the shortest path $(e_1, \ldots, e_n)$ from the root node to the leaf node. Finally we output $\boldsymbol{\pi}^*$ such that $\boldsymbol{\pi}^{*-1} = (C[l(e_1)], \ldots, C[l(e_n)]) \in S_G^{-1}$. Computation time for step 4 is $O(n \log n + n|\mathcal{L}_{\tilde{G}}|)$.

We now state our main result.

**Theorem 2.** *Our algorithm solves problem (1) in time* $O(n^2 + n|\mathcal{L}_{\tilde{G}}|) = O(n|\mathcal{L}_{\tilde{G}}|)$.

*Proof.* First of all, we show that the set $S_{\tilde{G}}$ of linear extensions of $\tilde{G}$ satisfies $S_{\tilde{G}} = \{\tilde{\boldsymbol{\pi}} \mid \boldsymbol{\pi} \in S_G\}$. For any $\boldsymbol{\pi} \in S_{\tilde{G}}$, it is clear that $\boldsymbol{\pi} \in S_G$ by the construction of $\tilde{G}$. Furthermore, since $\boldsymbol{\pi}$ satisfies additional precedence constraints regarding job equivalence classes $A_j$s, for any jobs $k, l \in A_j$, $k < l$ implies $\pi_k > \pi_l$. This means that $\boldsymbol{\pi}$ is a representative for some $[\boldsymbol{\pi}']$ in $S_G$. Therefore, $S_{\tilde{G}} \subseteq \{\tilde{\boldsymbol{\pi}} \mid \boldsymbol{\pi} \in S_G\}$. For any $\boldsymbol{\pi} \in S_G$, by definition, the representative $\tilde{\boldsymbol{\pi}} of [\boldsymbol{\pi}]$ satisfies

additional precedence constraints for job equivalence classes. Thus $\tilde{\boldsymbol{\pi}} \in S_{\tilde{G}}$, implying $S_{\tilde{G}} \supseteq \{\tilde{\boldsymbol{\pi}} \mid \boldsymbol{\pi} \in S_G\}$.

Then, we prove the correctness of step 4.

From lemma 2, for any permutation $\boldsymbol{\pi} \in S_{\tilde{G}}$, it satisfies constraints represented with $\tilde{G}$ even if the elements of any equivalence classes are arbitrarily rearranged within the classes. Now, the order of elements of any equivalence class $A = \{i_1, i_2, \ldots, i_{|A|}\} \subseteq [n]$ with a smallest flow time is obviously the order such that for $i, j \in A (i \neq j)$ if $\pi_i > \pi j$, then $p_i < p_j$. Therefore, in the first half of step 4, for any equivalence class $A \subseteq [n]$, we make rules for converting the series of job number that job $i, j \in A (s.t.\ \pi_i > \pi_j)$ satisfies $i < j$ to the series of job number that job $i, j \in A (s.t.\ \pi_i > \pi_j)$ satisfies $p_i \leq p_j$ as array $C$. Accordingly, the conversion by array $C$ can be regarded as playing the role of following function $F : S_{\tilde{G}} \to S_G$

$$F(\tilde{\boldsymbol{\pi}}) = \underset{\boldsymbol{\pi} \in [\tilde{\boldsymbol{\pi}}]}{\operatorname{argmin}} \boldsymbol{\pi} \cdot \boldsymbol{p}.$$

Thus, in $\pi$-MDD $\tilde{D}_C$ obtained by replace each edge label $l(e)$ of $\pi$-MDD $\tilde{D}$ with $C[l(e)]$, any path corresponds to the permutation with a smallest flow time in equivalence class which the permutation belongs. In the latter half of step 4, it can be regarded as solving the shortest path problem on $\pi$-MDD $\tilde{D}_C$, so the solution of the problem is also the inverse permutation of optimal solution of the original problem. Hence, by evaluating the inverse permutation of that permutation, we can get optimal solution. □

At last, we show that the size of $\pi$-MDD constructed by our method doesn't become large than that of $\pi$-MDD constructed by Matsumoto et al.'s method.

