


Decision Diagrams for Solving a Job Scheduling Problem Under Precedence Constraints

Kosuke Matsumoto

Kyushu University
matumoabyss@gmail.com

Kohei Hatano

Kyushu University / RIKEN AIP
hatano@inf.kyushu-u.ac.jp
 <https://orcid.org/0000-0002-1536-1269>

Eiji Takimoto

Kyushu University
eiji@inf.kyushu-u.ac.jp

Abstract

We consider a job scheduling problem under precedence constraints, a classical problem for a single processor and multiple jobs to be done. The goal is, given processing time of n fixed jobs and precedence constraints over jobs, to find a permutation of n jobs that minimizes the total flow time, i.e., the sum of total wait time and processing times of all jobs, while satisfying the precedence constraints. The problem is an integer program and is NP-hard in general. We propose a decision diagram π -MDD, for solving the scheduling problem exactly. Our diagram is suitable for solving linear optimization over permutations with precedence constraints. We show the effectiveness of our approach on the experiments on large scale artificial scheduling problems.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases decision diagram, permutation, scheduling, NP-hardness, precedence constraints, MDD

Digital Object Identifier 10.4230/LIPIcs.SEA.2018.5

Supplement Material Source codes are available at https://bitbucket.org/kohei_hatano/pimdd/.

Funding This work is supported in part by JSPS KAKENHI Grant Number JP16K00305 and JSPS KAKENHI Grant Number JP15H02667, respectively.

Acknowledgements We thank Fumito Miyake for insightful discussions and anonymous reviewers for helpful comments.

1 Introduction

Scheduling problems are typical problems which are known to be NP-hard in general. Hence, practical fast approximate solvers for scheduling are quite useful in practice. Among them, the job scheduling problem of a single machine with precedence constraints is a classical one, where, given n fixed jobs and their processing times, as well as precedence constraints over jobs (i.e., job i must be done prior to job j), the task is to find a permutation of n jobs (a schedule) which minimizes the sum of processing times and wait times (called flow time) of all jobs among those permutations satisfying precedence constraints. This



© Kosuke Matsumoto, Kohei Hatano, and Eiji Takimoto;
licensed under Creative Commons License CC-BY
17th International Symposium on Experimental Algorithms (SEA 2018).

Editor: Gianlorenzo D'Angelo; Article No. 5; pp. 5:1–5:12



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem is NP-hard [15, 16] as well and 2-approximation polynomial time algorithms are known [24, 11, 5, 17, 4]. For further details, see, e.g., [8, 2]. On the other hand, practical exact algorithms are quite non-trivial to obtain; Naive algorithms using integer programming solvers still take prohibitive time.

BDDs (Binary Decision Diagrams) [1, 3] and ZDDs (Zero Suppressed BDDs) [20, 21, 14] are data structures which represent sets of binary vectors (or sets of fixed objects). BDDs/ZDDs can compress sets succinctly and various functions over sets (such as union and intersection) are efficiently computed using the data structures. In particular, ZDDs are suitable for representing sparse sets and often advantageous in practice (see, e.g., [22, 12]). A variant of ZDDs called π DDs are specially designed for representing permutations [23]. The structure is suitable for counting or enumeration, but not designed for optimization. In addition, we are not aware of other non-trivial applications of BDDs/ZDDs for the scheduling problem with precedence constraints. MDDs (Multiple-Valued Decision Diagrams)[19] are of variants of BDDs which can treat multiple values naturally and applications of MDDs to scheduling problems are known [6, 7]. The scheduling problems considered are different from ours and thus are not applicable.

In this paper, we propose a data structure π -MDD, which directly represents a set of permutations over $\{1, \dots, n\}$ ¹. A π -MDD is a DAG and each path in the DAG represents a permutation. Using the data structure, we show an exact optimization scheme for the job scheduling problem under precedence constraints. More specifically, our scheme consists of the following two parts.

- (i) We propose an algorithm which, when given a set of precedence constraints represented by a DAG as input, constructs a π -MDD representing permutations satisfying the precedence constraints in output-linear time. We show that the size of the π -MDD made by the algorithm is $O(h(G)(n/h(G) + 1)^{h(G)})$, where G is the DAG representing precedence constraints and $h(G)$ is the width of the graph.
- (ii) Given a π -MDD which represents a set of permutations, and processing times of jobs, we show a method for finding a permutation π optimizing the flow time among the set which is a linear optimization over the permutations in the set. Like BDDs/ZDDs, linear optimization of the set of interest can be reduced to the shortest path problem over the corresponding MDD which represents the set. Thus the computation time for the optimization is linear in the size of the π -MDD.

A potential advantage of our method (and other BDDs/ZDDs/MDDs based approaches) over naive integer-programming based methods is that once we construct a π -MDD representing permutations satisfying precedence constraints, we can re-use it for different cost criteria without reconstructing π -MDDs. This advantage is crucial for (i) the case where several different cost criteria are considered and (ii) a repeated game version of the job scheduling under fixed precedence constraints under uncertainty (see, e.g., [9]).

