Consent Management in Data Workflows: A Graph Problem

Dorota Filipczuk
University of Southampton
United Kingdom
dorota@ecs.soton.ac.uk

Enrico H. Gerding
University of Southampton
United Kingdom
eg@ecs.soton.ac.uk

George Konstantinidis
University of Southampton
United Kingdom
G.Konstantinidis@soton.ac.uk

ABSTRACT

In modern data processing systems users expect a service provider to automatically respect their consent in all data processing within the service. However, data may be processed for many different purposes by several layers of algorithms that create complex workflows. To date, there is no existing approach to automatically satisfy fine-grained privacy constraints of a user in a way which optimises the service provider's gains from processing. In this paper, we model a data processing workflow as a graph. User constraints and processing purposes are pairs of vertices which need to be disconnected in this graph. We propose heuristics and algorithms while at the same time we show that, in general, this problem is NP-hard. We discuss the optimality versus efficiency of our algorithms and evaluate them using synthetically generated data. On the practical side, our algorithms can provide a nearly optimal solution in the face of tens of constraints and graphs of thousands of nodes, in a few seconds.

1 INTRODUCTION

Modern computing has created a proliferation of large volumes of user data across data processing systems. When user data enters such a workflow, it is automatically processed, at several processing points for different purposes and by several service providers. Consequently, this processing creates new information, often conveying predictions and inferences that are used to fulfill a certain purpose. By doing so, the service provider may gain utility – e.g., monetary benefits in the case of a commercial website selling the data or providing personalised features – or achieve a research goal, e.g., in the case of a research organisation.

Such data workflows are used in many commercial, scientific and research applications. Consider for example the data workflow in Fig. 1, which is a real bioinformatics workflow found in [22], where an individual's genetic sequence is subject to different processing by different algorithms. While each algorithm creates additional value at each stage, the end purpose of data processing, in this example, is that of visualising the individual's phylogenetic tree; that is, placing them in an evolutionary diagram relative to their ancestors and other species.

Individuals, users and patients should have control over how their data is used in a data processing system and across different providers. and a user may refuse to consent to some of the data processing. Particularly, in certain regulatory frameworks such as the GDPR [13], unless there exists another legal basis, user's consent is necessary to legally allow the data to be processed.

Consider another example from a social media service provider seen in Fig. 2. Social media platforms have aggregated a multitude of in-website services that build on users' data, from dedicated e-shops and marketplaces all the way to disaster detection and

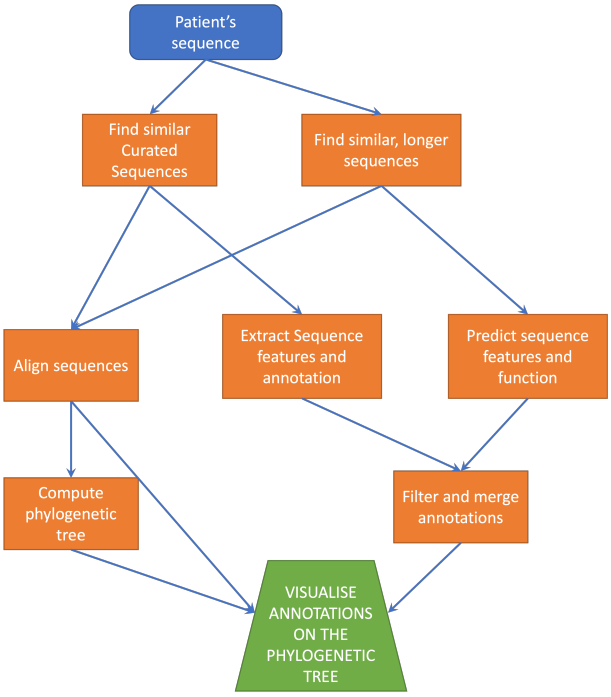


Figure 1: An example of a bioinformatics workflow, illustrating the flow of data through various services towards the end goal of visualisation[22].

notification of individual users. In Fig. 2 we have depicted a part of the data workflow for these two types of functions. User posts, including photos, can be used - in combination with outside sources such as sensors or video feeds - to detect a natural disaster and alert users based on their location; at the same time users' information including location is used to serve orders, advertisements or different kinds of recommendations. Again, although there is a growing interest and incentive to offer users fine-grained choices regarding their personal data usage in the system, today's options are limited.

In this example, the user may be happy for their purchase information, e.g., their home address, to be used for suggesting book clubs/communities to join, but may not wish to be subject to personalised product recommendations based on it. There are many alternatives to satisfy these user *privacy constraints* imposed on the data processing: the purchase prediction service could refuse to share its output with the product recommendation service. Another way would be for the geolocation service to not receive at all the user address, however this one seems more conservative, since the system will lose the capability of using the user's address or location for disaster notification purposes, even though the user might allow such a processing of their location.

Options such as these are expressed in Figure 2 as different paths from the user's nodes to the product recommendation node. In particular, in order to have the user's address not be used in

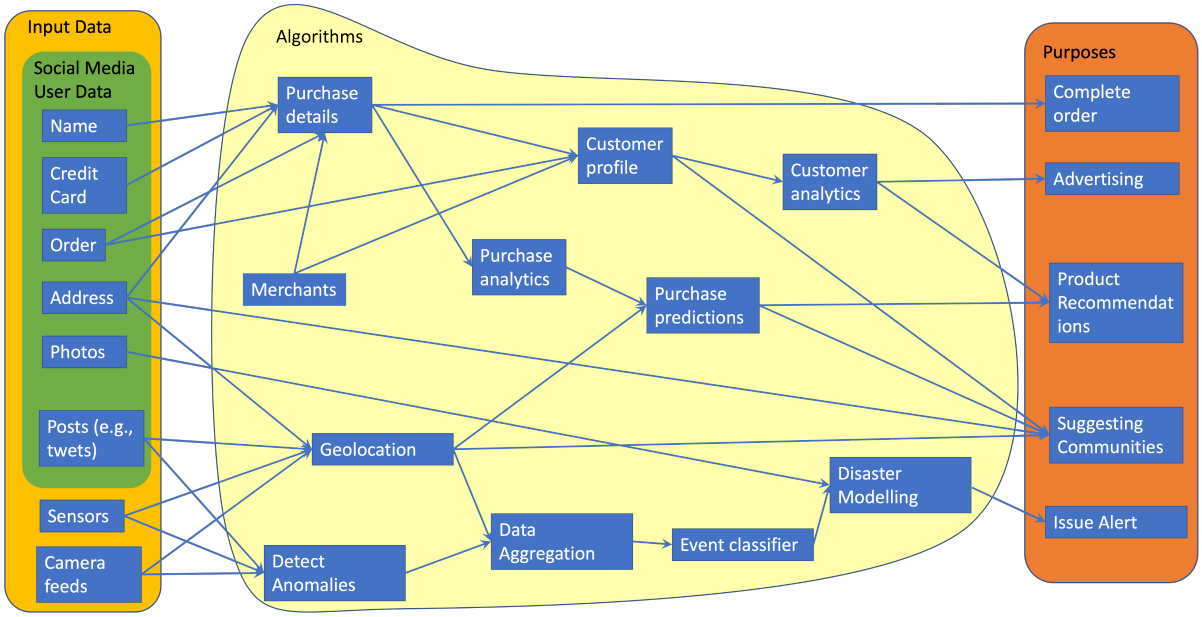


Figure 2: Part of social media data processing workflow, servicing a range of different purposes.

product recommendation we need to break paths between these two nodes in the figure. Evidently, enforcing privacy constraints may affect the utility of the service providers in different ways, so the option chosen should minimise the utility loss of the system overall. What complicates the task even more is its large scale: there may be many further stages of data processing, in a workflow that involves hundreds of nodes.

In this paper, we propose a novel approach to finding the most optimal ways of satisfying the user's privacy constraints. Specifically, we model the data workflow as a graph and privacy constraints as pairs of vertices of the graph. We formulate the problem as an optimisation problem, where pairs of graph vertices must be disconnected such that utility is maximised (Section 2).

We model the utility of a purpose node as a black-box function of the subgraph connected to that node, and suggest a simple breakdown of a prupose node's utility as a sum of the "value" that each edge has on this subgraph. We call this, still generic problem, CDW and we prove that it is NP-hard through a polynomial-time reduction from the minimum multicut problem (MinMC) (Section 3). Additionally, we present 5 alternative generic heuristics for implementing the privacy constraints into data workflows (Section 5) with an optimal or approximately optimal global utility, depending on the actual utility function used. In order to implement and present experiments, we present a simple concrete instance of the general problem, called CDW-LA, where we employ a utility that is linearly additive (Section 4), as a natural first instance of this problem.

Then, we analyse the (non-)optimality of our heuristics for CDW-LA with respect to their computational complexity (Section 6). We evaluate and compare the algorithms in terms of accuracy and performance (Section 7). Notably, we show that, although computing the optimal solution can be very time-consuming, the proposed heuristics can provide very accurate and efficient alternatives. Finally, we juxtapose this approach with related work (Section 9) and discuss the further lines of research that this paper opens (Section 8).

2 THE CONSENTED DATA WORKFLOW PROBLEM

We consider consent in data processing as a graph theory problem. First, we propose a new data processing model, which describes the workflow of personal information. Second, we present a formal formulation of the problem of identifying the parts of data processing workflows that align with the user's constraints, and bring the most benefits to the service provider.