**Theorem 3.** *Let $D$ be the $\pi$-MDD representing linear extensions $S_G$ of $G$, and let $\tilde{D}$ be the $\pi$-MDD representing linear extensions $S_{\tilde{G}}$ of $\tilde{G}$, $|\tilde{D}| \leq |D|$.*

*Proof.* The DAG $\tilde{G}$ can be obtained by adding constraints to $G$, so any lower set of $\tilde{G}$ is also a lower set of $G$. Accordingly, it holds that $\mathcal{L}_{\tilde{G}} \subseteq \mathcal{L}_G$. By Corollary 1, $\tilde{D}$ is isomorphic to the Hasse diagram of the partially ordered set $(\mathcal{L}_{\tilde{G}}, \supseteq)$. Clearly, the partially ordered set $(\mathcal{L}_{\tilde{G}}, \supseteq)$ is obtained by restricting $(\mathcal{L}_G, \supseteq)$ on the domain $\mathcal{L}_{\tilde{G}}$. Therefore, $\tilde{D}$ is the subgraph of $D$ induced by the subset $\mathcal{L}_{\tilde{G}}$ of states. □

## 5 Extension to $1|prec|\sum w_j c_j$

In this section, we show that Matsumoto et al.'s algorithm can be extended for solving a more general problem of minimizing weighted total flow time.

We consider about extensive setting which each job $i \in [n]$ has weight $w_i \in \mathbb{R}_+$. Now, we define a single machine scheduling problem to minimize weighted

total flow time under precedence constraints($1|prec|\sum w_j c_j$) as follows [2].

$$\begin{aligned} \text{Input: } & \text{DAG } G = ([n], E) \text{ process time vector } \boldsymbol{p} \in \mathbb{R}^n \\ & \text{weight vector } \boldsymbol{w} \in \mathbb{R}^n_+ \\ \text{Output: } & \boldsymbol{\pi} = \operatorname*{argmin}_{\boldsymbol{\pi} \in S_G} \boldsymbol{w} \cdot \mathbf{c} \\ & \text{where, } c_i = \sum_{j:\pi_j \geq \pi_i} p_j \end{aligned} \tag{2}$$

Think about for every $i \in [n]$, $w_i = 1$, the flow time of (2) equals the flow time of (1). Thus, the problem (2) is clearly generalized setting of the problem (1).

For this problem, we propose simple extension of Matsumoto et al.'s method which there is only a difference point. The **Step 2** of Matsumoto et al.'s method in section 3, assigns weights $v_e = dp_{l(e)}$ to each edge $e \in E_D$ of $\pi$-MDD $D = (V_D, E_D)$. For problem (2), instead, introduce cumulative weight $\hat{w}_e = \sum_{i:[n]\setminus V_e} w_i$ and assign $a_e = \hat{w}_e p_{l(e)}$. Then, for each path $(e_1, e_2, \ldots, e_n)$ from the root to the leaf and the corresponding permutation $\boldsymbol{\pi}^{-1} = (l(e_1), l(e_2), \ldots, l(e_n))$, the weighted length of the path is exactly the flow time of $\boldsymbol{\pi}$. Certainly, computation time is equal to the time of Matsumoto et al.'s method. Now, we show a theorem.

**Theorem 4.** *Our algorithm solves problem (2) in time $O(n|\mathcal{L}_G|)$.*

The proof is omitted and shown in Appendix.

## 6 Experiments

In this section, we compare the efficiency of proposed method and previous methods for the scheduling problem with precedence constraints on artificial data sets.

### 6.1 Settings of artificial data sets and methods

As artificial data sets, we generate Erdös-Rényi random graph $G_{n,q}$ as constraints $G = ([n], E)$. That is, over $G = ([n], E)$, for each job $i, j \in [n](i < j)$, there exists $(i, j) \in E$ with probability $0 \leq q \leq 1$ [4]. Also, we chose processing time vectors $\boldsymbol{p}$ according to the uniform distribution over $[0, 1]^n$.

