Privacy-and-Utility-Aware Publishing of Schedules

Maike Basmer[1], **Stephan A. Fahrenkrog-Petersen**[1,2], **Ali Kaan Tutak**[1], **Arik Senderovich**[3], **Matthias Weidlich**[1]

[1]Humboldt-Universität zu Berlin, Unter den Linden 6, 10117 Berlin, Germany
[2]Weizenbaum Institute, Hardenbergstraße 32, 10623 Berlin, Germany
[3]York University, 4700 Keele St, Toronto, ON M3J 1P3, Canada
{basmermo,fahrenks,tutakali,matthias.weidlich}@hu-berlin.de
sariks@yorku.ca

Abstract

Scheduling is adopted in various domains to assign jobs to resources, such that an objective is optimized. While schedules enable the analysis of the underlying system, publishing them also incurs a privacy risk. Recently, privacy attacks on schedules have been proposed, which may reveal sensitive information on the jobs by solving an inverse scheduling problem. In this work, we study the protection against such attacks. We formulate the problem of privacy-and-utility preservation of schedules, which bounds both, the privacy leakage and the loss in the utility of the schedule due to obfuscation. We address the problem based on a set of perturbation functions for schedules, study their instantiations for standard scheduling problems, and implement privacy-and-utility-aware publishing of a schedule using constraint programming. Experiments with synthetic and real-world schedules demonstrate the feasibility, robustness, and effectiveness of our mechanism.

1 Introduction

By assigning resources to tasks while incorporating a certain objective, schedules enable the efficient utilization of scarce resources (Pinedo 2016). Consequently, they are employed in a variety of domains, such as healthcare, production, or service industries. Reflecting the underlying processes and decisions, they lend themselves to analysis and data mining (Harding et al. 2005; Li and Olafsson 2005), which possibly entails sharing them with third parties.

Publishing schedules, however, holds the risk of exposing sensitive information. In particular, an adversary may link knowledge of the environment constraining the scheduling outcome with the published schedule to infer private properties of the jobs (Fahrenkrog-Petersen et al. 2023). Due to legal regulations, economic implications, and ethical considerations, it is essential to counteract the threat of privacy leakage from schedules. At the same time, the publishing of schedules shall still be facilitated, for operational management and the improvement of the underlying system.

As a real-world example from healthcare, taken up again in our evaluation, consider the scheduling of infusions in a clinic, see Fig. 1. The schedules are guided by priorities that emerge from factors such as the patients' medical conditions, and optimize the total weighted completion time of treatments. To optimize the operational processes, the clinic shares schedules with an external consulting firm, which analyzes the impact of operational decisions on performance indicators (e.g., wait time, resource utilization). Although the priorities that encode sensitive information on patients are not published with the schedule, they may potentially be reconstructed through an inverse scheduling attack.

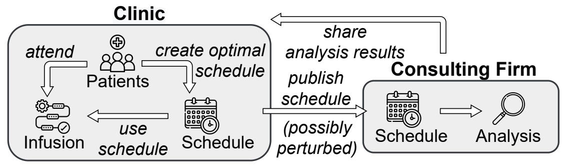


Figure 1: Healthcare example for schedule publishing.

We set out to address this challenge by an approach for the release of schedules that preserves the undisclosed characteristics of jobs, while maintaining the utility of the schedule in terms of the properties that are deemed relevant for some analysis. Based on a formal characterization of the problem, we propose perturbation functions for schedules, explore their instantiation, and operationalize our approach to publishing of schedules using constraint programming. Specifically, we make the following contributions:

- We define the problem of Privacy-and-Utility Preservation (PUP) when releasing schedules. Given a schedule, this problem points to the existence of a transformed schedule that satisfies a privacy loss threshold, in addition to a utility loss threshold.
- We instantiate the PUP with a set of perturbation functions for schedules; and study them for two well-established scheduling problems for parallel machines: the minimization of (i) the total weighted completion times and (ii) the makespan for jobs with release dates.
- To solve the instantiated PUP, we provide a formulation using constraint programming. It exploits the neighborhood of a schedule as induced by the perturbations.

We experimentally demonstrate the feasibility of our perturbation functions: We solve the PUP for 58-80% of synthetic schedules, and 20-83% of real-world schedules from a medical clinic. We also highlight the effectiveness of the PUP: Important analytical properties of schedules are preserved, even when they are not incorporated in the utility function.

2 Related Work

2.1 Privacy Models

Various models for privacy in data publishing have been proposed (Fung et al. 2010). Some limit the success rate of linkage attacks by hiding individuals in groups (Sweeney 2002; Samarati 2001; Machanavajjhala et al. 2007). In contrast, differential privacy defines limits on the impact that one individual has on the published data (Dwork 2006). While differential privacy represents a privacy guarantee for release mechanisms, often operating on aggregations and achieved by adding random noise, we explore the protection of a public schedule with individual jobs that shall be preserved. To achieve a given privacy notion, generalization, suppression, and perturbation are common anonymization operators (DeSmet and Cook 2021). Privacy concerns have also found their way into related domains dealing with sequential or graph data, such as process mining (Mannhardt et al. 2019) or social network analysis (Majeed and Lee 2021). Graph modifications like deleting/adding edges represent structural changes to protect sensitive information (Casas-Roma, Herrera-Joancomartí, and Torra 2016; Liu and Terzi 2008) similar to ours, yet they target different attacks.