2.1 Model

In order to describe the data workflow in the system, we formulate our data processing model as a directed graph $G = (V, E)$ with a set of edges E representing the data flow and a set of vertices V representing the stages of data processing. These stages refer to three activities: data collection from the user (start), algorithmic data processing (possibly at multiple stages) and satisfying the purpose of processing (end). For this reason, we distinguish three kinds of vertices, i.e. $V = V^U \cup V^A \cup V^P$, where:

- V^U is a set of user data vertices, which represent the types of data collected directly from the user, such as name, shipping address, credit card number. These will be vertices where the information originates from, and are seen as the input data vertices in Fig. 2;
- V^A is a set of algorithm vertices, which represent data processing algorithms that take one or more data types as input, e.g. purchase analytics and location, and output a new data type, e.g. predicted purchases – these are vertices in the intermediate layers of the graph, seen as "Algorithms" in Fig. 2;
- V^P is a set of purpose vertices, which represent the end goals of data processing, e.g. serving personalised advertising or suggesting communities to join – these are terminal vertices, (seen in the Purposes box in Fig. 2).

Importantly, we view data processing algorithms as 'black boxes'. That is, our model does not make assumptions about their internal workings – we only describe the input and output

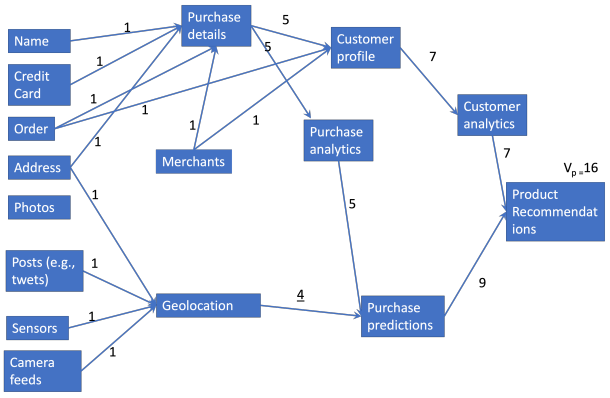


Figure 3: Reachability subgraph for purpose Product Recommendations from Fig. 2. The utility of the purpose is calculated in a linearly additive way: each node outputs value which is the sum of its incoming edges.

dependencies by the incoming and outgoing edges. To summarize, in our model, vertices in V^A have at least one incoming and at least one outgoing edge. Differently, vertices in V^U have no incoming edges and those in V^P no outgoing edges.

Furthermore, satisfying the given purposes is what brings service providers utility. For any vertex in V^P, we will have an associated utility that reflects the value processing that this purpose brings to service providers. In practice, the service providers' valuation depends on factors such as the accuracy of the datasets used as the input to the processing algorithms [14, 20]. In particular, where data processing is a multi-stage process, the utility is affected by all stages and all datasets processed for the purpose.

Therefore, in order to calculate the utility of data processing for a given purpose, in our model we look for all vertices and edges that carry the data workflow to the given purpose vertex. Formally, we say that a vertex $v_i \in V$ is *reachable* from a vertex $v_j \in V$ if there exists a path in G defined as a graph $(\{v_1, v_2, \ldots, v_k\}, \{(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)\})$ such that $v_1 = v_j$ and $v_k = v_i$. We also consider the so-called *reachability subgraph* of a purpose vertex. For each purpose vertex $p \in V^P$ our graph G, the reachability subgraph of p is the graph $G_p = (V_p, E_p)$ where $V_p \subseteq V$ is the set of the vertices that p is reachable from and $E_p \subseteq E$ is the set of edges from G that connect them. For example, the reachability graph of purpose "Product Recommendations" from Fig. 2 is shown in Fig. 3. If an edge is removed from G, the reachability subgraph of one or more purpose vertices is affected. In general, we write $\mathcal{R}(G_p)$ to denote the set of all subgraphs of the reachability subgraph of p in G - that are still reachability graphs when some edges are removed. Then, to calculate the utility of fulfilling a purpose, for each purpose vertex $p \in V^P$ we define a utility function $u_p : \mathcal{R}(G_p) \rightarrow \mathbb{R}_0^+$, which is a function of a reachability subgraph of p.

While u_p can be an arbitrary function dependent on the valuation of datasets in the corresponding reachability subgraph, the valuations of some datasets may influence the valuations of others. For example, when users are clustered based on their shipping address to be served personal advertising, the accuracy and, thus, valuation of the address may impact the accuracy, i.e. valuation of the clustering. To describe the relationships between these valuations in our model, we define a valuation function $\pi : E \rightarrow \mathbb{R}_0^+$, representing the valuation of the data propagating through the edge in the data processing system. As we later describe in Section 4, given the reachability subgraph $G_p = (V_p, E_p)$ of a vertex $p \in V^P$, the utility function $u_p(G_p)$ at p can be defined as a function of the valuations of edges in E_p.

2.2 Problem Formulation

The presented data processing model describes the workflow of data in the system. However, the user may refuse consent to some of this processing. For example, one of the user's constraints may be: *I'm happy for my shipping address to be used for recommending more products, but I don't want to be served more general advertising based on it.* To formulate this specific constraint, the user does not need to have an expert understanding of *how* their personal data is processed. Whereas, in order to satisfy this constraint, the system should be able to analyse the data workflow and make sure that there is no connection between the vertex where the shipping address enters the workflow and the vertex representing advertising. If there is no such connection, we call this data workflow *consented*.

Thus, our goal is to find the *consented data workflow* under certain constraints. We focus on what we call the Consented Data Workflow problem (CDW): given user's constraints expressed in terms of the vertices that they do not wish to be connected, find a subgraph of the original workflow where these constraints are satisfied. However, if there is more than one way of disconnecting the given vertices, the optimal solution should minimise the utility loss for the service provider from applying the constraints. In other words, we are looking for the utility-maximising solution subject to the users' privacy constraints.

Users' constraints can be expressed as a set of pairs of vertices. In this paper we focus on personal consent against a set of purposes, thus we will want to disconnect particular user vertices in V^U with purpose vertices in V^P. Formally, our set of constraints is a set $\mathcal{N} = \{(v_s, v_t) \mid v_s \in V^U, v_t \in V^P\}$. In order to satisfy the constraints, the initial graph G needs to be modified by removing one or more edges that belong to the paths between pairs (v_s, v_t), such that the utilities u_p are maximised. In essence, our problem is a multi-objective optimisation problem, where the objectives are to maximise u_p for all $p \in V^P$. The most common approach to multi-objective optimization is to turn the problem into a single-objective optimization using a weighted sum [21]. This allows us to define the utility of the system G as:

$$U(G) = \sum_{p \in V^P} w_p u_p(G_p), \tag{1}$$

where w_p is the weight of the purpose corresponding to vertex p and G_p is the reachability subgraph of p. Therefore, given $\mathcal{N}$, the objective of CDW is to find the *consented subgraph* of G:

$$G^* = arg \max_{G'} U(G') \tag{2}$$

such that $G' = (V, E')$ is a subgraph of G where $E' \subseteq E$ and there is no path from s to t for each $(s, t) \in \mathcal{N}$.

Note that, the way that the utility functions and their weights (and the weighted sum) can be estimated would be similar to the way that, for example, a business estimates its returns for online advertising. This is typically done by estimating the click-through-rate, the conversion rate, and the average value of each conversion. Each of these components would be influenced by the precision of the prediction, which in turn would depend on the data used as input. Additionally, in a real setting the values of the various data inputs could be established experimentally.

Note as well, that our framework can clearly support an algorithm that is unable to produce outputs in the absence of certain inputs. This would be implemented by having the utility function removing the outgoing edges of the algorithm as well (that is, certain edges have to be removed together with their successor edge). Elimination of the output edges would result in a normal subgraph and our algorithms can evaluate and treat this.

3 COMPLEXITY ANALYSIS

In this section we study the complexity of our problem. Specifically, we show that the CDW problem is $\mathcal{NP}$-hard. In order to do so, we reduce from the minimum multicut (MinMC) problem which is $\mathcal{NP}$-hard in directed acyclic graphs (DAGs) [7]. Given a DAG, $G = (V, E, w)$, where V is a set of vertices, E is a set of edges and $w : E \rightarrow \mathbb{N}^*$ an edge weight function, as well as a set $\mathcal{N}$ of pairs (source s, sink t) of terminal vertices of G, the objective of MinMC is to find a set of edges of G, E_{MinMC}, whose removal leaves no directed path from s to t for each $(s, t) \in \mathcal{N}$ such that:

$$E_{MinMC} = arg \min_{E'} \sum_{e \in E'} w(e). \tag{3}$$

To prove the complexity of CDW, let us consider an instance $I_{MinMC} = (G, \mathcal{N})$ of MinMC in DAGs and translate it into an instance of CDW. Let $G = (V, E)$. We are going to construct an instance of CDW on the same DAG G and the same constraints $\mathcal{N}$ such that, for a set of edges E^{MinMC}, the subgraph of G, $(V, E \setminus E^{MinMC})$ is a solution to the CDW instance if and only if E^{MinMC} is a solution to I_{MinMC}.

For simplicity, for any vertex $v \in V$, we use $in(v)$ to denote a set of incoming edges of v and $out(v)$ to denote outgoing edges in G. We first construct the set $V^U \subseteq V$ such that for any vertex $v \in V$ $in(v) = \emptyset$ if and only if $v \in V^U$. Similarly, we construct the set $V^P \subseteq V$ such that, for any vertex v, $out(v) = \emptyset$ if and only if $v \in V^P$. Note that V^U and V^P cannot be empty sets, because G is acyclic. Then, we construct a set $V^A = V \setminus (V^U \cup V^P)$. Moreover, we construct a purpose-reachability function $r : V^U \cup V^A \rightarrow \mathcal{P}(V^P)$, which for any vertex $v \in V^U \cup V^A$ returns a set of vertices in V^P such that $p \in r(v)$ iff p is reachable from v. Note that this construction happens in polynomial time. Then, for each edge $e = (v, v') \in E$, we construct a valuation function $\pi : E \rightarrow \mathbb{R}$ such that $\pi(e) = \frac{w(e)}{|r(v)|}$. In addition, for each $p \in V^P$, we have a utility function $u_p(G_p) = \sum_{e \in E_p} \pi(e)$, where $G_p = (V, E_p)$ is a reachability subgraph of p. Lastly, we construct a utility function of G as $U(G) = \sum_{p \in V^P} u_p(G_p)$, which implies that:

$$U(G) = \sum_{e \in E} w(e) \tag{4}$$

Note that each edge contributes to $U(G)$ exactly its original weight $w(e)$ because of the way $\pi(e)$ is constructed.