We compare the proposed method, Matsumoto et al.'s method, and integer programming (IP) with permutation matrices and comparison matrices [5], respectively. Let $n = 30$, and for each $q \in \{0.01, 0.02, \ldots, 1\}$, we generate 10 random graphs and processing time vectors, and observe the average of computation times of each method and sizes of $\pi$-MDD constructed by the proposed method and Matsumoto et al.'s method. These methods are implemented by C++ with Gurobi optimizer 8.1.0 [9] to solve integer programs. We run them in a machine with Intel(R) Xeon(R) Processor X5560 2.80GHz and 198GB memory.

[4] Note that, because of the constraint that $i < j$, the graph $G = ([n], E)$ is a DAG.
[5] Please refer to [15] for details.

### 6.2 Results and Discussion

Figure 6 shows the computation times(in the logarithmic scale) of each method for different choices of $q \in \{0.01, 0.02, \ldots, 1\}$. Figure 7 shows the size of $\pi$-MDD(logarithmic axis) generated by the proposed method and Matsumoto et al.'s method for different choices of $q \in \{0.01, 0.02, \ldots, 1\}$.

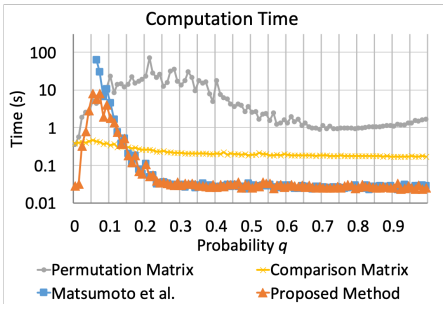


**Fig. 6.** Average computation time for $n = 24, q \in \{0.01, 0.02, \ldots, 1\}$

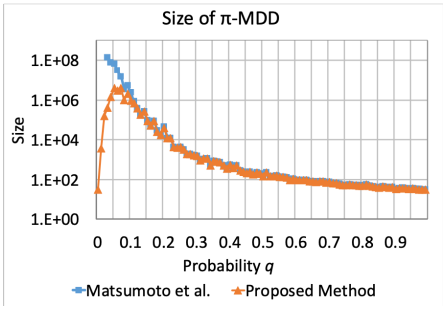


**Fig. 7.** Average size of $\pi$-MDD for $n = 24, q \in \{0.01, 0.02, \ldots, 1\}$

These results show that the proposed method is fastest and most space-efficient among others for any $q$. Also, the results of Matsumoto et al.'s method show that for sparse precedence constraints, its computational complexity become much worse. However, with proposed method, even for sparse constraints its computational time is moderately small and still smallest among others.

## 7 Conclusion and Future Work

We proposed an improved algorithm of Matsumoto et al. which exploits the symmetry in permutations satisfying precedence constraints by introducing a notion of equivalence class among jobs. Our future work includes improving our algorithm for the cases where conventional IP solvers are still advantageous. Also, extension of our algorithm for weighted flow time is still open. In addition, it may be possible to construct only necessary parts of $\pi$-MDD dynamically, which would further improve our algorithm.

## References

1. Akers, S.B.: Binary Decision Diagrams. IEEE Transactions on Computers **C-27**(6), 509–516 (1978)
2. Brucker, P.: Scheduling algorithms (4. ed.). Springer (2004)

3. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers **C-35**(8), 677–691 (aug 1986)
4. Chekuri, C., Motwani, R.: Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. Discrete Applied Mathematics **98**(1-2), 29–38 (1999)
5. Chudak, F.A., Hochbaum, D.S.: A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. Oper. Res. Lett. **25**(5), 199–204 (1999)
6. Ciré, A.A., van Hoeve, W.J.: MDD Propagation for Disjunctive Scheduling. In: Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12) (2012)
7. Ciré, A.A., van Hoeve, W.J.: Multivalued Decision Diagrams for Sequencing Problems. Operations Research **61**(6), 1411–1428 (2013)
8. Fujita, T., Hatano, K., Kijima, S., Takimoto, E.: Online Linear Optimization for Job Scheduling under Precedence Concstraints. In: Proceedings of 26th International Conference on Algorithmic Learning Theory(ALT 2015). LNCS, vol. 6331, pp. 345–359 (2015)
9. Gurobi Optimization, I.: Gurobi optimizer reference manual (2018), `http://www.gurobi.com`
10. Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. Math. Oper. Res. **22**(3), 513–544 (1997)
11. Knuth, D.E.: The Art of Computer Programming, Volume 4A, Combinatorial Algorithms,Part 1. Addison-Wesley Professional (2011)
12. Lawler, E.L.: On Sequencing jobs to minimize weighted completion time subject to precedence constraints (1978)
13. Lenstra, J.K., Kan, A.H.G.R.: Complexity of scheduling under precedence constraints. Operations Research **26**(1), 22–35 (1978)
14. Margot, F., Queyranne, M., Wang, Y.: Decompositions, network flows, and a precedence constrained single-machine scheduling problem. Operations Research **51**(6), 981–992 (2003)
15. Matsumoto, K., Hatano, K., Takimoto, E.: Decision diagrams for solving a job scheduling problem under precedence constraints. In: SEA 2018 (2018)
16. Miller, M.D., Drechsler, R.: Implementing a Multiple-Valued Decision Diagram Package. In: Proceedings of the 28th IEEE International Symposium on Multiple-Valued Logic (ISMVL'98). pp. 52–57 (1998)
17. Minato, S.i.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In: Proceedings of the 30th international Design Automation Conference (DAC'93). pp. 272–277 (1993)
18. Minato, S.i.: Zero-suppressed BDDs and their applications. International Journal on Software Tools for Technology Transfer **3**(2), 156–170 (2001)
19. Sakaue, S., Ishihata, M., Minato, S.i.: Efficient Bandit Combinatorial Optimization Algorithm with Zero-suppressed Binary Decision Diagrams. In: Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics (AISTATS 2018). PMLR, vol. 84, pp. 585–594 (2018), `http://proceedings.mlr.press/v84/sakaue18a.html`
20. Schulz, A.S.: Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In: Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3-5, 1996, Proceedings. pp. 301–315 (1996)

# Appendix

### Proof of theorem 4.

We show the proof of theorem 4.

*Proof.* We assign $a_e = \hat{w}_e p_{l(e)}$ to each edge $e \in E_D$ of $\pi$-MDD $D = (V_D, E_D)$. Then, for each path $(e_1, e_2, \ldots, e_n)$ from the root to the leaf and the corresponding permutation $\boldsymbol{\pi}^{-1} = (l(e_1), l(e_2), \ldots, l(e_n))$, the weighted length of the path can be calculated as follows.

$$\begin{aligned}
\sum_{d=1}^{n} a_{e_d} &= \sum_{d=1}^{n} \hat{w}_{e_d} p_{l(e_d)} \\
&= \sum_{d=1}^{n} \left( \sum_{i:[n]\setminus V_{e_d}} w_i \right) p_{l(e_d)} \\
&= \sum_{d=1}^{n} \left( \sum_{i:\pi_i \le \pi_{l(e_d)}} w_i \right) p_{l(e_d)} \\
&= \sum_{d=1}^{n} \left( \sum_{i:\pi_i \le \pi_{\pi_d^{-1}}} w_i \right) p_{\pi_d^{-1}} \\
&= \sum_{j=1}^{n} \left( \sum_{i:\pi_i \le \pi_j} w_i \right) p_j \\
&= \sum_{i=1}^{n} w_i \left( \sum_{j:\pi_j \ge \pi_i} p_j \right) = \boldsymbol{w} \cdot \mathbf{c}
\end{aligned}$$

Now, we get the flow time of problem (2) . □