In our preliminary experiments over large artificial data sets of job scheduling under precedence constraints, our method outperforms naive methods based on the integer programming, especially when there are more precedence constraints.

1.1 Related Work

Note that the data structure π -MDD is a special case of MDDs and not new itself. Furthermore, the structure of π -MDD is quite similar to ones used in the previous work of Hadzic et al. [10] and Ciré and van Hoesve [6, 7]. However, their approach to construct the data

¹ More precisely, π -MDDs can deal with permutations over n fixed different numbers.

structure is totally different from ours. Their approach is to construct a relaxed MDD which represents a super set of the feasible solutions first and to solve the problem by refining the MDD as well as filtering infeasible solutions. On the other hand, our approach directly constructs the exact set of feasible solutions.

The technical contribution of the paper is not to derive a new data structure, but to derive an efficient construction method of π -MDDs satisfying precedence constraints as well as an efficient exact optimization of the job scheduling problem using the structure.

2 Preliminaries

2.1 Notations

Let $[n] = \{1, 2, \dots, n\}$ be the set of integers $1, \dots, n$. A permutation π over $[n]$ is a bijection from $[n]$ to $[n]$. Each permutation π can be represented as the corresponding vector $\boldsymbol{\pi} = (\pi(1), \dots, \pi(n))$. For convenience, for each $i \in [n]$, let π_i be the i -th element of $\boldsymbol{\pi}$, and π_i^{-1} be the position of element i , respectively. Let $S_{[n]}$ be the set of permutations over $[n]$. For example, $S_{[3]} = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$.

A directed graph (DAG) $G = (V, E)$ is a pair, where V is the set of nodes and $E \subseteq V \times V$ is the set of directed edges in which there is no directed cycle, i.e., there is no sequence $(v_1, v_2), (v_2, v_3) \dots, (v_k, v_{k+1}) \in E^k$ with $v_1 = v_{k+1}$.

Let d_v^+ denote the out-degree of node v in V , that is $d_v^+ = |\{v' \in V \mid (v, v') \in E\}|$. We say that a node $v \in V$ is reachable from $v' \in V$ if there is a directed path starting from v' ending at v , i.e., a sequence of directed edges $(v', v_1), (v_1, v_2), \dots, (v_{k-1}, v_k), (v_k, v) \in E$. For a DAG $G = (V, E)$, the width $h(G)$ denote the maximum size of the set $V' \subseteq V$ where each pair of nodes are not reachable from each other. A partially ordered set (poset) (P, R) is a pair, where P is a set and $R \subset P \times P$ is a binary relation satisfying reflexivity ($\forall a \in P, aRa$), transitivity ($\forall a, b, c \in P, aRb$ and bRc implies aRc), and antisymmetry ($\forall a, b \in P, aRb$ and bRa implies $a = b$). A poset (P, R) can be viewed as a DAG $G = (V, E)$ with $V = P$ and $R = E$ and known as a Hasse diagram.

Let $S_V(G)$ denote the set of permutations over $V \subseteq [n]$ satisfying the precedence constraints corresponding to the DAG $G = (V, E)$ and is defined as

$$S_V(G) = \{\boldsymbol{\pi} \in S_V \mid \forall (v, v') \in E \ \pi_v < \pi_{v'}\}.$$

Similarly, let $S_V^{-1}(G)$ the set of inverses of permutations in $S_V(G)$, i.e.,

$$S_V^{-1}(G) = \{\boldsymbol{\pi}^{-1} \in S_V \mid \boldsymbol{\pi} \in S_V(G)\}$$

Note that, by definition of the inverse, $S_V^{-1}(G) = \{\boldsymbol{\pi} \in S_V \mid \forall (v, v') \in E \ \pi_v^{-1} < \pi_{v'}^{-1}\}$. We will make use of this property extensively in later discussions.

2.2 The job scheduling problem under precedence constraints

We consider the job scheduling problem of n jobs with a single machine under the precedence constraints given as a DAG $G = ([n], E)$. Given processing times of jobs represented as a vector $\boldsymbol{w} \in \mathbb{R}^n$ and the precedence constraints G , the task is to find a permutation over the set $[n]$ of jobs minimizing the sum of flow times (the sum of processing time and wait time) of all jobs. For example, when $n = 4$, jobs 3, 2, 4, 1 are done successively, flow times of these jobs are $w_3, w_3 + w_2, w_3 + w_2 + w_4, w_3 + w_2 + w_4 + w_1$, respectively and the sum of flow times is $4w_3 + 3w_2 + 2w_4 + w_1$. If we represent the schedule as a permutation $\boldsymbol{\pi} = (1, 3, 4, 2)$ (each component i can be viewed as its priority), the sum of flow times of the schedule is exactly

the inner product $\boldsymbol{\pi} \cdot \boldsymbol{w}$. This relationship holds in general. That is, a permutation $\boldsymbol{\pi} \in S_{[n]}$ represents a schedule where the priority of job i is π_i (i.e., the job is the $(n + 1 - \pi_i)$ -th to be done) and the sum of flow times is $\boldsymbol{\pi} \cdot \boldsymbol{w}$.