This concludes the construction of instance I_{CDW}, which is a CDW instance with the objectives weighted equally, i.e. for all $p \in V^P$, w_p is equal. Note that since all pairs $(s, t) \in$ consist of terminal vertices, $\mathcal{N}$ is the same set for I_{CDW}. Now, let us prove the following:

Lemma 3.1. *Given an instance I_{MinMC} of MinMC, there is a polynomial time reduction to an instance I_{CDW} of CDW such that a graph $(V, E \setminus E_{MinMC})$ is the solution to I_{CDW} iff E_{MinMC} is the solution to I_{MinMC}.*

Proof. ($\Leftarrow$) Let E_{MinMC} be a solution to I_{MinMC}. Since E_{MinMC} is a multicut of G given $\mathcal{N}$, this guarantees that when the edges in E_{MinMC} are removed from G, there is no directed path from s to t for each $(s, t) \in \mathcal{N}$. We show that the removal of set E_{MinMC} maximises the utility $U(G^*)$. Firstly, the set of edges after removal of E_{MinMC} from G can be expressed as:

$$E^* = E \setminus E_{MinMC}. \tag{5}$$

Then, if we plug Equation 3 to Equation 5, we have:

$$E^* = E \setminus \{arg \min_{E'} \sum_{e \in E'} w(e)\}. \tag{6}$$

Since E^* is a difference between E and the subset of E whose sum of edge weights is minimal, the sum of edge weights of E^* is maximal. Therefore, Equation 6 is equivalent to:

$$E^* = arg \max_{E'} \sum_{e \in E'} w(e). \tag{7}$$

Thus, if we plug Equation 4 into Equation 7, we have:

$$G^* = arg \max_{G'} U(G). \tag{8}$$

As this is exactly Equation 2, the graph $G^* = (V, E \setminus E_{MinMC})$ is the solution to I_{CDW}. Notably, this transition is performed in polynomial time.

($\Rightarrow$) Conversely, let $G^* = (V, E \setminus E_{MinMC})$ be a solution to I_{CDW}. By definition of the consented subgraph, in graph G^* there is no directed path from s to t for each $(s, t) \in \mathcal{N}$ and the utility is maximised as per Equation 2. If we plug Equation 4 into Equation 2, we have:

$$E^* = arg \max_{E'} \sum_{e \in E'} w(e). \tag{9}$$

Since $E^* \subseteq E$, there exists a set of edges $E \setminus E^*$ such that:

$$E \setminus E^* = E \setminus \{arg \max_{E'} \sum_{e \in E'} w(e)\}. \tag{10}$$

This is equivalent to:

$$E \setminus E^* = arg \min_{E'} \sum_{e \in E'} w(e). \tag{11}$$

If we call this set E_{MinMC}, i.e. $E \setminus E^* = E_{MinMC}$, then:

$$E_{MinMC} = arg \min_{E'} \sum_{e \in E'} w(e). \tag{12}$$

As this is exactly Equation 3, E_{MinMC} is the minimum multicut of G given $\mathcal{N}$. Therefore, E_{MinMC} is the solution to I_{MinMC}. Notably, this is achieved in polynomial time. □

Given this, we can now show that even for a small number of user's constraints our problem is $\mathcal{NP}$-hard, since MinMC is an $\mathcal{NP}$-hard problem in di-graphs:

Theorem 3.2. *CDW is $\mathcal{NP}$-hard, even if $|\mathcal{N}| = 2$.*

Proof. By Lemma 3.1, an instance of MinMC in DAGs can be converted to CDW in polynomial time. Moreover, solving CDW yields in polynomial time a solution to MinMC. Since MinMC in DAGs is known to be an $\mathcal{NP}$-hard problem for any $|\mathcal{N}| > 1$ [7], there exists a polynomial-time reduction from a known $\mathcal{NP}$-hard problem to CDW. Therefore, the CDW problem is $\mathcal{NP}$-hard for any $|\mathcal{N}| > 1$, which concludes the proof. □

4 ADDITIVE MODEL

As shown in Section 3, CDW in general is $\mathcal{NP}$-hard, which makes it difficult to expect a relatively efficient algorithm. Even more so because of the fact that

The valuations and utilities defined in Section 2.1, that we used in order to show NP-hardness can be arbitrary complex functions. In this section, we focus on a simple but practical instance of the problem, where these functions are linearly additive. In practice, data valuation is determined by a complex interaction of multiple factors including its age, accuracy and reliability [15]. Here, we choose a linear valuation function as a natural choice of a function. In particular, we assume that the valuation function of the data type going out of a vertex is linearly additive with respect to the importance of the data types on the incoming edges. That is, the value of every node's output is the sum of the values of the data products that reach its input (see Fig. 3). If we initialise each input edge to have an original value of 1, as in Fig. 3, then the utility of a purpose node sums up how many times the inputs "have been used" in algorithms before reaching the purpose node, thus giving some hint about the overall importance of an input. The linear additive model is the simplest model that captures the intuition of each input having an "added value" on the subsequent algorithm. This model may not apply to all settings and, e.g., a sub-additive model might be more appropriate. In many cases the linear additive model could be a reasonable approximation of a model where the interdependencies are difficult to measure.

In more detail, consider an instance of CDW, where for each edge $e = (v, v') \in E$, the valuation is defined recursively as follows:

$$\pi(e) = \sum_{e' \in in(v)} \pi(e'). \tag{13}$$

Similarly, we model the utility gained from processing the data for a purpose as a linearly additive function with respect to the valuation of the data types on the incoming edges. That is, for each purpose vertex $p \in V_P$ and its reachability subgraph G_p, we define a utility function as follows:

$$u_p(G_p) = \sum_{e \in in(p)} \pi(e). \tag{14}$$

Since the valuation is defined recursively, we also assume that our model has no cycles. That is, the original graph G is a directed acyclic graph (DAG).

Formally, the objective of our linearly additive instance of the CDW problem, called CDW-LA, is to find a the consented subgraph of G, given: a DAG $G = (V^U \cup V^A \cup V^P, E)$, a valuation function $\pi(e) = \sum_{e' \in in(v)} \pi(e')$ for each $e = (v, v') \in E$, a utility function $u_p(G_p) = \sum_{e \in in(p)} \pi(e)$ for each reachability subgraph G_p of each $p \in V_P$, a weight $w_p = 1$ of the purpose represented by vertex p and a set $\mathcal{N}$ of pairs (s, t) of terminal vertices of G such that $s \in V_U$ and $t \in V_P$.

5 ALGORITHMS

In this section, we devise a range of new algorithmic approaches that can solve the CDW-LA problem. Although there might exist multiple optimal solutions, we design our algorithms looking for a single solution $G^* = (V^*, E^*)$. While some of them offer optimal solutions to CDW-LA, others serve as viable heuristics. Note that, even though the algorithms may not be optimal (i.e. utility maximising), all five algorithms always return a *feasible* solution, which is any subgraph of G with no path between each $(s_i, t_i) \in \mathcal{N}$ for $i \in \{1, \ldots, |\mathcal{N}|\}$.

Firstly, a simple heuristic for finding a feasible solution is an algorithm that removes a random edge from each of the paths connecting $(s, t) \in \mathcal{N}$. In more detail, Algorithm 1 RemoveRandomEdge finds all paths from s to t (in lines 1 - 2) and from each of the paths selects a random edge to remove (in lines 3 - 4). Then, before the edge is removed (in line 7), the other edges whose valuation depends on the presence of the given edge in the graph must be updated. This is done by the `updateDependencies` function (in line 6). In particular, if the valuation of an edge after the update is 0, such edge must also be removed, e.g. if edge (s, v_1) in Figure **??** is removed, edges (v_1, t) and (v_1, t') also require removal. Although the solution has a high variance, the run time of this algorithm is polynomial.

Algorithm 1 RemoveRandomEdge

Input: A graph G and a set of constraints $\mathcal{N}$.
Output: A graph G.

```
1: for all (s, t) ∈ N do
2:     for all p ∈ getAllEdgePaths(G, s, t) do
3:         edgeIndex ← getRandomInteger(1, |p|)
4:         e ← p[edgeIndex]
5:         if hasEdge(G, e) then
6:             updateDependencies(G, e)
7:             removeEdge(G, e)
```

Secondly, as the valuation function of the edges is additive, and because the valuation of the incoming edge of an algorithm vertex is always greater or equal than the outgoing one, the removal of the first edge of each path from s to t can serve as another trivial heuristic. Specifically, Algorithm 2 RemoveFirstEdge is very similar to RemoveRandomEdge, except that, instead of selecting a random edge, it removes the first edge from each path (in line 3). This algorithm reflects an approach whereby the user's data type is removed entirely and not even collected by the system. Similarly to RemoveRandomEdge, the runtime of this algorithm is polynomial.