2.2 Scheduling and Inverse Scheduling

Scheduling concerns the sequencing of jobs and the allocation of resources to them while optimizing for some objectives (Pinedo 2016). In contrast, inverse scheduling problems (ISP) adjust job parameters, to the least extent possible, such that a given sequence of jobs optimizes a predetermined objective (Brucker and Shakhlevich 2009). Koulamas (2005) also include a perturbation cost for each parameter to prioritize changes. In addition to altering the job parameters, Mou et al. (2015) allow for an adaptation of the job sequence, but penalize the deviation. Particularly, minimizing the deviations from the original schedule is one of multiple objectives of the proposed ISP. To solve it, the authors suggest a hybrid evolutionary search that applies genetic operators targeting the processing times and the scheduling sequence. Genetic approaches have also been used in forward scheduling, that similarly employ mutation operators like insertion mutation, reciprocal exchange mutation, or shift mutation to alter the sequence of jobs (Min and Cheng 1999; Cheng, Gen, and Tsujimura 1999). Likewise, the privacy protection framework proposed in this paper also alters a given schedule along with the job parameters to satisfy an objective while ensuring that the resulting schedule diverges from the original one only to a predetermined extent.

3 Background

3.1 Parallel Machine Scheduling

We use $[k]$ to denote the set $\{1, ..., k\}$ for $k \in \mathbb{N}$. Let $\mathcal{S} = \mathbb{R}^n \times [m]^n$ denote the universe of parallel schedules with $n \in \mathbb{N}$ jobs and $m \in \mathbb{N}$ machines. A schedule $s \in \mathcal{S}$ is a tuple of two n-size vectors (π, μ), where π denotes the vector of starting times and μ represents the allocation of machines, i.e., π_j is the starting time of job j and μ_j is the machine to which the job j is assigned ($j = 1, \ldots, n$).

Adopting common notations, a parallel-machine scheduling problem is a quintuple $\Pi = (J, R, X, C(X), \phi)$ where:

- $J = [n]$ refers to the n jobs up for processing,
- $R = [m]$ refers to the m machines (aka resources),
- X is an $n \times q$ matrix describing the job features, i.e., $x_{j,k}$ denotes the k-th feature of the j-th job,
- $C(X) \subseteq 2^{\mathcal{S}}$ encompasses the constraints, parametrized by job features, that define a feasible schedule, and,
- ϕ denotes the objective function that is to be optimized.

Let $\mathcal{S}_\Pi \subseteq \mathcal{S}$ refer to the set of feasible schedules for the scheduling problem Π with respect to the constraints $C(X)$, i.e., $\mathcal{S}_\Pi = \bigcap_{c \in C(X)} c$. Also, by $\mathcal{X}_k$, we denote the feature domain of feature $k \in [q]$. In this work, we assume that ϕ is a regular objective function, i.e., it is non-decreasing in job completion times, which leads to feasible schedules without unnecessary delays and idle times (Pinedo 2016).

Furthermore, we assume that some features of jobs are published, whereas others remain private. Hence, there are two submatrices X_{priv} and X_{pub} of X, with dimensions $n \times q_{priv}$ ($q_{priv} < q$) and $n \times q_{pub}$ ($q_{pub} = q - q_{priv}$), s.t. $X = [X_{priv} X_{pub}]$. Let $\mathcal{X}_{priv}$ and $\mathcal{X}_{pub}$ denote the domains of valid private and public feature matrices, i.e., $\forall\, X \in \mathcal{X}_{priv}, j \in J, k \in [q_{priv}] : (x)_{jk} \in \mathcal{X}_k$ (analogously for $\mathcal{X}_{pub}$). Further, let $\Pi(X_{priv})$ be a parametrized scheduling problem derived from the original problem Π, where only the private features X_{priv} and the constraints $C(X_{priv})$ depending on them vary, while the remaining parameters are fixed. Then, given X_{priv} and $\Pi(X_{priv})$, a scheduling function f produces a schedule, while we write $f(X)$ instead of $f(X, \Pi(X_{priv}))$, when the scheduling problem is clear from the context. A scheduling function may yield an optimal schedule (choosing one, if there are multiple candidates) or approximate it for efficiency reasons, as discussed later.

3.2 Privacy Attack on Schedules

The features X_{priv} may contain sensitive information such as priorities in the healthcare example. Even if they are not published, they influenced the creation of a schedule, so that an adversary may try to infer them by an *Inverse Scheduling Attack* (ISA) (Fahrenkrog-Petersen et al. 2023). Specifically, given a schedule $s \in \mathcal{S}_\Pi$, an adversary relies on partial knowledge on the scheduling problem that was solved to create s, namely the jobs J and machines R, the public features X_{pub} like the job durations, the constraints $C(X_{pub})$ depending on X_{pub}, and the objective function ϕ. While the private features X_{priv} are not available, the adversary is assumed to be familiar with their domain, $\mathcal{X}_{priv}$. The adversary expects that the published schedule s is an optimal or approximated solution to $\Pi(X_{priv})$, i.e., they make an assumption on the scheduling function f. Then, they aim at finding $X' \in \mathcal{X}_{priv}$, such that $f(X') = s$.

The result of the ISA is a set $\mathbb{Y}(s) = \{X' \in \mathcal{X}_{priv} \mid f(X') = s\}$ containing candidate private feature values. Then, a distance-based privacy loss ξ_{Dist} measures the incurred information leakage, as follows. Let $Y^l_{j,k}$ be the k-th feature of the j-th job in the l-th guess in the candidate set $\mathbb{Y}$. The guessed value $Y^l_{j,k}$ is compared to the actual value $x_{j,k}$ using a distance measure, denoted

$d(x_{j,k}, Y^l_{j,k})$, depending on the feature domain: the absolute difference for numeric domains, and the discrete metric for discrete domains. With $\hat{p}_Y(x)$ as the frequency of value x in the multi-set $Y_{j,k} = [Y^1_{j,k}, \dots, Y^{|\mathbb{Y}|}_{j,k}]$ induced by the $|\mathbb{Y}|$ guesses, the plain loss is derived by Lebesgue integration as $D(x_{j,k}, Y_{j,k}) = \int_{x \in \mathcal{X}_k} d(x_{j,k}, x)\hat{p}_Y(x)\,\mathrm{d}x$. Then, normalization yields $D_N(x_{j,k}, Y_{j,k}) = {D(x_{j,k}, Y_{j,k})}/{D(x_{j,k}, \mathcal{X}_k)}$ with $D(x_{j,k}, \mathcal{X}_k) = {\sum_{x \in \mathcal{X}_k} d(x_{j,k}, x)}/{|\mathcal{X}_k|}$ as the normalization factor. The privacy loss is the max value over all jobs and features, capturing the worst case, i.e., $\xi_{dist}(s) = \max_{j \in J, k \in [q_{priv}]} (1 - D_N(x_{j,k}))$.