Now we define the job scheduling problem under the precedence constraints represented as a DAG $G = ([n], E)$ and $\boldsymbol{w} \in \mathbb{R}^n$ as the following linear optimization problem:

$$\begin{aligned} \text{Input} &: \text{ DAG } G = ([n], E), \boldsymbol{w} \in \mathbb{R}^n \\ \text{Output} &: \boldsymbol{\pi}^* = \arg \min_{\boldsymbol{\pi} \in S_{[n]}(G)} \boldsymbol{\pi} \cdot \boldsymbol{w} \end{aligned} \quad (1)$$

The problem can be formulated as an integer program where variables takes values in $[n]$ and it is NP-hard [15, 16]. We will solve this problem exactly using a new data structure later.

This problem can be reduced to the 0-1 integer programs in two ways. The first reduction represents a permutation as a permutation matrix as follows: Given a permutation $\boldsymbol{\pi} \in S_{[n]}$, the corresponding permutation matrix $X \in \{0, 1\}^{n \times n}$ is defined as $X_{i,j} = 1$ if $\pi_j = i$ and otherwise $X_j = 0$ for each $j \in [n]$. For example, for $\boldsymbol{\pi} = (2, 3, 1)$, the permutation matrix X is

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

When we represent permutations as permutation matrices and we are given $\boldsymbol{w} \in \mathbb{R}^n$, let

$$W = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \\ 2w_1 & 2w_2 & \cdots & 2w_n \\ \vdots & \vdots & \ddots & \vdots \\ nw_1 & nw_2 & \cdots & nw_n \end{bmatrix}.$$

Then the problem can be given as the following integer program.

$$\begin{aligned} \text{minimize}_{X \in \{0,1\}^{n \times n}} & \sum_{i=1}^n \sum_{j=1}^n X_{i,j} W_{i,j} \\ \text{subject to} & \end{aligned} \quad (2a)$$

$$\forall i \in [n] \quad \sum_{j=1}^n X_{i,j} = 1 \quad (2b)$$

$$\forall j \in [n] \quad \sum_{i=1}^n X_{i,j} = 1 \quad (2c)$$

$$\forall (v, v') \in E \quad \forall i \in [n] \quad \sum_{j=i}^n X_{j,v} \leq \sum_{j=i}^n X_{j,v'} \quad (2d)$$

This formulation has n^2 variables and $n(|E| + 2)$ linear constraints.

The second reduction to an 0-1 integer program uses a comparison matrix as a representation of a permutation. A comparison matrix $Y \in \{0, 1\}^n$ corresponding to a permutation $\boldsymbol{\pi}$ is defined as

$$Y_{i,j} = \begin{cases} 0 & (\pi_i > \pi_j) \\ 1 & (\pi_i \leq \pi_j) \end{cases} \quad (3)$$

By adopting the comparison matrix representation, the scheduling problem is given as the following integer program with the input

$$W = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \\ w_1 & w_2 & \cdots & w_n \\ \vdots & \vdots & \ddots & \vdots \\ w_1 & w_2 & \cdots & w_n \end{bmatrix}.$$

$$\text{minimize}_{Y \in \{0,1\}^{n \times n}} \sum_{i=1}^n \sum_{j=1}^n Y_{i,j} W_{i,j}$$

subject to

$$\forall i, j, k \in [n] \quad 1 \geq Y_{i,j} - Y_{i,k} + Y_{j,k} \geq 0 \quad (4a)$$

$$\forall i, j \in [n] \quad \begin{cases} Y_{i,j} + Y_{j,i} = 1 & (i \neq j) \\ Y_{i,j} = 1 & (i = j) \end{cases} \quad (4b)$$

$$\forall (v, v') \in E \quad Y_{v,v'} = 1 \quad (4c)$$

The optimum of the scheduling problem is the permutation represented by the solution Y . This problem has n^2 variables and $2n^3 + n(n+1)/2 + |E|$ constraints. This formulation is well-known in the scheduling literature. For details, see, e.g., the result of Chudak and Hochbaum [5].

3 π -MDD

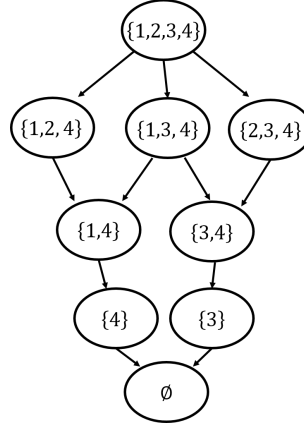
We propose π -MDD, a variant of MDD (Multiple-Valued Decision Diagram[18][19]). An MDD is a data structure representing a set of vectors, while a π -MDD represents a set of permutation vectors.