Algorithm 2 RemoveFirstEdge

Input: A graph G and a set of constraints $\mathcal{N}$.
Output: A graph G.

```
1: for all (s, t) ∈ N do
2:     for all path ∈ getAllEdgePaths(G, s, t) do
3:         e ← getFirstEdge(path)
4:         if hasEdge(G, e) then
5:             updateDependencies(G, e)
6:             removeEdge(G, e)
```

Next, we look for algorithms that can provide more accurate solutions. In particular, we propose a greedy algorithm that can provide a feasible solution in polynomial time. This algorithm follows the heuristic of making locally optimal choices for each constraint. To do so, it uses a polynomial-time algorithm solving the Minimum Cut problem (MinCut) [11, 12], defined as follows: given a graph $G = (V, E)$, a weight function $w : E \to \mathbb{N}^*$ and a single pair (source s, sink t) of terminal vertices of G, find a set of edges of G, E_{MinCut} whose removal leaves no directed path from s to t for each $(s, t) \in \mathcal{N}$ such that:

$$E_{MinCut} = arg \min_{E'} \sum_{e \in E'} w(e). \tag{15}$$

To design a greedy algorithm, we can use algorithms solving MinCut to find a minimum cut of G for each $(s,t) \in \mathcal{N}$. Consequently, we remove the minimum cut before moving on to the next constraint, which results in a partial solution. In more detail, Algorithm 3 RemoveMinCuts starts from initialising the weights $w(e) = \pi(e) \sum_{p \in r(v)} w_p$ for all edges $e \in E$ (in lines 1 - 4). Then, for each constraint $(s,t) \in \mathcal{N}$, it finds the minimum cut that solves MinCut for vertices s and t in G with weights w (in line 6). For each edge in the minimum cut, it uses the `updateDependencies` function to update the valuations of the consecutive edges (in line 8) before removing the given edge (in line 9). Given that MinCut is known to be solvable in polynomial time [11], the outcome of this heuristic can also be found in polynomial time.

Algorithm 3 RemoveMinCuts

Input: A graph $G = (V, E)$ and a set of constraints $\mathcal{N}$.
Output: A graph G.

1: $w \leftarrow \emptyset$
2: **for all** $e \in E$ **do**
3: $\quad w(e) \leftarrow \pi(e) \sum_{p \in r(v)} w_p$
4: **for all** $(s,t) \in \mathcal{N}$ **do**
5: $\quad$ **for all** $e \in$ MinCut(G, w, s, t) **do**
6: $\quad\quad$ **if** `hasEdge`(G, e) **then**
7: $\quad\quad\quad$ `updateDependencies`(G, e)
8: $\quad\quad\quad$ `removeEdge`(G, e)

Another way of approximating the solution is by converting our problem to MinMC, defined in Section 3. That is, we can solve MinMC with weights $w(e) = \pi(e) \sum_{p \in r(v)} w_p$ for all edges $e \in E$ and then use the MinMC solution to find a solution to CDW-LA. In the same way as RemoveMinCuts, Algorithm 4 RemoveMinMC starts from initialising the weights w (in lines 1 - 4). Then, it finds the minimum multicut of graph G for constraints $\mathcal{N}$ by executing the algorithm solving MinMC for input $(G, \mathcal{N}, w)$ (in line 5). Subsequently, for each edge in the minimum multicut, it uses the `updateDependencies` function to update the valuations of the consecutive edges (in line 8) before removing the given edge (in line 9).

Algorithm 4 RemoveMinMC

Input: A graph $G = (V, E)$, a set of constraints $\mathcal{N}$.
Output: A graph G.

1: $w \leftarrow \emptyset$
2: **for all** $e \in E$ **do**
3: $\quad w(e) \leftarrow \pi(e) \sum_{p \in r(v)} w_p$
4: `multicut` $\leftarrow$ MinMC$(G, \mathcal{N}, w)$
5: **for all** $e \in$ `multicut` **do**
6: $\quad$ **if** `hasEdge`(G, e) **then**
7: $\quad\quad$ `updateDependencies`(G, e)
8: $\quad\quad$ `removeEdge`(G, e)

Finally, we propose an algorithm that can guarantee achieving an optimal solution. That is, Algorithm 5 BruteForce is an exhaustive search algorithm that enumerates all feasible candidates for the solution and compares them to eventually output the one that maximises the utility. More specifically, BruteForce starts from finding the set of all paths $\mathcal{A}$ from s to t for all $(s,t) \in \mathcal{N}$, which need to be broken (in lines 1 - 4). In order to list all feasible multicuts of G for the given $\mathcal{N}$, the Cartesian product of $\mathcal{A}$ is computed (in line 5). Then, the algorithm systematically checks the utility of G after the removal of each multicut (in lines 8 - 27). Importantly, at the beginning of the multicut check, copies are made of the valuation values π' of each edge and the number of paths p' the edge belongs to in G, as well as of the graph G itself (in lines 9 - 14). Before an edge of the feasible multicut is removed from the copy of G (in line 18), the valuation π' and the number of paths p' in the copy of G are updated for its dependencies (in line 17). At the end of the multicut check, the utility of the copy of G is compared to the utility of the most optimal solution found so far (in lines 21 - 25). This way, given that all possible solutions that satisfy the constraints are checked, the algorithm can guarantee eventually finding the optimal solution. However, the runtime of this algorithm is exponential even in the best case.

Table 1: Algorithm Comparison.

Algorithm	Runtime	Exact Solution
RemoveRandomEdge	Polynomial	No
RemoveFirstEdge	Polynomial	No
RemoveMinCuts	Polynomial	No
RemoveMinMC	Exponential	No
BruteForce	Exponential	Yes

Algorithm 5 BruteForce

Input: A graph $G = (V, E)$ and a set of constraints $\mathcal{N}$.
Output: A graph G^*.

1: $\mathcal{A} \leftarrow \emptyset$
2: **for all** $(s,t) \in \mathcal{N}$ **do**
3: $\quad \mathcal{A} \leftarrow \mathcal{A} \cup$ `getAllEdgePaths`(G, s, t)
4: `multicuts` $\leftarrow$ `cartesianProduct`$(\mathcal{A})$
5: `maxUtility` $\leftarrow 0$
6: $G^* \leftarrow G$
7: **for all** `multicut` $\in$ `multicuts` **do**
8: $\quad G' \leftarrow G$
9: $\quad \pi', p \leftarrow \emptyset, \emptyset$
10: $\quad$ **for all** $e \in E$ **do**
11: $\quad\quad \pi'(e) \leftarrow \pi(e)$
12: $\quad\quad p(e) \leftarrow \sum_{p \in r(v)} w_p$
13: $\quad$ **for all** $e \in$ `multicut` **do**
14: $\quad\quad$ **if** `hasEdge`(G', e) **then**
15: $\quad\quad\quad$ `updateDependencies`(G', e, π', p)
16: $\quad\quad\quad$ `removeEdge`(G', e)
17: $\quad$ `utility` $\leftarrow U(G')$
18: $\quad$ **if** `utility` $>$ `maxUtility` **then**
19: $\quad\quad$ `maxUtility` $\leftarrow$ `utility`
20: $\quad\quad G^* \leftarrow G'$
21: **return** G^*

While Alg. 1, 2 and 5 are designed to work on models with arbitrary valuation and purpose utility functions, specifying these functions is needed to calculate the weights $w(e)$ in Alg. 3 and 4. Table 1 presents a summary of the different algorithms.

6 OPTIMALITY OF SOLUTIONS

Out of five algorithms proposed in Section 5, only BruteForce guarantees an optimal solution to CDW-LA. In contrast, it is clear that RemoveRandomEdge does not guarantee an optimal

solution – we use it as a benchmark for our evaluation. In this section, we analyse the properties of the solutions returned by our three remaining heuristics and prove that none of them can guarantee finding an optimal solution even for the linear setting.

Firstly, we show that a simple removal of the first edge of each path proposed in Algorithm 2 RemoveFirstEdge does not guarantee an optimal solution. In more detail, for each $(s_i, t_i) \in \mathcal{N}$, there is at least one path $P = (V_P, E_P) \in \mathcal{A}$ of the form $V_P = \{v_1, v_2, \ldots, v_k\}, E_P = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}$ where $v_1 = s_i$ and $v_k = t_i$. From each such path $P \in \mathcal{A}$, we could remove edge (v_1, v_2). We refer to (v_1, v_2) as the *first edge*. We show that the removal of the first edge from each P does not always result in an optimal solution to CDW-LA by the following example.

Let G be a data processing model where $V^U = \{v_1\}$, $V^A = \{v_2\}$, $V^P = \{v_3, v_4\}$, $E = \{(v_1, v_2), (v_2, v_3), (v_2, v_4)\}$ and for each $p \in V^P$, $w_p = 1$. In addition, assume that for edge $e_1 = (v_1, v_2)$, $\pi(e_1) = a$ where $a \in \mathbb{R}_0^+$ and that $\mathcal{N} = \{(v_1, v_3)\}$.

In such case, we use Equation 13 to calculate the valuation of edges $e_2 = (v_2, v_3)$ and $e_3 = (v_2, v_4)$, which is $\pi(e_2) = \pi(e_3) = a$. We also use Equation 1 to calculate the initial utility of G, which is $U(G) = 2a$. Given that $\mathcal{N} = \{(v_1, v_3)\}$, we establish that there is one path that needs to be disconnected in order to satisfy the constraints, i.e. $\mathcal{A} = \{P\}$ where $P = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3)\})$.

Then, we remove the first edge (v_1, v_2) from P. The utility of the resulting graph G_1' is $U(G_1') = 0$, since purpose vertices v_3 and v_4 are now not linked to any user vertex. However, if instead we removed the alternative edge (v_2, v_3), vertex v_4 would still be linked to v_1 and therefore the utility of the resulting graph G_2' would be $U(G_2') = a$. Thus, in this case the removal of the first edge does not provide us with an optimal solution.