As such, if $\mathbb{Y}(s)$ is empty, then the loss is zero; with one or more elements, the distances are computed accordingly.

3.3 Attacks on Specific Scheduling Problems

In the remainder, we consider two well-established scheduling problems for n jobs over m identical parallel machines: $P||\sum_{j \in J} w_j C_j$, the minimization of total weighted completion times (TWCT); and $P|r_j|C_{max}$, the minimization of makespan under release dates (MAKE). Here, available machines start available (released) jobs without delay.

In TWCT, each job $j \in J$ has a weight $w_j \in \mathcal{X}_w$ and processing time $p_j \in \mathcal{X}_p$ of domains $\mathcal{X}_w$ and $\mathcal{X}_p$, respectively. Thus, X is a two-dimensional matrix. The feasible schedules $\mathcal{S}_{twct}$ are determined by two constraints: each job is scheduled once, and jobs do not overlap on a given machine. With C_j as the completion time of job j, an optimal TWCT schedule minimizes $\sum_{j \in J} w_j C_j$. The single-machine version of the problem, $1||\sum w_j C_j$, is efficiently solvable using the Weighted Shortest Processing Time (WSPT) dispatching rule (Pinedo 2016). It sorts the jobs in non-decreasing order of their ratios ${w_j}/{p_j}$. For more than one machine, the problem is NP-hard (Bruno, Coffman, and Sethi 1974). Yet, in this case, sorting the jobs by WSPT and assigning them in the order in which machines become available yields an approximation, with an objective value within $(\sqrt{2} + 1)/2$ of the optimum (Kawaguchi and Kyan 1986).

In MAKE, each job $j \in J$ has a release date $r_j \in \mathcal{X}_r$ (of domain $\mathcal{X}_r$) and processing time $p_j \in \mathcal{X}_p$, rendering X a two-dimensional matrix. The feasible schedules $\mathcal{S}_{make}$ are determined by the two above constraints, and those derived from the release dates, i.e., the starting time π_j of job j must not be smaller than its release date r_j, i.e., $\pi_j \geq r_j$. An optimal schedule minimizes the makespan, i.e., the maximal completion time $\max_{j \in J} C_j$. While the problem is NP-hard (Garey and Johnson 1978), it can be approximated with a dispatching rule: Sorting jobs in non-increasing order of their durations and assigning them in the order in which machines become available or jobs are released yields a $3/2$-approximation (Chen and Vestjens 1997).

Turning to the ISA, processing times p are publicly available (published or derived from the schedule for TWCT). An adversary aims to find the weight vector w (for TWCT) or the release date vector r (for MAKE). In our healthcare example, these features may denote patients' priorities and their availability for treatments following preparatory steps, both originating from sensitive medical diagnoses.

The attack on a schedule is framed as a constraint satisfaction problem (CSP). The inputs originate from the public schedule and the domain knowledge, the decision variable represents possible assignments to the private features, and the scheduling function f determines the constraints that limit the space of variable assignments. Thus, the CSP for a given schedule $s = (\pi, \mu)$ can be defined as follows:

Inputs:	J	set of jobs
	π	start times in s
	p	processing times
	$\mathcal{X}_{priv}$	priv. feature domains
Decision Var.:	$X' \in \mathcal{X}_{priv}$	private features
Constraints:	$f(X') = s$	
Output:	$\mathbb{Y}(s) \in \mathcal{P}(\mathcal{X}_{priv})$	cand. priv. features

For TWCT, for instance, the CSP incorporates the weight domain $\mathcal{X}_{priv} = \mathcal{X}^n_w$ as input, the weight vector $w \in \mathcal{X}^n_w$ as decision variable, and a set of candidate weight vectors $\mathbb{Y}(s) = W \subseteq \mathcal{X}^n_w$ as output.

As mentioned, defining f to construct an optimal schedule is NP-hard for TWCT and MAKE. To implement the attack efficiently, in practice, an adversary adapts the above CSP by approximating an optimal schedule. In that case, the constraint $f(X') = s$ is represented by a dispatching rule for the considered scheduling problem. For TWCT, for instance, the constraint represents the WSPT rule:

$$\frac{w_j}{p_j} \geq \frac{w_{j'}}{p_{j'}} \quad \forall\, j \in J\, \forall\, j' \in succ(s, j)$$

where $succ(s, j) = \{k \in J \mid \pi_j < \pi_k \wedge \nexists\, j' \in J : \pi_j < \pi_{j'} < \pi_k\}$ are the direct successors of a job j in schedule s.

4 Privacy-and-Utility Preservation Problem

In the light of the above privacy threat, publishing of a schedule for operational analysis shall be done in a way that protects private information about jobs. With an adversary trying to infer the 'true' private feature matrix, the amount of information disclosed is tied to the distance of the actual private values to the feature values in the ISA solution set $\mathbb{Y}(s)$. Hence, one may try to deceive the adversary by distorting the schedule before publishing it.

Yet, when distorting the schedule, the utility of the published schedule for legitimate analyses must be considered. We quantify the potential negative implications of a distortion for any analysis conducted on the published schedule by a utility loss. To this end, let $z : \mathcal{S} \to \mathbb{R}$ be a function that maps a schedule to real values capturing a property deemed relevant for the analysis, such as the average waiting time. We further define $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ as the deviation in utility between two schedules, e.g., the absolute difference between the average waiting times. Applying the distance measure to the optimal schedule and the distorted schedule enables us to quantify the utility loss for the analysis of the system. We call $\chi(s^*, s) = d(z(s), z(s^*))$ the utility loss function.