A π -MDD $D = (V_D, E_D)$ over $N \subset \mathbb{N}$, $|N| = n$ is a DAG with the root node \mathbf{r} whose in-degree is 0 and the terminal node \mathbf{t} whose out-degree is 0, where $V_D \subseteq 2^N$ consists of sets of nodes. Each node of V_D corresponds to a subset of N . In particular, $\mathbf{r} = N$ and $\mathbf{t} = \emptyset$. The structure of a π -MDD has several layers. At the first layer, only the root $\mathbf{r} = N$ exists and i -th layer consists of nodes u with size $|u| = n - i + 1$ ($i = 1, \dots, n$). Edges of a π -MDD appear only between consecutive layers. More precisely, $(u, u') \in E_D$ if and only if $u' \subset u$ and u and u' differ in exactly one element. Each path from the root \mathbf{r} to the terminal \mathbf{t} has length exactly n and each node u in V_D with distance i from \mathbf{t} corresponds to a set of size i , that is $|u| = i$. Let P_D be the set of all paths from the root $\mathbf{r} = N$ to the terminal $\mathbf{t} = \emptyset$. For convenience, we sometimes regard a path in P_D as a sequence of vertices along the directed path. That is, $P_D \subset (2^N)^{n+1}$. Figure 1 illustrates a π -MDD. Given a π -MDD D , a path $\mathbf{p} = (p_1, \dots, p_{n+1}) \in P_D$ defines a permutation $\pi_{\mathbf{p}}$ as follows:

$$\pi_{\mathbf{p}} = (\pi_{\mathbf{p},1}, \dots, \pi_{\mathbf{p},n}) \text{ s.t. } \pi_{\mathbf{p},i} \in p_{n-i+1} \setminus p_{n-i+2} \quad (i = 1, \dots, n).$$

Note that the set $p_{n-i+1} \setminus p_{n-i+2}$ is a singleton and thus the definition is well-defined. For example, in Figure 1, for the path $\mathbf{p} = (\{1, 2, 3, 4\}, \{2, 3, 4\}, \{3, 4\}, \{3\}, \emptyset)$, the corresponding permutation $\pi_{\mathbf{p}}$ is $(3, 4, 2, 1)$. Similarly, two sets of permutations associated with a π -MDD D over $N \subset \mathbb{N}$ are defined as follows:

$$\Pi(D) = \{\pi_{\mathbf{p}} \mid \mathbf{p} \in P_D\}, \text{ and } \Pi^{-1}(D) = \{\pi^{-1} \mid \pi \in \Pi(D)\}.$$



■ **Figure 1** An illustration of a π -MDD D .

For the π -MDD in Figure 1, $\Pi(D) = \{(4, 1, 2, 3), (4, 1, 3, 2), (3, 4, 1, 2), (3, 4, 2, 1)\}$, and $\Pi^{-1}(D) = \{(2, 3, 4, 1), (2, 4, 3, 1), (3, 4, 1, 2), (4, 3, 1, 2)\}$. $S_{\pi}^{-1}(D) S_{\pi}(D)$, respectively.

Now we propose a method to solve the scheduling problem using π -MDDs. The method consists of two parts.

1. Given a DAG $G = ([n], E)$ which represents precedence constraints, construct a π -MDD D such that $\Pi(D) = S_{[n]}^{-1}(G)$. That is, the π -MDD D represents inverses of permutations satisfying the constraints.
2. Given the π -MDD D over $[n]$ and a weight vector $\mathbf{w} \in \mathbb{R}^n$, solve

$$\boldsymbol{\pi} = \arg \min_{\boldsymbol{\pi} \in S_{[n]}(G)} \boldsymbol{\pi} \cdot \mathbf{w}.$$

3.1 Construction of a π -MDD

In this subsection, we consider the following problem:

Input : DAG $G = ([n], E)$

Output : π -MDD D s.t. $\Pi(D) = S_{[n]}^{-1}(G)$.

Let $G(V')$ denote the subgraph of G induced by the vertex subset V' , that is, $G(V') = (V', E')$, where $E' = \{(v, v') \in V' \mid (v, v') \in E\}$. Also, let $E(V') = \{(v, v') \in V' \mid (v, v') \in E\}$. First, we describe the algorithm Make π -MDD in Algorithm 1.

The algorithm Make π -MDD recursively constructs a π -MDD from the root node $\mathbf{r} = [n]$ to the terminal node $\mathbf{t} = \emptyset$. For any $\boldsymbol{\pi} \in S_V$ and an integer q , we denote $\boldsymbol{\pi}q$ as $\boldsymbol{\pi}q = (\pi_1, \dots, \pi_{|V|}, q)$. For any set $Y \subseteq \mathbb{R}^n$ and any real number $y \in \mathbb{R}$, let $Y \times y = \{(\mathbf{y}, y) \in \mathbb{R}^{n+1} \mid \mathbf{y} \in Y\}$. Then we prove an important property of $S_V^{-1}(G)$.