In a similar way, we can prove that removing the last edge (v_{k-1}, v_k) from each path in $\mathcal{A}$ also does not guarantee the optimal solution. However, since for all $(s, t) \in \mathcal{N}$ all paths from s to t are broken, it similarly provides a feasible solution.

Furthermore, we show that solving the problem in a greedy way, i.e. where we apply the constraints one at a time as proposed in Algorithm 3 RemoveMinCuts, also does not lead to an optimal solution. Specifically, consider a series of graphs $G^0, G^1, \ldots, G^{|\mathcal{N}|}$. These graphs are computed recursively such that $G^0 = G = (V, E)$ and for all $i \in \{1, \ldots, |\mathcal{N}|\}$, $G^i = (V, E^i)$ where $E^i = E^{i-1} \setminus \text{MinCut}(G^{i-1}, s_i, t_i, w)$ corresponds to i-th pair of vertices in $\mathcal{N}$. Given a solution to MinCut, one could transform CDW-LA into a repeated MinCut problem looking for $G^{|\mathcal{N}|}$. While this approach leads to a feasible solution, $G^{|\mathcal{N}|}$ is not necessarily an optimal solution.

We show this by the following example. Let G be a data processing model where $V^U = \{s_1, s_2\}$, $V^A = \{v_1\}$, $V^P = \{t_1, t_2\}$, $E = \{(s_1, v_1), (s_2, v_1), (v_1, t_1), (v_1, t_2)\}$ and for each $p \in V^P$, $w_p = 1$. Assume that for $e_1 = (s_1, v_1)$, $\pi(e_1) = a$ and for $e_2 = (s_2, v_1)$, $\pi(e_2) = b$, where $a, b \in \mathbb{R}_0^+$ and $a > b$. This model is illustrated in Figure 4. In addition, there are two constraints: $\mathcal{N} = \{(s_1, t_1), (s_1, t_2)\}$.

In such case, $i = 2$. We look for $G^2 = (V, E \setminus \text{MinCut}(G^1, s_1, t_1, w))$. Thus, we first calculate $G^1 = (V, E \setminus \text{MinCut}(G, s_1, t_1, w))$. We observe that there is one path $P = (\{s_1, v_1, t_1\}, \{(s_1, v_1), (v_1, t_1)\})$ between vertices s_1 and t_1. Because $a > b$, $w((v_1, t_1)) = a + b < w((s_1, v_1)) = 2a$. Thus, $G^1 = (V, E \setminus \{(v_1, t_1)\})$. With this information, we return to looking for G^2. We observe that there is one path $P = (\{s_1, v_1, t_2\}, \{(s_1, v_1), (v_1, t_2)\})$ between vertices s_1 and t_2. However, now $w((s_1, v_1)) = a$ and $w((v_1, t_2)) = a + b$. So, $w((s_1, v_1)) < w((v_1, t_2))$ and $G^2 = (V, E \setminus \{(s_1, v_1), (v_1, t_1)\})$.

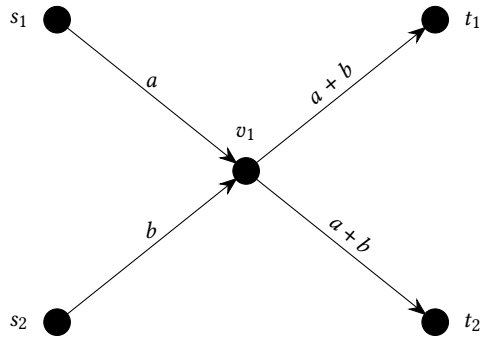


Figure 4: A data processing model where $V^U = \{s_1, s_2\}$, $V^A = \{v_1\}$, $V^P = \{t_1, t_2\}$ and $E = \{(s_1, v_1), (s_2, v_1), (v_1, t_1), (v_1, t_2)\}$.

After that, we calculate utility $U(G^2) = b$. However, we can see that in order to optimally solve CDW-LA, it is sufficient to remove edge (s_1, v_1) only. That is, the optimal solution to CDW-LA in this case is $G^* = (V, E \setminus \{(s_1, v_1)\})$, because its utility is $U(G^*) = 2b$. Thus, G^2 is not an optimal solution to CDW-LA.

Intuitively, it is reasonable to assume that the optimal solution requires removing no more than one edge per path between s and t for each $(s, t) \in \mathcal{N}$. In what follows, we first prove that, for settings where this is indeed the case, our Algorithm 4 RemoveMinMC finds the optimal set of edges to remove. However, we then show that there are settings where this assumption does not hold, and where the optimal solution requires removing more than one edge from the same path. However, in Section 7 we show that these cases are rare and, in most cases, the algorithm does return the optimal solution. Hence showing that the algorithm guarantees the optimal solution in restricted settings is useful.

Theorem 6.1. *If for each path $P = (V_P, E_P) \in \mathcal{A}$ there is exactly one edge $e \in E_P$ such that $e \in E_{MinMC}$, then there exists a solution $G^* = (V, E \setminus E_{MinMC})$.*

Proof. Let $T \subseteq V^P$ be a set of purpose vertices such that for all $(s, t) \in \mathcal{N}$, $t \in T$. If for each path $P = (V_P, E_P) \in \mathcal{A}$ there is exactly one edge $e \in E_P$ such that $e \in E_{MinMC}$, then the removal of a set of edges E_{MinMC} reduces the utility of a purpose vertex $t \in T$ by $\sum_{e \in E_t} \pi(e)$ where $E_t \subseteq E_{MinMC}$ is a set of those edges in E_{MinMC} that are within the reachability subgraph of t, G_t. Thus, if the resulting graph after the removal of set E_{MinMC} from G is $G' = (V, E')$, then using Equation 1, the total loss of the utility can be calculated as follows:

$$U(G) - U(G') = \sum_{t \in T} \sum_{e \in E_t} w_t \pi(e). \tag{16}$$

This is equivalent to the following equation:

$$U(G') = U(G) - \sum_{e \in E \setminus E'} \pi(e) \sum_{t \in T} w_t. \tag{17}$$

We are looking for the consented subgraph $G^* = (V, E^*)$. In fact, if we plug Equation 17 into Equation 2, we have:

$$G^* = arg \max_{G'} \{U(G) - \sum_{e \in E \setminus E'} \pi(e) \sum_{t \in T} w_t\}. \tag{18}$$

Equivalently, we are looking for a subgraph $G^* = (V, E^*)$ where:

$$E^* = E \setminus \{arg \min_{E \setminus E'} \sum_{e \in E \setminus E'} \pi(e) \sum_{t \in T} w_t\}. \tag{19}$$

Thus, by Equation 3, E^* is the set difference of E and the minimum multicut of G given $\mathcal{N}$, where the edge weight is $w(e) = \pi(e) \sum_{t \in T} w_t$. In more detail, the minimum multicut with edge weights $w(e) = \pi(e) \sum_{t \in T} w_t$ can be expressed as:

$$E_{MinMC} = arg \min_{E'} \sum_{e \in E'} \pi(e) \sum_{t \in T} w_t. \tag{20}$$

If we plug Equation 19 into Equation 20, then what we are looking for is $G^* = (V, E^*)$ where:

$$E^* = E \setminus E_{MinMC}. \tag{21}$$

Since E_{MinMC} is a solution to MinMC, there is $G^* = (V, E \setminus E_{MinMC})$. □

However, removing a single edge from each path does not always result in an optimal solution. We prove this by the following example. Consider a graph G where $V^U = \{s_1, s_2\}$, $V^A = \{v_1\}$, $V^P = \{t_1, t_2\}$, $E = \{(s_1, v_1), (s_2, v_1), (v_1, t_1), (v_1, t_2)\}$ and for each $p \in V^P$, $w_p = 1$. In addition, assume that for $e_1 = (s_1, v_1)$, $\pi(e_1) = a$ and for $e_2 = (s_2, v_1)$, $\pi(e_2) = b$, where $a, b \in \mathbb{R}_0^+$ and $a > b$. This graph is illustrated in Figure 4. Now, let the set of constraints be as follows: $\mathcal{N} = \{(s_1, t_1), (s_1, t_2), (s_2, t_1)\}$. We can see that the optimal solution in this case is $G^* = (V, \{(s_2, v_1), (v_1, t_2)\})$. However, since $(s_1, t_1) \in \mathcal{N}$ and there is a path from s_1 to t_1 in the original graph G, we can observe that the optimal solution G^* does not contain two edges (s_1, v_1) and (v_1, t_2) from that path.Therefore, in general, it is not true that Algorithm 4 RemoveMinMC can guarantee finding an optimal solution to CDW-LA by removing only one edge from each path.

7 EVALUATION

In this section, we evaluate the performance of the proposed algorithms empirically. First, we discuss the experimental setup, including details on the algorithm implementation and graph data generation. Then, we present results of the experiments focusing on the runtime and accuracy of each algorithm.

For our experiments, we have used synthetic data, since the very nature of privacy and consent makes obtaining real data very difficult. Users would be hesitant to share the ground truth of their data and their privacy preferences on top of that. Moreover, commercial service providers are still hesitant to share their internal data workflow models. Most importantly, there has not been a technology so far to support such fine-grained user privacy constraints that disconnect arbitrary input data from an arbitrary workflow purposes; thus real data is rare on this problem. We believe that this is changing; service providers tend to offer more choices and research that offers automation of fine-grained consent is starting to take off [18].

7.1 Experimental Setup

We implement the proposed algorithms using the NetworkX[1] library. In particular, this library provides a method to solve the MinCut problem in Algorithm 3. In addition, we implement Algorithm 4 using the PICOS[2] API for optimization solvers. Specifically, we use the GLPK (GNU Linear Programming Kit)[3] package for solving the MinMC problem.