Using these notions, we define the *Privacy-and-Utility-Preservation Problem (PUP)*.

Problem 1 (PUP). *Given a schedule $s^* \in \mathcal{S}_\Pi$, find a schedule $s \in \mathcal{S}_\Pi$, such that $\xi(s) \leq \epsilon$ and $\chi(s^*, s) \leq \delta$.*

The two thresholds, i.e., the utility loss threshold δ and the maximum privacy loss ϵ, are user-defined. The former delimits the maximal tolerable utility loss, e.g., for downstream analysis, whereas the latter constrains the privacy loss.

Note that the complexity of the PUP depends on three aspects: the complexity of finding a feasible solution to Π', the complexity of computing $\xi(s)$ (which can be hard, as it requires obtaining the set of all solutions $\mathbb{Y}(s)$), and the complexity of computing the distance between $z(s)$ and $z(s^*)$.

5 Perturbation Functions

Since parallel scheduling involves sequencing jobs on the corresponding machines, changing these sequences and machine allocations influences the outcome of an inverse scheduling attack. Thus, perturbing the sequence and allocation of jobs lends itself to the PUP. Distorting job properties may also serve as a perturbation mechanism, as those features influence the inferences the adversary can draw. In this part, we propose a set of perturbation functions that distort published schedules to prevent the success of an ISA.

5.1 Definition of Perturbations

Let Π be a scheduling problem and $s = (\pi, \mu) \in \mathcal{S}_\Pi$ be a schedule with public feature matrix X_{pub} of dimension $n \times q_{pub}$. We propose the following perturbation mechanisms for the protection of published schedules: $\rightleftarrows$ to swap jobs; $\curvearrowright$ to move jobs; and $\uparrow\downarrow$ to perturb public features.

Swap Jobs ($\rightleftarrows$) Swapping jobs is modeled by a function that, given a schedule $s \in \mathcal{S}_\Pi$, changes the position of two jobs $j, j' \in J$, such that j takes the position of j' on the same machine, and vice versa. As a shorthand, we write $\rightleftarrows (j, j')$ to denote the respective swapping of jobs. The resulting schedule is then determined by the *earliest starting time (EST) principle*, i.e., setting the starting times of the jobs assigned to a certain machine as the earliest time points that the jobs may be processed (i.e., as derived from the end time of the preceding job and additional constraints, such as release dates).

Move Jobs ($\curvearrowright$) Moving a job is defined as a function that moves a job $j \in J$ to position $k \in \{1, \dots, l_i + 1\}$ on machine $i \in R$, where l_i is the number of jobs on machine i. By $\curvearrowright (j, i, k)$, we denote such a move. As before, the resulting schedule is derived by EST. Note that in the case of $m = 1$, moving a job to a different position reduces to a sequence of swaps. Otherwise, moving jobs allows us to relocate jobs to different machines, and thus change the number of jobs allocated to a machine.

Perturb Features ($\uparrow\downarrow$) Perturbing features is captured by a function that sets $(x_{pub})_{jk}$ to a new value $x \in \mathcal{X}_k$ for job $j \in J$ and feature $k \in \{q_{pub}, \dots, q\}$. We write $\uparrow\downarrow (q_{pub})$ to denote the change of public features. Again, the schedule is then constructed adopting EST. In contrast to swapping or moving, changing the job features allows for relatively fine-grained changes. In general, only features involved in the schedule generation are sensible targets for perturbation. For example, manipulating processing times impacts any objective function that relies on completion times.

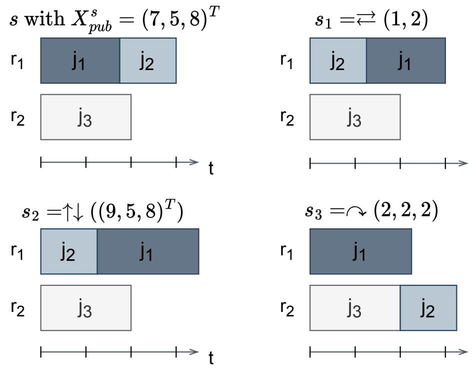

s with $X^s_{pub} = (7,5,8)^T$
$s_1 = \rightleftarrows (1,2)$
$s_2 = \uparrow\downarrow ((9,5,8)^T)$
$s_3 = \curvearrowright (2,2,2)$
r1
r2
j1
j2
j3
t

Figure 2: Application of perturbation functions to schedule $s = ((0,7,0),(1,1,2))$ with $X_{pub} = p = (7,5,8)^T$.

Since EST respects the scheduling constraints, the perturbations guarantee closure. They yield a feasible (not necessarily optimal) schedule when applied to a feasible schedule.

Figure 2 illustrates the application of a sequence of perturbations to an exemplary schedule s with the public features being the processing times of 7 (j_1), 5 (j_2), and 8 (j_3) time units (upper left). First, jobs j_1 and j_2 are swapped (upper right). Then, the first public feature (i.e., the processing time) of the first job (j_1) is perturbed, i.e. set to 9 time units (lower left). Finally, the second job (j_2) is moved to the second position of the second machine (lower right).

5.2 Reachability with Perturbations

Perturbation functions can be used to randomly add noise until no more information is leaked. Due to their nature, they also allow for the systematic exploration of schedule spaces. In particular, we obtain the following reachability properties.

Proposition 1 (Reachability)**.** *Let Π be a scheduling problem with a regular objective function. For all $s, \hat{s} \in \mathcal{S}_\Pi$, s is reachable from $\hat{s}$ using the operators $\{\uparrow\downarrow, \curvearrowright\}$.*

Proof. Assume $\hat{s}$ is given and s is the target schedule. Set a temporary schedule $t := \hat{s}$, and move all jobs in t to machine 1. Then, redistribute all the jobs in t that are not allocated to machine 1 in s to their respective machines and positions, ordered by their positions in s, by applying $\curvearrowright$ in the corresponding order. Reorder the remaining jobs on machine 1 in t by moving the jobs based on the order of jobs on machine 1 in s. Finally, adapt the features in X_{pub} of t by applying $\uparrow\downarrow$ to each job, such that they reflect X_{pub} of s. By definition of $\{\uparrow\downarrow, \curvearrowright\}$, none of the steps violate the constraints of Π nor introduce unnecessary idling. $\square$

Note that the result does not hold if one relaxes the assumption that the objective function, ϕ, is regular. The following result is immediate from Proposition 1.