► **Lemma 1.** For a DAG $G = (V, E)$ and $Q = \{q \in V \mid d_q^+ = 0\}$,

$$S_V^{-1}(G) = \bigcup_{q \in Q} S_{V \setminus \{q\}}^{-1}(G(V \setminus \{q\})) \times q. \quad (5)$$

Algorithm 1 Make π -MDD.

Require: DAG $G = (V, E)$, where $V \subseteq [n]$

- 1: **if** $V = \emptyset$ **then**
- 2: **return** node \emptyset
- 3: **else if** have never memorized the π -MDD D_G for G **then**
- 4: π -MDD $D \leftarrow (V_D, E_D)$ with $V_D = \{V\}$ and $E_D = \emptyset$.
- 5: **for** each $v \in V$ whose out-degree is 0 in $G = (V, E)$ **do**
- 6: $V' \leftarrow V \setminus \{v\}$
- 7: π -MDD $D' \leftarrow \text{Make}\pi\text{-MDD}(G(V'))$ // root node is V'
- 8: $D \leftarrow D$ with D' and edge (V, V')
- 9: **end for**
- 10: memorize D as D_G
- 11: **end if**
- 12: **return** D_G

Proof. For any fixed $q \in Q$,

$$\begin{aligned}
& \pi q \in S_{V \setminus \{q\}}^{-1}(G(V \setminus \{q\})) \times q \\
\Leftrightarrow & \pi q \in S_V, \forall (v, v') \in E(V \setminus \{q\}), \pi_v^{-1} \leq \pi_{v'}^{-1} \\
& \quad \text{(by definition of } S_V^{-1}(G) \text{ and its property)} \\
\Leftrightarrow & \pi' = \pi q \in S_V, \forall (v, v') \in E(V \setminus \{q\}) \cup \{(i, q) \in V^2 \mid i \in V \setminus \{q\}\}, \pi_v'^{-1} \leq \pi_v^{-1} \\
& \quad \text{(since } \pi_{|V|} = q \Leftrightarrow \pi_q'^{-1} = |V| \text{)} \\
\Rightarrow & \pi' = \pi q \in S_V, \forall (v, v') \in V, \pi_v'^{-1} \leq \pi_v^{-1} = S_V^{-1}(G) \\
& \quad \text{(since } E \subseteq E(V \setminus \{q\}) \cup \{(i, q) \in V^2 \mid i \in V \setminus \{q\}\}\text{),}
\end{aligned}$$

which implies that $\cup_{q \in Q} S_{V \setminus \{q\}}^{-1}(G(V \setminus \{q\})) \times q \subseteq S_V^{-1}(G)$.

For the opposite direction, let π be any member of $S_V^{-1}(G)$. Then, by definition, for any $(v, v') \in E$, it holds that $\pi_v^{-1} \leq \pi_{v'}^{-1}$. Let $q = \pi_{|V|}$. Then, we have $\pi_q^{-1} = |V| > \pi_v^{-1}$ for any $v \in V \setminus \{q\}$. Therefore, there is no out-going edge from q (if exists, it implies a contradiction) and thus $q \in Q$. Together with the fact that $\pi \in S_{V \setminus \{q\}}^{-1}(G(V \setminus \{q\})) \times q$, the opposite direction also holds. \blacktriangleleft

The following lemma holds for Make π -MDD. Now we describe an important relationship between the output of Make π -MDD and the input DAG $G = (V, E)$. Roughly speaking, the π -MDD made by the algorithm represents a set of inverses of permutations satisfying the precedence constraints.

► **Lemma 2.** *Make π -MDD constructs a π -MDD D such that $S_\pi(D) = S_V^{-1}(G)$.*

Proof. The proof is done by induction on the size $k = |V|$ in the input DAG $G = (V, E)$. (i) For $|V|=0$, Make π -MDD outputs the π -MDD $D = (V_D, E_D)$ with $V_D = \{\emptyset\}$ and $E = \emptyset$. Thus the statement is true. (ii) For $|V| = k$, assume that the statement is true. Let $D_{G(V')}$ be the π -MDD made by Make π -MDD($G(V')$). Then,

$$\begin{aligned}
\Pi(D) &= \bigcup_{q \in Q} (\Pi(D_{G(V \setminus \{q\})}) \times q) \\
&= \bigcup_{q \in Q} (S_{V \setminus \{q\}}^{-1}(G(V \setminus \{q\})) \times q) \text{ (by the inductive assumption)} \\
&= \bigcup_{q \in Q} (\{\boldsymbol{\pi} \in S_{V \setminus \{q\}} \mid \forall (v, v') \in E(V \setminus \{q\}) \ \pi_v^{-1} < \pi_{v'}^{-1}\} \times q) \\
&= S_V^{-1}(G) \text{ (by Lemma 1),} \tag{6}
\end{aligned}$$

which completes the inductive proof. \blacktriangleleft

► **Corollary 3.** For the output D of $\text{Make}\pi\text{-MDD}(G)$ with $G = ([n], E)$, $\Pi^{-1}(D) = S_{[n]}(G)$.

The size of the π -MDD can be bounded based on the result of Inoue and Minato [13].