We compare the different algorithms by measuring their performance on synthetic data. This choice allows us to test the algorithms when all the assumptions of the CDW-LA instance are met. To do so, our graph generation method includes the following parameters:

- number of constraints $|\mathcal{N}|$;
- number of vertices $|V|$;
- path length k – for any $(s, t) \in \mathcal{N}$, if there is a path $P = ((v_1, v_2, \ldots, v_k), ((v_1, v_2) \ldots (v_{k-1}, v_k)))$ such that $v_1 = s$ and $v_k = t$, then k defines the number of workflow stages, i.e. data that 'flows' from s to t though $k - 2$ algorithm nodes;
- vertex distribution vector X_k – proportions of vertices at workflow stages, e.g. a setting $X_k = (50\%, 25\%, 10\%, 10\%, 5\%)$ represents a scenario for $k = 5$ where half of the vertices are the user data vertices, 35% are the algorithm vertices and 5% are the number of the purpose vertices;
- minimum density d – the proportion of initially generated edges between any two workflow stages.

Table 2: Parameter configurations for datasets 1, 2 and 3.

	Dataset 1			Dataset 2	Dataset 3
	a	b	c		
$\|\mathcal{N}\|$	1 – 50	1 – 50	1 – 50	10	5
$\|V\|$	100	1000	100	150 – 5000	100 – 10000
k	5	5	5	3 – 50	5
X_k	NU	NU	U	U	NU
d	0	0	20%	0	0

To generate a graph, we distribute $|V|$ vertices onto k workflow stages as per vector X_k. For any two workflow stages, the initial d of all possible edges are generated though a pseudo-random[4] selection of vertices. Then, to ensure that all vertices in V^U and V^A have at least one outgoing edge, and that all vertices in V^A and V^P have at least one incoming edge, we add additional edges with one of the vertices selected randomly. All edges going out of the user vertices are assigned integer valuations $\pi(e)$ through a uniform selection from a range of 1–100. Furthermore, for each purpose vertex $p \in V_p$, the purpose weight introduced in Equation 1 is set to $w_p = 1$. Then, we generate the set of constraints by selecting $|\mathcal{N}|$ distinct pairs of randomly selected user data vertices and purpose vertices, ensuring that for any $(s, t) \in \mathcal{N}$, there exists at least one path from s to t.

This way, we prepare three datasets with different configurations of the above parameters. Firstly, to observe how the number of constraints affects the runtime of algorithms and graph utility, we generate dataset 1. We create this dataset in different variants: a variant with 100 vertices (1a), a variant with 1000 vertices (1b) and a variant with a minimum density of 20% (1c). To observe how the shape of the graph affects the runtime and utility, we apply a non-uniform vertex distribution ($X_k = (50\%, 25\%, 10\%, 10\%, 5\%)$ abbreviated as 'NU') in variants 1a and 1b, and a uniform vertex distribution ($X_k = (20\%, 20\%, 20\%, 20\%, 20\%)$ abbreviated as 'U') in variant 1c. Secondly, to observe the impact of the path length on the runtime and utility, we generate dataset 2 with graphs that have a constant number of paths. To do so, we first generate graphs with $|V| = 150$, $k = 3$, and vertices distributed uniformly such that $|V^U| = 50$, $|V^A| = 50$ and $|V^P| = 50$. Then, we keep generating new graphs by adding 50 additional vertices to the previous graph and connecting each vertex to the graph with a single edge. This way we extend the length of each path in

[1] NetworkX, https://networkx.org/.
[2] PICOS, https://picos-api.gitlab.io/picos/.
[3] GLPK (GNU Linear Programming Kit), https://www.gnu.org/software/glpk/.
[4] For details, see the Python Standard Library documentation: https://docs.python.org/3/library/random.html.

the previous graph while keeping the number of paths constant. We also adjust the constraints such that they relate to the same paths as for the previously generated graph. Finally, to observe how the size of the graph affects the runtime and utility, we generate graphs of 100–10,000 vertices with a constant number of constraints. The exact parameter configuations are specified in Table 2.

We perform our experiments on the University of Southampton High Performance Computing service Iridis 4[5] which offers 750 compute nodes in total with dual 2.6 GHz Intel Sandybridge processors. Each compute node has 16 CPUs per node with 64 GB of memory and the maximum runtime of a job is 60 hours. In order to ensure that the average result has low variance, we repeat the experiments until we have at least 30 runs with a sufficiently low standard error (SE).

7.2 Results

We apply the algorithms from Section 5 to datasets 1, 2 and 3. In this section, we report on the runtime of the algorithms and changes in the graph's utility with respect to the number of privacy constraints, number of data processing stages and the size of the workflow.

7.2.1 Number of constraints. Figure 5 shows the runtime of the algorithms as the number of privacy constraints grows. We measure their performance on 100-vertex graphs (dataset 1a) and compare it to the performance on 10 times larger, 1000-vertex graphs (dataset 1b). Since in datasets 1a and 1b the number of paths between the constraints is equal or close to the number of constraints given, we also consider slightly denser 100-vertex graphs, where the number of edges between each level of data processing is at least 20% of all possible edges (dataset 1c). In general, when we compare the average runtime on datasets 1a (Figure 5a), 1b (Figure 5b) and 1c (Figure 5c), we observe very similar trends. As the graph size increases 10 times, the average runtimes of BruteForce, RemoveMinMC and RemoveMinCuts also increases approximately 10 times. This suggests that, as expected, the execution time of these three algorithms depends on graph size and the number of constraints given as input.

In particular, we observe that the runtime of BruteForce increases rapidly with the increasing number of constraints, reaching an average time of 14838508.46 ms (i.e. over 4 h) for just 10 constraints on dataset 1a and 8563968 ms (i.e. over 2 h; SE: 1058979.73 ms) on dataset 1b. For dataset 1c, in most cases, BruteForce is unable to return a result in 60 hours even for just a single pair of constraints. Thus, although BruteForce guarantees finding an optimal solution, its average runtime on such a small and sparse graphs makes this algorithm impractical. At the same time, the solver-based RemoveMinMC can reach an approximate solution for even 50 constraints on average in 6.51 seconds (SE: 405.02 ms) on dataset 1a, 74.1 seconds (SE: 439.96 ms) on dataset 1b and 11 seconds (SE: 290.47 ms) on dataset 1c. RemoveMinCuts can on average find an approximate solution for 50 constraints in 220.2 milliseconds (SE: 1.1 ms) on dataset 1a, 2.55 seconds (SE: 17.27 ms) on dataset 1b and 450.57 ms (SE: 3.17 ms) on dataset 1c.

Moreover, we observe the change in the graph's utility as the number of privacy constraints grows. Specifically, in Figure 6a, we consider sparse graphs from dataset 1a. We can see that for RemoveMinMC, RemoveMinCuts and RemoveFirstEdge the utility of the graph after applying the algorithm decreases almost linearly as the number of constraints grows. Out of these three, RemoveMinMC tends to provide the most accurate solutions, reducing the utility down to 17.63% on average (SE: 1.15%) for 50 constraints. Although the exponential runtime of the BruteForce algorithm means we cannot run the experiments for more than 10 constraints, we still compare the results to RemoveMinMC for this limited setting. Results are presented in Table 3 and show that the utility using RemoveMinMC is nearly optimal in this case. In Section 6, we have shown that the algorithm is only guaranteed to be optimal for specific settings where the optimal solutions consists of only a single edge being removed from each path. Nevertheless, this empirical outcome suggests that, for graphs with a relatively small number of constraints RemoveMinMC is likely to provide very accurate solutions.

Table 3: Comparison of the graph's utility after applying RemoveMinMC and BruteForce.

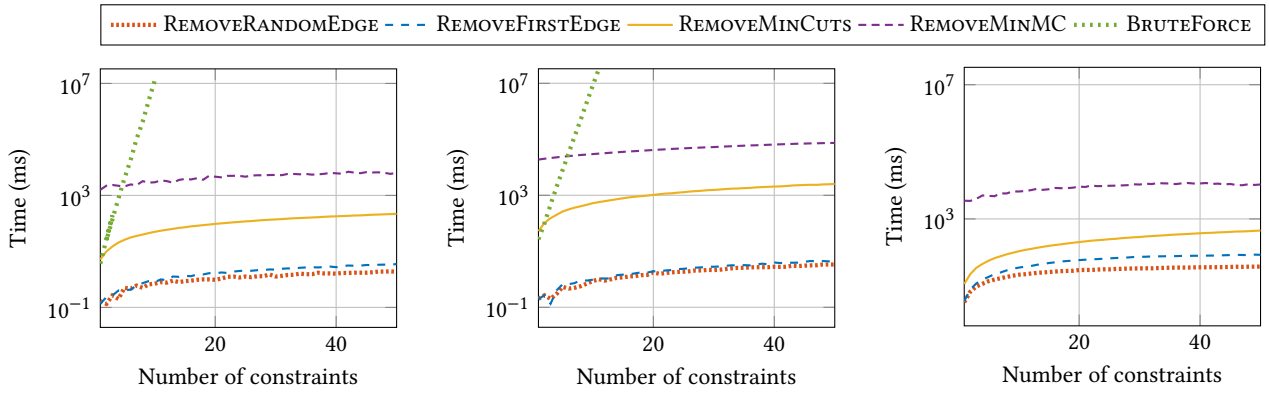
Number of constraints	RemoveMinMC		BruteForce	
	% of original	SE	% of original	SE
1	97.79	0.32	97.79	0.32
2	95.08	0.35	95.08	0.35
3	92.71	0.58	92.71	0.58
4	90.62	0.57	90.63	0.57
5	88.59	0.75	88.65	0.75
6	86.59	0.72	86.66	0.72
7	84.71	0.72	84.77	0.71
8	83.22	0.70	83.28	0.70
9	81.24	0.69	81.33	0.69
10	79.30	0.69	79.39	0.68

In Figure 6b, we consider the utility changes in sparse graphs with 1000 vertices distributed non-uniformly (dataset 1b). As in Figure 6a, RemoveMinMC provides solutions with the highest utility, i.e. on average 89.24% (SE: 0.19%) given 50 constraints. As expected, the utility here is higher than in Figure 5a. After applying the algorithms with same number of constraints, proportionally less paths are broken in the graphs with 1000 vertices than with 100 vertices. At the same time, this results suggests that, for very large graphs, faster algorithms such as RemoveMinCuts or even RemoveFirstEdge may be able to provide sufficiently accurate solutions.