Corollary 1. *For all $s, \hat{s} \in \mathcal{S}_\Pi$ such that $\Pi \in \{make, twct\}$, s is reachable from $\hat{s}$ using $\{\uparrow\downarrow, \curvearrowright\}$.*

Proof. Both makespan and total weighted completion time are regular objective functions, which implies that $\mathcal{S}_{make}$ and $\mathcal{S}_{twct}$ are special cases of Proposition 1. $\square$

6 Implementation

6.1 PUP

Following the definition of the PUP, we can frame the privacy-and-utility-aware publishing of a schedule as a CSP. For a scheduling problem Π, the latter is defined as:

Inputs:	$s^* \in \mathcal{S}_\Pi$	original schedule
Decision Var.:	$s \in \mathcal{S}_\Pi$	schedule
Constraints:	$\xi(s) \leq \epsilon$ $d(z(s^*), z(s)) \leq \delta$	
Output:	$s \in \mathcal{S}_\Pi$	feasible, privacy-aware, utility-aware schedule

We realize the PUP as a Breadth-First Search. A neighborhood function $h : \mathcal{S}_\Pi \rightarrow 2^{\mathcal{S}_\Pi}$ induces the neighborhood of a given schedule and, thus, defines the search space. Here, the perturbation functions introduced in the previous section serve as neighborhood functions that allow the PUP to systematically explore $\mathcal{S}_\Pi$.

6.2 Neighborhood Functions

Let $s = (\pi, \mu) \in \mathcal{S}_\Pi$ be a schedule with public features X_{pub}. The $\rightleftarrows$-neighborhood of a schedule $s = (\pi, \mu)$ is computed by swapping each job $j \in J$ with every other job.

Similarly, the $\curvearrowright$-neighborhood of s is defined by moving every job $j \in J$ to every valid position k on every machine $i \in R$ within the schedule, where each move defines a separate neighboring schedule.

We obtain the $\uparrow\downarrow$-neighborhood for feature $k \in \{q_{pub}, \ldots, q\}$ by setting $(x_{pub})_{jk}$ to $\max(\{x \in \mathcal{X}_k : x < (x_{pub})_{jk}\})$ to generate one neighboring schedule and to $\min(\{x \in \mathcal{X}_k : x > (x_{pub})_{jk}\})$ to generate another one for each job $j \in J$, assuming that $\mathcal{X}_k$ is discrete. For example, if $\mathcal{X}_k \subseteq \mathbb{N}$ is consecutive, $(x_{pub})_{jk}$ is incremented by 1 to generate one neighboring schedule and decremented by 1 to generate another for each job $j \in J$ in s, unless the resulting $(x_{pub})_{jk}$ is not in $\mathcal{X}_k$ anymore. Combined distortion mechanisms, such as using moving jobs together with changing processing times, are realized by joining the corresponding neighborhoods.

7 Evaluation

We assess our approach in a series of experiments that evaluate the PUP along three axes: *feasibility*, *robustness* to various privacy and utility configurations, and *effectiveness* in preserving key properties of the original schedule. Feasibility concerns the success rate of each perturbation function *S* ($\{\rightleftarrows\}$) , *M* ($\{\curvearrowright\}$), *P* ($\{\uparrow\downarrow\}$) and the combination of multiple perturbation functions *SP* ($\{\rightleftarrows, \uparrow\downarrow\}$) and *MP* ($\{\curvearrowright, \uparrow\downarrow\}$). Robustness evaluates PUP under different configurations, considering different privacy and utility thresholds. Effectiveness pertains to potential downstream analyses that are different from the original objective function. For example, for MAKE we ask: While the ISA targets the original schedule, which uses makespan as the objective function, can PUP preserve waiting times by adjusting utility z?

Data We rely on two datasets: (i) a synthetically generated set of schedules in a controlled setup, and (ii) a dataset drawn from real-world historical schedules of a health facility.

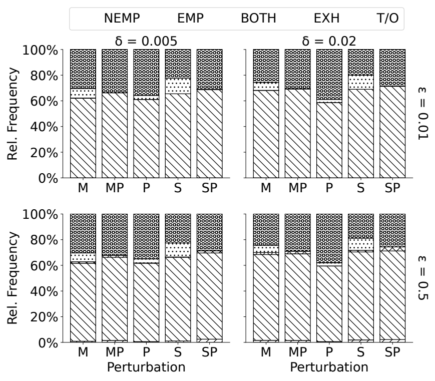
Synthetic data: A dataset consisting of 1000 schedules was created by using randomly drawn scheduling parameters to construct new schedules. The number of machines m and number of jobs n were drawn from $\{1, \ldots, 4\}$ and $\{5, \ldots, 20\}$, respectively. The parameters $\overline{p}$ and $\underline{p}$, were selected at random from $\{5, \ldots, 50\}$, s.t. $\underline{p} \leq \overline{p}$, to set the processing time domain $\mathcal{X}_p = \{\underline{p}, \ldots, \overline{p}\}$. For TWCT, the range $\{1, \ldots, w_{max}\}$ was assigned to the weight domain $\mathcal{X}_w$, with $w_{max} \in \{2, \ldots, 10\}$. Once the scheduling environment was determined, a schedule was instantiated by drawing p_j and w_j uniformly at random from domains $\mathcal{X}_p$ and $\mathcal{X}_w$, respectively, for each job $j \in J$. Similarly for MAKE, the release date domain $\mathcal{X}_r$ was set to $\{0, \ldots, r_{max}\}$, with $r_{max} \in \{1, \ldots, 9\}$, and a scheduling problem was created by drawing p_j and r_j uniformly at random from domains $\mathcal{X}_p$ and $\mathcal{X}_r$, respectively, for each job $j \in J$. The instantiated TWCT or MAKE scheduling problems were solved by employing the respective dispatching rule.