► **Lemma 4.** For a DAG $G = (V, E)$, let $h(G)$ be the width of G . Then, the size $|D|$ of π -MDD D obtained from $\text{Make}\pi\text{-MDD}(G = (V, E))$ is

$$|D| = O(h(G)(n/h(G) + 1)^{h(G)}).$$

Proof. Let $IS(G)$ be the set of DAGs which can be constructed by updating $G = G(V \setminus \{v \in V \mid d^+(v) = 0\})$ recursively. The total number of recursions in $\text{Make}\pi\text{-MDD}$ is at most $|IS(G)|$. It is known that the size $|IS(G)|$ is at most $(n/h(G) + 1)^{h(G)}$ [13]. Since any pair of elements in the set $Q = \{v \in V' \mid d_v^+ = 0\}$ are not reachable to each other, $|Q| \leq h(G)$. Therefore, at each recursion, at most $h(G)$ edges are added, and thus the total edges in the π -MDD made by the algorithm is $O(h(G)(n/h(G) + 1)^{h(G)})$. \blacktriangleleft

3.2 Optimization over a π -MDD

We describe how to find an optimal solution of the scheduling problem (1) using a π -MDD. More precisely, we deal with the following optimization problem.

$$\begin{aligned}
\text{Input : } & \pi\text{-MDD } D = (V_D, E_D) \text{ s.t. } \Pi(D) = S_{[n]}^{-1}(G) \text{ for some DAG } G, \text{ and } \boldsymbol{w} \in \mathbb{R}^n \\
\text{Output : } & \boldsymbol{\pi} = \arg \min_{\boldsymbol{\pi} \in S_{[n]}(G)} \boldsymbol{\pi} \cdot \boldsymbol{w}
\end{aligned}$$

We solve this problem by reducing it to the shortest path problem over the π -MDD D . The reduction is as follows: For each edge $(u, u') \in E_D$, we set the cost $L_{(u, u')}$ as

$$L_{(u, u')} = |u|w_{u \setminus u'}.$$

The cost $L_{\boldsymbol{p}}$ of each path $\boldsymbol{p} \in P_D$ is defined as $L_{\boldsymbol{p}} = \sum_{i \in [n]} L_{(p_i, p_{i+1})}$. Then we consider the shortest path problem over D from the root $\boldsymbol{r} = [n]$ to the terminal $\boldsymbol{t} = \emptyset$ with cost $L_{(v, v')}$ for each edge $(v, v') \in E_D$. This problem can be solved in time $O(|D|)$. We prove the following relationship between the cost of each path $\boldsymbol{p} \in P_D$ and its corresponding permutation.

► **Lemma 5.** For any $\boldsymbol{p} \in P(D)$,

$$L_{\boldsymbol{p}} = \boldsymbol{\pi}_{\boldsymbol{p}}^{-1} \cdot \boldsymbol{w}.$$

Proof.

$$\begin{aligned}
L_{\mathbf{p}} &= \sum_{i \in [n]} L_{(p_i, p_{i+1})} \\
&= |p_1|w_{p_1 \setminus p_2} + \cdots + |p_n|w_{p_n \setminus p_{n+1}} \\
&= nw_{p_1 \setminus p_2} + \cdots + w_{p_n \setminus p_{n+1}} \\
&= nw_{\pi_{\mathbf{p}, n}} + \cdots + w_{\pi_{\mathbf{p}, 1}} \\
&= \sum_{i \in [n]} iw_{\pi_{\mathbf{p}, i}} \\
&= \sum_{i \in [n]} i \sum_{j \in [n]} \mathbb{1}[j = \pi_{\mathbf{p}, i}]w_j \\
&= \sum_{i \in [n]} i \sum_{j \in [n]} \mathbb{1}[\pi_{\mathbf{p}, j}^{-1} = i]w_j \\
&= \sum_{j \in [n]} \sum_{i \in [n]} \mathbb{1}[\pi_{\mathbf{p}, j}^{-1} = i]iw_j \\
&= \sum_{j \in [n]} \pi_{\mathbf{p}, j}^{-1} w_j \\
&= \pi_{\mathbf{p}}^{-1} \cdot \mathbf{w}
\end{aligned}$$

By combining Lemma 2, 4, 5, we obtain the main result.

► **Theorem 6.** *There is an algorithm that, given a DAG $G = ([n], E)$ as precedence constraints, computes a solution of problem (1) in time $O(h(G)(n/h(G) + 1)^{h(G)})$.*