In Figure 6c we consider denser graphs with 100 vertices. We observe that the differences in utility between algorithms is more evident when the graphs are denser, resulting in significantly poorer performance especially for RemoveMinCuts, RemoveFirstEdge and RemoveRandomEdge. This is because denser graphs have more paths that need to be broken. Nonetheless RemoveMinMC provides the best solution with average utility being 33.29% (SE: 1.09%) of original utility of the graph.

7.2.2 Number of paths. Next, we observe how the execution time depends on the number of paths between pairs of vertices that connect the constraints. Figure 7 presents a scatter plot of the runtime of the algorithms and distribution of utility in dense graphs (dataset 1c). In particular, we can see that, in case of RemoveMinCuts and RemoveMinMC, the runtimes increase almost linearly with respect to the number of paths. However, the execution times for these two algorithms differ significantly. For example, for a graph where 822 paths are required to be broken, RemoveMinMC takes 12116 ms to return a solution, whereas RemoveMinCuts can provide one in only 472 ms. Similarly, we can

[5]The Iridis Compute Cluster, https://cmg.soton.ac.uk/iridis.

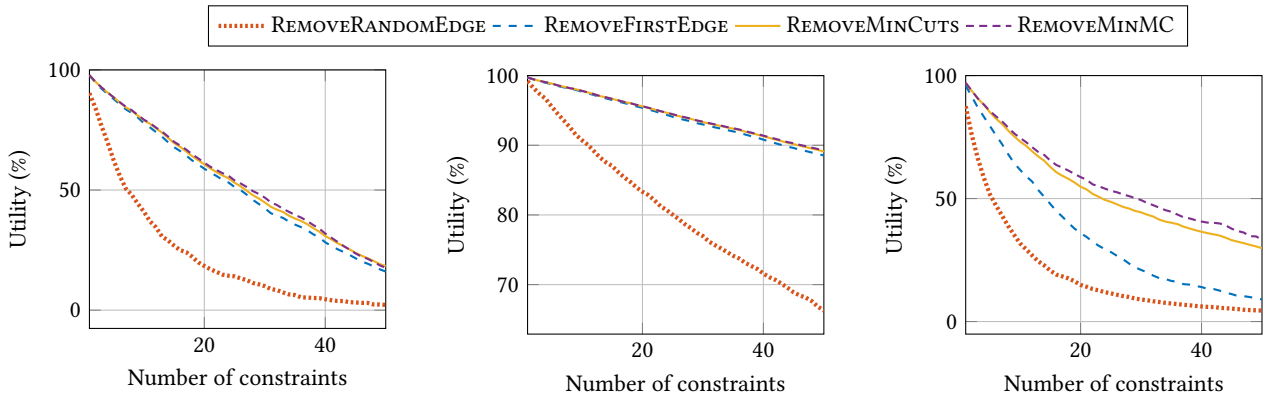


(a) Graphs with 100 vertices (dataset 1a).

(b) Graphs with 1000 vertices (dataset 1b).

(c) Dense graphs (dataset 1c).

Figure 5: The number of constraints vs. the runtime of the algorithms in graphs from dataset 1.



(a) Graphs with 100 vertices (dataset 1a).

(b) Graphs with 1000 vertices (dataset 1b).

(c) Dense graphs (dataset 1c).

Figure 6: The number of constraints vs. graph utility after applying the algorithms on graphs from dataset 1.

see that the utility decreases as the number of paths connecting constraints increases. Yet again, the utility after executing RemoveMinMC tends to decrease the slowest, reaching on average the utility of 32.27% of the original utility for the graph with 822 paths to be broken. For the same graph, the next best solution is RemoveMinCuts with an accuracy of 24.58%. For comparison, RemoveFirstEdge achieves on average a utility of 15.73% .

7.2.3 Number of data processing stages. Next, we apply the algorithms to graphs with a constant number of paths. Since, in the linear model, only edges connected to the purpose vertices affect the utility of the graph, increasing the lengths of the paths on its own does not affect the utility. Thus, in this experiment, we focus on the execution time of the algorithms as the length of the paths grows. In Figure 8, we consider sparse graphs with vertices distributed uniformly with the same number of user data vertices as purpose vertices (dataset 2). We can see that as the path length grows, the runtime in case of BruteForce increases faster than the others.

7.2.4 Workflow size. Lastly, we analyse how the number of vertices in the graph impacts the runtime and the utility of the graph after applying the algorithms. To do this, we run the algorithms on sparse graphs of sizes between 100 and 10000 vertices and corresponding sets of 10 constraints (dataset 3). As the number of paths between the constraints and their length are equal for these graphs, in Figure 9 we can see that the size of the graph has only a slight impact on the execution time for BruteForce. In addition, RemoveMinCuts is faster on average compared to BruteForce and RemoveMinMC. Considering the utility, Figure 9 shows that the graph size does not have a significant impact on the utility when the number of paths between the constraints and their length remain equal for the graphs.

8 OPEN PROBLEMS

We designed a theoretical mechanism where the data workflow is structured as a graph and privacy constraints collected from the user point to pairs of vertices in this workflow graph. When all of these pairs are disconnected, we have a Consented Data Workflow. We are currently in the process of integrating our algorithms with the more fine-grained system developed in [18]. Interestingly, there is an extensive collection of open problems and challenges around consented data workflows.

First, our theoretical results show that the problem in general is NP-hard. This result is provides us with the lower bound on its complexity. The upper bound, however, depends largely on the complexity of the selected valuation and utility functions. Future work should investigate the upper bound of the problem with different functions that depend on the application.

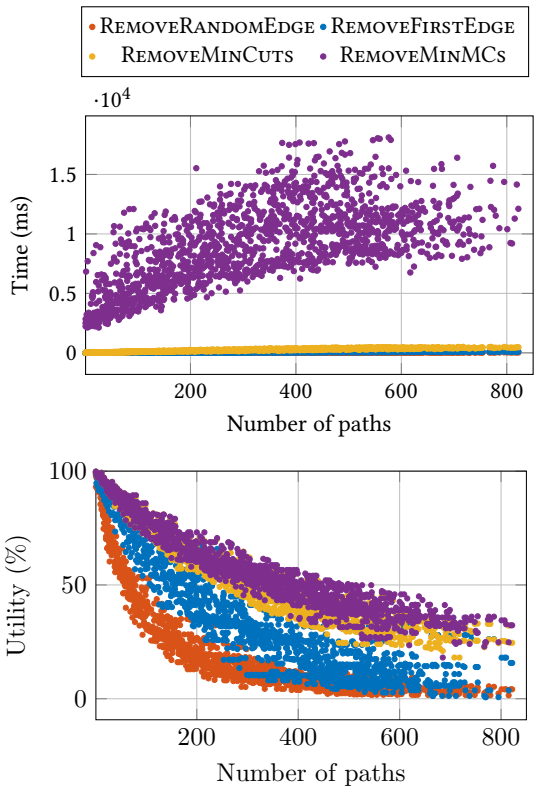


Figure 7: No. of paths vs. runtime and utility (dataset 1c).

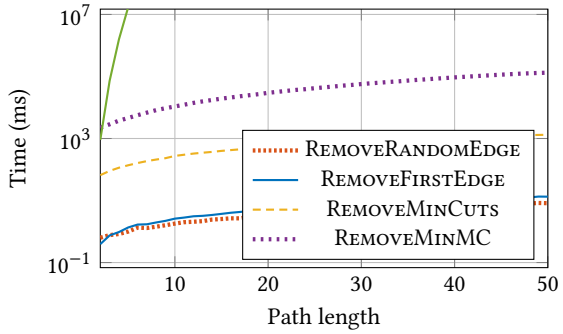


Figure 8: Path length vs. time in sparse graphs (dataset 2).

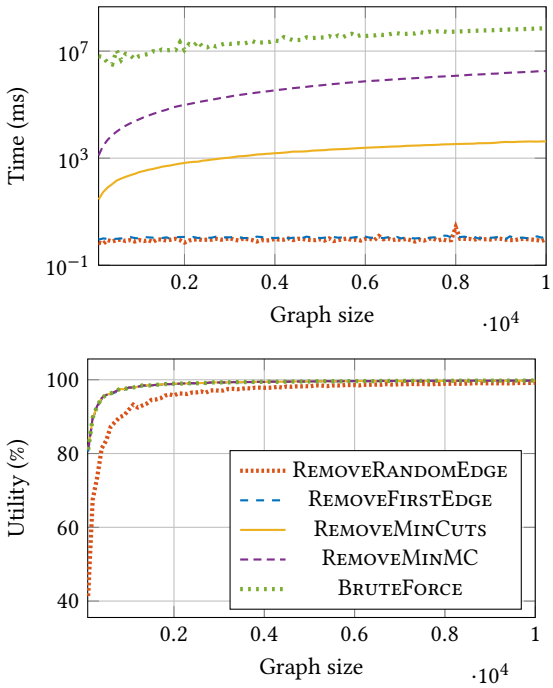


Figure 9: Graph size vs. runtime and utility (dataset 3).