Real-world data: The dataset bases upon jobs extracted from real patient scheduling data of an outpatient cancer hospital in the United States. A schedule is generated by applying the WSPT rule to the n jobs that are assigned to the treatment "Infusion" (chemotherapy infusion) for each day and medical floor (with $\min = 17$, median $= 48.5$, and $\max = 84$ jobs per day and floor). Different medical floors correspond to different cancer types. The processing time domain $\mathcal{X}_p$ (in minutes) comprises the values from 5 to 720, which are also the observed minimum and maximum in the data (with median $= 120$). The weight domain $\mathcal{X}_w$ is assumed to span $[1 \ldots 5]$. The number of infusion chairs per floor represents the number of available machines m (with $\min = 15$, median $= 28$, and $\max = 33$) and the estimated time of a job j as the processing time p_j.

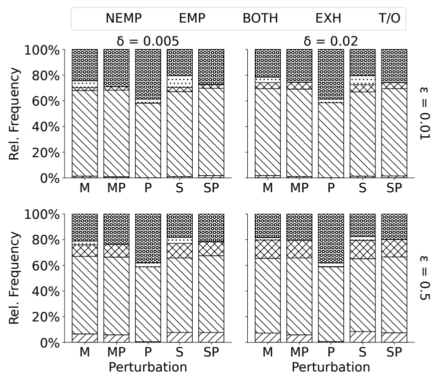
The time a job j was scheduled serves as a proxy for its weight w_j by ranking all the timestamps and then binning the ranks to fit the weight domain $\mathcal{X}_w$. In total, we considered 5 days (1 working week in April 2021) and 6 medical floors, which amounts to 30 historical schedules in total.

Setup We employ the privacy loss measure ξ_{dist} and the following instantiations of the utility function z: 1. the objective function of the scheduling problem considered, i.e., $\sum_j w_j C_j$ for TWCT or C_{max} for MAKE, and 2. the average waiting time $\sum_j \pi_j / n$ (AWT), an important characteristic of a schedule that is not directly incorporated in the scheduling objective. In the presence of release dates, AWT is adapted to $\sum_j (\pi_j - r_j)/n$. The difference d in utility between two schedules s and $\hat{s}$ is measured as the relative change in utility $|z(s) - z(\hat{s})|/z(\hat{s})$.

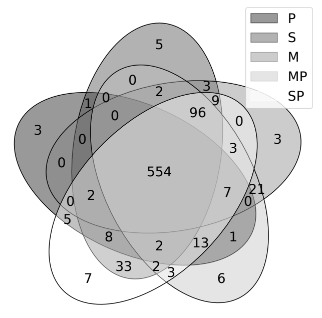
For both TWCT and MAKE, a set of schedules is generated for each utility function as described above. Further, perturbation functions $h \in \{S, P, M, MP, SP\}$ are assessed, with *S* denoting swapping of jobs ($\{\rightleftarrows\}$), *M* denoting moving of jobs ($\{\curvearrowright\}$), *P* and denoting feature perturbation ($\{\uparrow\downarrow\}$), while *MP* and *SP* capture their combinations.



(a) $z =$ TWCT.



(b) $z =$ AWT.



(c) $z =$ TWCT, $\delta = 0.02, \epsilon = 0.5$.

Figure 3: (a) and (b): Breakdown of the outcomes of the PUP for different perturbation functions under varying configurations (TWCT, synth. data); (c): Number of schedules exclusively solved by subsets $H \in \mathcal{P}^{\{M,P,S,SP,MP\}}$ (TWCT, synth. data).

We consider the values $\{0.01, 0.5\}$ for the privacy parameter ϵ and $\{0.005, 0.02\}$ for the utility threshold δ.

For each utility function and each parameter combination (h, ϵ, δ), the PUP is applied to each schedule with the respective configuration. Each PUP instance is allocated a time budget of 5min. A PUP instance failed, when the search stopped – due to either the timeout or exhausting the search space – without finding a solution. The PUP for a schedule s is considered successful, when (i) a solution with a non-empty candidate set ($\mathbb{Y}(s) \neq \emptyset$) or (ii) a solution with an empty candidate set ($\mathbb{Y}(s) = \emptyset$) had been encountered within the time limit. We also consider the case when both were eventually found (in which case the search also stopped). Thus, we distinguish the following outcomes:

- Solution found: *Non-Empty (NEMP), Empty (EMP)*, or *Both (BOTH)*
- No solution found: *Exhausted (EXH)* or *Timeout (T/O)*

A time budget of 60s was enforced when computing the privacy loss of a release candidate. If it was exceeded, the candidate was dismissed. The experiments with synthetic data ran on a Dell R920 server with 1TB RAM, 4/60/120 CPUs/cores/threads at 2.5GHz, running `openSUSE 15.3`. The real-world data experiments ran on a laptop with an i7-12800H 2.40 GHz processor and 64.0GB RAM. We used `Python 3.10` along with several libraries, most importantly `or-tools 9.4.1874` (Perron and Furnon 2022) for the ISA in the privacy loss computation. The code is publicly available on GitHub[1].

Results Figures 3a and 3b illustrate the distribution of outcomes across the 1000 synthetic PUP instances as well as the overall success rate for the utility functions *AWT* and *TWCT*, respectively. The PUP was solved successfully for 58.7-74.4% TWCT instances and 58.3-80.1% AWT instances, depending on the PUP configuration and the perturbation function in use. We note that using *P* alone consistently performs the worst. The other perturbation functions yield better success rates, but perform similarly among each other.