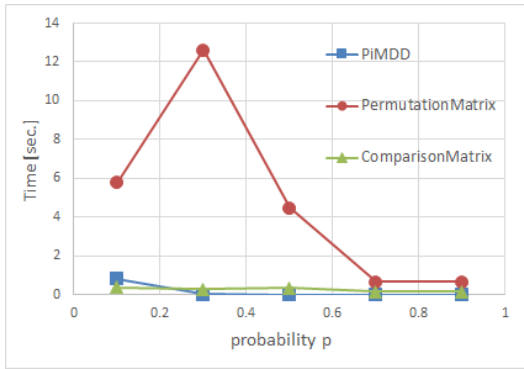
Proof. By Lemma 2, given a DAG $G = ([n], E)$, Make π -MDD constructs a π -DD D such that $\Pi(D) = S_{[n]}^{-1}(G)$. By Lemma 5, we can compute the linear optimization problem over $\Pi^{-1}(D) = S_{[n]}(G)$. By Lemma 4, both constructing a π -MDD and solving the shortest path problem take time $O(h(G)(n/h(G) + 1)^{h(G)})$. ◀

4 Experimental Results

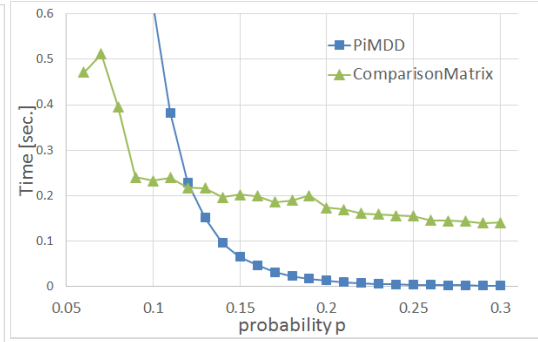
In this section, we show experimental results on artificial data. We construct the artificial data sets of the scheduling problem with precedence constraints by generating DAGs and weight vectors randomly. More precisely, given $V = [n]$, for each $(v_i, v_j) \in V \times V (v_i < v_j)$, we assign the edge $(v_i, v_j) \in E$ with probability p ($0 < p < 1$). Note that, because of the constraint that $v_i < v_j$, the resulting random graph is a DAG. We generate a random weight vector by generating each w_i according to the uniform distribution over $[0, 1]$ for $i \in [n]$. We use the parameters $n = 25$, and $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. We compare the proposed methods with π -MDD, and integer programming (IP) with permutation matrices and comparison matrices, respectively. We implemented these methods in C++ and used the Gurobi optimizer 6.5.0 to solve integer programs. We run them in a machine with Intel(R) Xeon(R) CPU X5560 2.80GHz and 198GB memory.

Figure 2 shows the computation times of each method for different choices of p . The shown results are obtained by averaging over 500 random instances for each fixed choice of p .

The proposed method is fastest among others when $p > 0.15$, i.e., precedence constraints are not sparse. (Figure3 shows the detailed results). In particular, for $p \geq 0.2$, the speed up by the proposed method is 10 times w.r.t. the IP with comparison matrices and more than 20 times w.r.t. the IP with permutation matrices. Also, as can be seen in Figure 2, computation



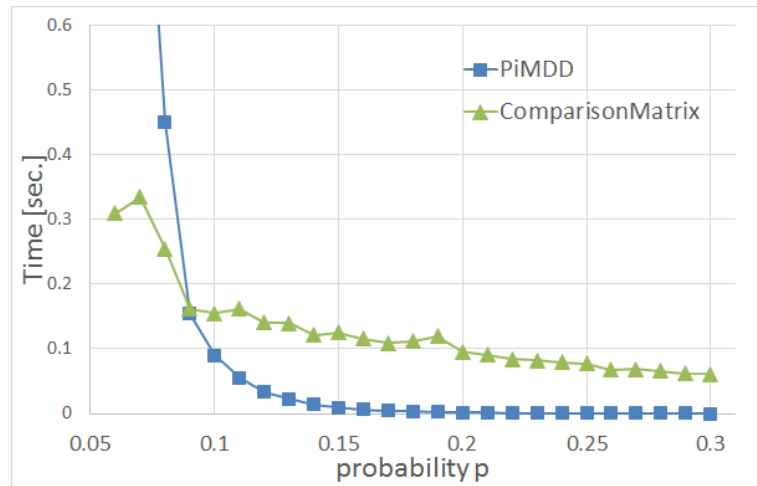
■ **Figure 2** Average running time for $n = 25$.



■ **Figure 3** Average running time for $n = 25$ in details.

■ **Table 1** Average number of constraints for $n = 25$.

	$k = 0.1$	$k = 0.3$	$k = 0.5$	$k = 0.7$	$k = 0.9$
permutation matrix	799.3	2304.8	3801.9	5294.0	6803.4
comparison matrix	31580.0	31640.2	31700.1	31759.8	31820.1



■ **Figure 4** Average computation time for optimization for $n = 25$.

time of the IP with permutation matrices is much larger than that of the IP with comparison matrices by 10 times for $p \leq 0.5$. In contrast, as in Table 1, the number of constraints used in IP with comparison matrices is much larger. So, the number of constraints does not seem to affect the computation time.

In general, if the DAG $G = (V, E)$ is sparse, the width of G tends to be larger. In fact, for $p = 0.1$, the average width of the random graph is 11.808, while for $p = 0.5$, the average width is 3.596. Note that the worst case time complexity bound $O(h(n/h + 1)^h)$ for constructing a π -MDD depends on the width h .