[1]https://github.com/hu-dbis/privacy-scheduling-tools

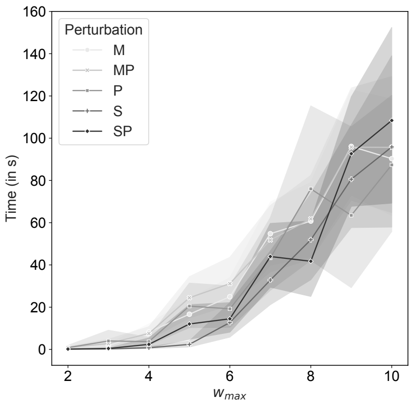
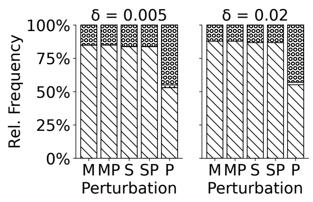


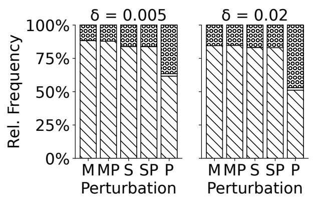
Figure 4: Time (in s) until first solution found depending on w_{max} with $z =$ TWCT, $\delta = 0.02, \epsilon = 0.5$ (synth. data).

With *AWT*, the perturbation function *SP* performs best in 3 out of 4 PUP configurations (with success rates ranging from 72.6-80.1% throughout the settings). The other perturbation functions *S*, *M*, and *MP* follow closely. Likewise, *SP* performs best for all PUP configurations when using *TWCT* as a utility function (with success rates ranging from 68.9-74.4% throughout the settings), most often followed by *MP* (with 66.3-71.0%). Furthermore, it becomes clear that the outcome *Empty* constitutes the majority of the solved PUPs, with the number of *non-empty* solutions increasing under a higher ϵ and δ. We observe that sometimes the search space was exhausted, but often a timeout caused the PUP to fail.

A more detailed view on the results obtained with the different distortion mechanisms is provided in the Venn diagram presented in Figure 3c. Specifically, we visualize the number of schedules exclusively solved by a certain subset of perturbation functions. It shows that the majority of the PUP was solved using all perturbation functions. Yet, each of the considered perturbation mechanisms was able to solve some instances of the PUP exclusively, meaning that any other set of perturbation functions was not successful

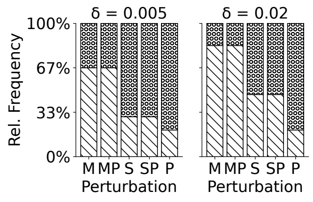


(a) $z = C_{max}$.

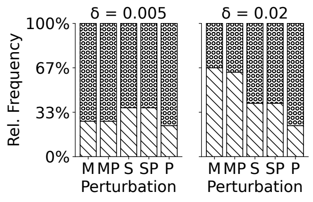


(b) $z =$ AWT.

Figure 5: Breakdown of PUP results for perturbation functions with $\epsilon = 0.01$ and varying δ (MAKE, synth. data).



(a) $z =$ TWCT.



(b) $z =$ AWT.

Figure 6: Breakdown of PUP results for perturbation functions with $\epsilon = 0.5$ and varying δ (real-world data).

for the respective problem instances. This empirical result highlights that the perturbation functions are, to some extent, complementary.

As part of our feasibility assessment, Figure 4 explores the efficiency of our approach. It shows that the runtime grows when w_{max} increases. Note that only the time measurements of successful PUP instances, whose number decreases with increasing w_{max}, are considered.

Figure 5 illustrates the breakdown of outcomes for solving the PUP for MAKE schedules with $z = C_{max}$ and $z =$ AWT, respectively ($\epsilon = 0.5$ omitted here due to obtaining similar results). We observe that *M*, *MP*, *S*, and *SP* were able to solve 83.8-87.9% of the PUP instances for $z = C_{max}$, whereas *S* managed 53.1-55.2% of the cases. Similarly, *M*, *MP*, *S*, and *SP* found a solution for 83-88.4% of the PUP instances with $z =$ AWT, while *S* solved 51.2-61.7% PUP instances successfully. As *empty* is the only recorded successful outcome, no solution with a non-empty candidate set during the ISA was found. The remaining PUP instances almost always ended in timeouts, and rarely due to exhausting the search space.

Figure 6 shows the overall success rate for the real-world data set for configurations that include a privacy loss threshold of $\epsilon = 0.5$ and varying utility thresholds (the outcomes are denoted in the same way as in Figures 3a and 3b). In general, *M* and *MP* tend to perform better than the other perturbation functions. There is also a noticeable difference in success rate with a higher utility threshold. We obtain the same results for $\epsilon = 0.01$ (omitted due to space constraints).

Discussion The results demonstrate that the PUP can be successfully solved for most synthetic schedules across various scheduling problems and privacy-utility configurations, highlighting both the feasibility and robustness of our approach. Using AWT as a utility function shows that the PUP effectively preserves schedule properties beyond the original objective function. However, for MAKE, while the *empty* outcome prevents an adversary from finding a combination of release dates that satisfy ISA constraints, it does not prevent the derivation of individual release dates for certain jobs. Also, the range of $\mathcal{X}_r$ and the spread of release dates may aid the adversary. We leave this area for future research.

Considering the choice among the proposed perturbation functions, the combined perturbation function *MP* shows the best overall success rate, which is in line with Property 1. However, the perturbation function *SP* mostly performs better than *MP* on the synthetic data set; functions *S* and *M* achieve a similar performance to *MP*, while inducing smaller neighborhoods. Also, each perturbation function appears to be uniquely suited to solve certain instances within the given time frame (see Figure 3c).

As expected, we found that a higher utility loss threshold δ leads to more neighboring schedules being evaluated w.r.t. the privacy loss, i.e., *P* performs slightly worse for $\delta = 0.02$, $\epsilon = 0.5$, and utility function *TWCT* compared to the same setup with $\delta = 0.005$. Further, the runtime depends on the size of the domain $\mathcal{X}_w$, which is expected since it determines the solution space of the solver used in the ISA.

The experiments on the real-world data confirm the general trends observed for the synthetic data. Our results indicate that the PUP is feasible for the majority of the schedules when using *M* or *MP*, once some utility loss is tolerated. A tighter utility bound reduces the number of successfully solved instances, especially with the AWT objective. The exclusive occurrence of the outcomes *Empty* and *Timeout* indicates that a large search space was explored at runtime.

8 Conclusion

This work addressed the challenge of protecting schedules from inverse scheduling attacks by introducing the Privacy-and-Utility Preservation Problem (PUP), which balances the trade-off between privacy and utility. The PUP seeks to identify a schedule that meets pre-determined privacy and utility criteria when published. We defined the problem for parallel machine environments and applied it to two well-known scheduling problems. We introduced scheduling-specific perturbation functions to solve the PUP.

Experiments on both synthetic and real-world schedules demonstrated (i) the general feasibility of the PUP, (ii) the robustness of the perturbation functions across different thresholds, and (iii) the effectiveness of our approach in preserving key analytical properties, such as average waiting time, beyond those captured by the utility function.

For future work, the PUP may be accelerated by use of adequate heuristics to enhance scalability. Another potential avenue for exploration is minimizing privacy or utility loss while constraining the other. Beyond that, advanced adversaries possessing comprehensive background knowledge may be assumed. Here, the case of an adversary not finding any values for the private features (i.e., $\mathbb{Y}(s) = \emptyset$), in particular, raises the question of a more flexible strategy that also incorporates schedules similar to the published one.

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF), grant number 16DII133 (Weizenbaum-Institute). We further acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) under the grant [RGPIN-2022-04593].

References

Brucker, P.; and Shakhlevich, N. V. 2009. Inverse Scheduling with Maximum Lateness Objective. *Journal of Scheduling*, 12(5): 475–488.

Bruno, J.; Coffman, E. G.; and Sethi, R. 1974. Scheduling Independent Tasks to Reduce Mean Finishing Time. *Communications of the ACM*, 17(7): 382–387.

Casas-Roma, J.; Herrera-Joancomartí, J.; and Torra, V. 2016. A Survey of Graph-Modification Techniques for Privacy-Preserving on Networks. *Artificial Intelligence Review*, 47(3): 341–366.

Chen, B.; and Vestjens, A. P. 1997. Scheduling on identical machines: How good is LPT in an on-line setting? *Operations research letters*, 21(4): 165–169.

Cheng, R.; Gen, M.; and Tsujimura, Y. 1999. A Tutorial Survey of Job-Shop Scheduling Problems Using Genetic Algorithms, Part II: Hybrid Genetic Search Strategies. *Computers & Industrial Engineering*, 36(2): 343–364.

DeSmet, C. N.; and Cook, D. J. 2021. Recent Developments in Privacy-preserving Mining of Clinical Data. *Trans. Data Sci.*, 2(4): 28:1–28:32.

Dwork, C. 2006. Differential Privacy. In *Automata, Languages and Programming*, 1–12. Springer Berlin Heidelberg.

Fahrenkrog-Petersen, S. A.; Senderovich, A.; Tichauer, A.; Tutak, A. K.; Beck, J. C.; and Weidlich, M. 2023. Privacy Attacks on Schedule-Driven Data. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10): 11972–11979.

Fung, B. C. M.; Wang, K.; Chen, R.; and Yu, P. S. 2010. Privacy-Preserving Data Publishing: A Survey of Recent Developments. *ACM Computing Surveys*, 42(4): 1–53.

Garey, M.; and Johnson, D. 1978. Strong NP-completeness results: motivation, examples, and implications. *J. Assoc. Comput. Mach.*, 25(3): 499–508.

Harding, J. A.; Shahbaz, M.; Srinivas; and Kusiak, A. 2005. Data Mining in Manufacturing: A Review. *Journal of Manufacturing Science and Engineering*, 128(4): 969–976.

Kawaguchi, T.; and Kyan, S. 1986. Worst Case Bound of an LRF Schedule for the Mean Weighted Flow-Time Problem. *SIAM Journal on Computing*, 15(4): 1119–1129.

Koulamas, C. 2005. Inverse Scheduling with Controllable Job Parameters. *International Journal of Services and Operations Management*, 1(1): 35.

Li, X.; and Olafsson, S. 2005. Discovering Dispatching Rules Using Data Mining. *Journal of Scheduling*, 8(6): 515–527.

Liu, K.; and Terzi, E. 2008. Towards Identity Anonymization on Graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. ACM.

Machanavajjhala, A.; Kifer, D.; Gehrke, J.; and Venkitasubramaniam, M. 2007. L-Diversity: Privacy beyond k-Anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1): 3.

Majeed, A.; and Lee, S. 2021. Anonymization Techniques for Privacy Preserving Data Publishing: A Comprehensive Survey. *IEEE Access*, 9: 8512–8545.

Mannhardt, F.; Koschmider, A.; Baracaldo, N.; Weidlich, M.; and Michael, J. 2019. Privacy-Preserving Process Mining - Differential Privacy for Event Logs. *Bus. Inf. Syst. Eng.*, 61(5): 595–614.

Min, L.; and Cheng, W. 1999. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, 13(4): 399–403.

Mou, J.; Li, X.; Gao, L.; and Yi, W. 2015. An Effective L-MONG Algorithm for Solving Multi-Objective Flow-Shop Inverse Scheduling Problems. *Journal of Intelligent Manufacturing*, 29(4): 789–807.

Perron, L.; and Furnon, V. 2022. OR-Tools. Google.

Pinedo, M. L. 2016. *Scheduling*. Springer International Publishing.

Samarati, P. 2001. Protecting Respondents Identities in Microdata Release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6): 1010–1027.

Sweeney, L. 2002. K-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05): 557–570.