Next, for sparse precedence constraints, we compare performances of our π -MDD based method and the IP with comparison matrices. For $n = 25$, and $p \in \{0.06, 0.07, \dots, 0.30\}$, we generate 500 random DAGs for precedence constraints, and run these algorithms for each random instance. Figure3 shows averaged results. The average running time of the proposed method becomes smaller than the IP-based method for $p \geq 0.13$.

Now we compare computation times of these methods for optimization only. Here we separate computation time into preprocessing time and time for optimization. For our π -MDD based method, time for constructing a π -MDD is for preprocessing and solving the shortest path problem over the π -MDD corresponds to the optimization part. For IP with comparison matrices, we regard time for constructing a problem instance (e.g., adding constraints) as preprocessing time.

Figure 4 shows computation times of the proposed method and the IP with comparison matrices for optimization only. For optimization, the proposed method is faster than the IP for $p \geq 0.09$. This result indicates that the proposed method is better suited when we solve scheduling problems with the same precedence constraints and different weight vectors.

References

- 1 Sheldon B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.
- 2 Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. On the Approximability of Single-Machine Scheduling with Precedence Constraints Christoph Ambühl. *Mathematics of Operations Research*, 36(4):653–669, 2011.
- 3 Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, aug 1986.
- 4 Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.
- 5 Fabian A. Chudak and Dorit. S. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, 25:199–204, 1999.
- 6 André A Ciré and Willem Jan van Hoeve. MDD Propagation for Disjunctive Scheduling. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 2012.
- 7 André A Ciré and Willem-Jan van Hoeve. Multivalued Decision Diagrams for Sequencing Problems. *Operations Research*, 61(6):1411–1428, 2013.
- 8 José R. Correa and Andreas S. Schulz. Single-Machine Scheduling with Precedence Constraints. *Mathematics of Operations Research*, 30(4):1005–1021, 2005.
- 9 Takahiro Fujita, Kohei Hatano, Shuji Kijima, and Eiji Takimoto. Online Linear Optimization for Job Scheduling under Precedence Constraints. In *Proceedings of 26th International Conference on Algorithmic Learning Theory(ALT 2015)*, volume 6331 of *LNCS*, pages 345–359, 2015.
- 10 Tarik Hadzic, John N Hooker, Barry O’Sullivan, and Peter Tiedemann. Approximate Compilation of Constraints into Multivalued Decision Diagrams. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP 2008)*, *LNCS* 5202, pages 448–462, 2008.
- 11 Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.
- 12 Takeru Inoue, Keiji Takano, Takayuki Watanabe, Jun Kawahara, Ryo Yoshinaka, Akihiro Kishimoto, Koji Tsuda, Shin-ichi Minato, and Yasuhiro Hayashi. Distribution Loss Minimization With Guaranteed Error Bound. *IEEE Transactions on Smart Grid*, 5(1):102–111, 2014.

- 13 Yuma Inoue and Shin-ichi Minato. An Efficient Method for Indexing All Topological Orders of a Directed Graph. In *Proceedings of the 25th international Symposium on Algorithms and Computation (ISAAC'14)*, pages 103–114, 2014.
- 14 Donald E. Knuth. *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 2011.
- 15 Eugene L. Lawler. On Sequencing jobs to minimize weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics* 2, 2:75–90, 1978.
- 16 Jan K. Lenstra and Alexander H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.
- 17 François Margot, Maurice Queyranne, and Yaoguang Wang. Decompositions, Network Flows, and a Precedence Constrained Single-Machine Scheduling Problem. *Operations Research*, 51(6):981–992, 2003.
- 18 Michael D. Miller. Multiple-Valued Logic Design Tools. In *Proceedings of the 23rd IEEE International Symposium on Multiple-Valued Logic (ISMVL'93)*, pages 2–11, 1993.
- 19 Michael D. Miller and Rolf Drechsler. Implementing a Multiple-Valued Decision Diagram Package. In *Proceedings of the 28th IEEE International Symposium on Multiple-Valued Logic (ISMVL'98)*, pages 52–57, 1998.
- 20 Shin-ichi Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proceedings of the 30th international Design Automation Conference (DAC'93)*, pages 272–277, 1993.
- 21 Shin-ichi Minato. Zero-suppressed BDDs and their applications. *International Journal on Software Tools for Technology Transfer*, 3(2):156–170, 2001.
- 22 Shin-ichi Minato. Efficient Database Analysis Using VSOP Calculator Based on Zero-Suppressed BDDs. In *New Frontiers in Artificial Intelligence, Joint JSAI 2005 Workshop Post-Proceedings*, pages 169–181, 2005.
- 23 Shin-ichi Minato. π DD: A New Decision Diagram for Efficient Problem Solving in Permutation Space. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*, pages 90–104, 2011.
- 24 Andreas S. Schulz. Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-Based Heuristics and Lower Bounds. In *Proceedings of the 5th Conference on Integer Programming and Combinatorial Optimization (IPCO1996)*, pages 301–315, 1996.