Second, we focused on a specific instance of the problem, where the importance of an outputted data type is linearly additive with respect to the input. In addition, we assumed that the utility functions of specific data processing purposes are also linearly additive with respect to the importance of the data processed for the purpose. For this instance of our problem, our algorithm RemoveMinMC can find very accurate approximate solutions in seconds even for large workflow graphs. Nonetheless, the complexity of the problem in this additive case remains unknown. Further investigation is needed not just to study even more efficient and optimal algorithms, but also to establish the bounds on its complexity.

Third, some of our heuristic algorithms rely on the simplifying assumption that the value of different information sources is additive. While this can be a reasonable approximation in some settings, in practice the value may be subadditive (e.g. in the case of redundant data) or superadditive (when data complements each other). At the moment, finding realistic data to create large real-world models and design algorithms for them is challenging. However, as data processing systems keep expanding, future work should focus on more realistic workflow models.

Note that in certain machine learning as well as statistical algorithms there is already work that addresses masking or distorting the input with noise. This gives rise to an exciting generalisation of our framework, where utility is affected not by removing the input edge but by distorting the input, and in the future we plan to explore this.

In addition, there are several open problems regarding scalability of our solution. Currently, the solution needs to be recomputed every time a new user enters the system or when an existing user updates their constraints. What is more, every time a change is made, some of algorithms that process the data would need to be re-run as well, which could be costly. At the same time, there could be many users of the same type, i.e. with similar privacy constraints, and a limited number of different user types which can be known in advance. To take advantage of this, users of the same type could e.g. be treated as a single user to enable the system to cope with thousands and even millions of users. This way, if new users enter the system, a new solution can be found quickly. Generally, as there are more and more users with privacy constraints, new methods are needed that take into account scalability by re-using some of the computation performed for the previous solution, as well as the costs of making changes.

Finally, there are plenty of opportunities to consider richer types of privacy constraints and user preferences. For example, users may have constraints on combinations of different data types for a specific purpose (e.g. a user may say '*I'm okay with you using my data for advertising, but don't combine my location with my purchase history*'), processing the data types by specific service providers (e.g. '*I'm okay with you sharing my purchase*

history with anyone but Nile') or time restrictions on data processing (*'I'm ok with you sharing my purchase history with Nile, but I don't want them to keep it for more than 30 days*'). For such constraints, future work should formulate new problems around consented data workflows.

9 RELATED WORK

With the scale of data collection and processing growing vastly in the recent years, there has been an emergence of initiatives and tools that aim to aid users in controlling the flow of their personal data. Early efforts include the Platform for Privacy Preferences (P3P) which aimed to enable machine-readable privacy policies [8]. Such privacy policies could be automatically retrieved by Web browsers and other tools that can display symbols, prompt users, or take other appropriate actions. Users were able to communicate their privacy constraints to these so-called *user agents* as a list of rules expressed in a P3P Preference Exchange Language (APPEL) [8]. The agents were then able to compare each policy against the user's constraints and assist the user in deciding when to exchange data with websites [8].

However, the P3P lacked a mechanism that would allow for enforcement of the privacy policy within the enterprise and for management of users' individual privacy preferences [3, 6, 17]. Thus, a new approach was proposed [6] where service providers would publish privacy policies, collect and manage user preferences and consent, and enforce the policies throughout their systems. Based on this framework, the Platform for Enterprise Privacy Practices (E-P3P) was developed [5, 16]. Our work builds on this idea and proposes how to satisfy the users' individual preferences *optimally*.

Nonetheless, more general approaches that go beyond the P3P were required to address the need for policy enforcement. To that end, Hippocratic databases were proposed [1–4, 19] as 'database systems that take responsibility for the privacy of data they manage'. In Hippocratic databases, personal data was required to be associated with the purposes it was collected for, as well as other metadata such as its retention period and information about who it can be given out to. Given the user's privacy constraints, the so-called Privacy Constraint Validator would check whether the service provider's privacy policy is acceptable to the user. Recently an approach to enable even more fine-grained control of such policies within relational databases was presented [18]. Here, we extend this approach by specifying we can optimise constraint satisfaction *across entire data workflows*.

Complementing our work, related work has proposed methods for privacy policy-compliant data processing. For example, once it is known how the user's privacy constraints can be optimally satisfied (e.g., using our proposed approach), it is possible to can ensure that the datasets used by the data-processing algorithms align with these constraints [9, 10]. Moreover, data processing techniques can be selected based on the consented types of the input data [24]. Furthermore, to make sure that the policies are enforced, a system for checking data usage policies automatically at query runtime has been proposed [23].

10 ACKNOWLEDGEMENTS

E. Gerding was partially funded by the EPSRC-funded platform grant "AutoTrust: Designing a Human-Centred Trusted, Secure, Intelligent and Usable Internet of Vehicles" (EP/R029563/1). G. Konstantinidis was partially funded by the UKRI Horizon Europe guarantee funding scheme for the Horizon Europe projects RAISE (101058479) and UPCAST (101093216).

REFERENCES

[1] Rakesh Agrawal, Roberto Bayardo, Christos Faloutsos, Jerry Kiernan, Ralf Rantzau, and Ramakrishnan Srikant. 2004. Auditing compliance with a hippocratic database. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 516–527.

[2] Rakesh Agrawal, Paul Bird, Tyrone Grandison, Jerry Kiernan, Scott Logan, and Walid Rjaibi. 2005. Extending relational database systems to automatically enforce privacy policies. In *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 1013–1022.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2002. Hippocratic databases. In *Proceedings of the 28th VLDB Conference*. VLDB Endownment, United States, 143–154.

[4] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2003. Implementing P3P using database technology. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*. IEEE, 595–606.

[5] Paul Ashley, Satoshi Hada, Günter Karjoth, and Matthias Schunter. 2002. E-P3P privacy policies and privacy authorization. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*. ACM, 103–109.

[6] Paul Ashley, Calvin Powers, and Matthias Schunter. 2002. From privacy promises to privacy management: a new approach for enforcing privacy throughout an enterprise. In *Proceedings of the 2002 Workshop on New Security Paradigms*. ACM, 43–50.

[7] Cédric Bentz. 2011. On the hardness of finding near-optimal multicuts in directed acyclic graphs. *Theoretical computer science* 412, 39 (2011), 5325–5332.

[8] Lorrie Faith Cranor. 2002. *Web privacy with P3P*. O'Reilly Media, Inc.

[9] Christophe Debruyne, Harshvardhan J Pandit, Dave Lewis, and Declan O'Sullivan. 2019. Towards generating policy-compliant datasets. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*. IEEE, 199–203.

[10] Christophe Debruyne, Harshvardhan J. Pandit, Dave Lewis, and Declan O'Sullivan. 2020. "Just-in-time" generation of datasets by considering structured representations of given consent for GDPR compliance. *Knowledge and Information Systems* 62, 9 (2020), 3615–3640.

[11] Yefim Dinitz. 2006. Dinitz'algorithm: The original version and Even's version. In *Theoretical Computer Science: Essays in Memory of Shimon Even*, Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman (Eds.). Springer, 218–240.

[12] Jack Edmonds and Richard M. Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* 19, 2 (1972), 248–264.

[13] European Parliament and the Council. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union* (4 May 2016), 1–88.

[14] Amirata Ghorbani and James Zou. 2019. Data Shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*. PMLR, Long Beach, California, USA, 2242–2251.

[15] Judd Randolph Heckman, Erin Laurel Boehmer, Elizabeth Hope Peters, Milad Davaloo, and Nikhil Gopinath Kurup. 2015. A pricing model for data markets. *iConference 2015 Proceedings* (2015).

[16] Günter Karjoth, Matthias Schunter, and Michael Waidner. 2002. Platform for enterprise privacy practices: Privacy-enabled management of customer data. In *2nd Workshop on Privacy-Enhancing Technologies. Lecture Notes in Computer Science*. Springer, 69–84.

[17] James H Kaufman, Stefan Edlund, Daniel A Ford, and Calvin Powers. 2002. The social contract core. In *Proceedings of the 11th International World Wide Web Conference (WWW)*. ACM, Honolulu, Hawaii, 210–220.

[18] George Konstantinidis, Jet Holt, and Adriane Chapman. 2021. Enabling Personal Consent in Databases. *Proc. VLDB Endow.* 15, 2 (oct 2021), 375–387. https://doi.org/10.14778/3489496.3489516

[19] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. 2004. Limiting disclosure in Hippocratic databases. In *Proceedings of the 30th International Conference on Very Large Databases*. VLDB Endowment, 108–119.

[20] Chao Li, Daniel Yang Li, Gerome Miklau, and Dan Suciu. 2014. A theory of pricing private data. *ACM Transactions on Database Systems* 39, 4 (2014), 1–28.

[21] R. Timothy Marler and Jasbir S. Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization* 26, 6 (2004), 369–395.

[22] Steve Pettifer, Jon Ison, Matï¿ 1/2š Kalaš, Dave Thorne, Philip McDermott, Inge Jonassen, Ali Liaquat, Josï¿ 1/2 M Fernï¿ 1/2ndez, Jose M Rodriguez, INB Partners, et al. 2010. The EMBRACE web service collection. *Nucleic acids research* 38, suppl_2 (2010), W683–W688.

[23] Prasang Upadhyaya, Magdalena Balazinska, and Dan Suciu. 2015. Automatic enforcement of data use policies with DataLawyer. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 213–225.

[24] Yang Wang and Alfred Kobsa. 2007. Respecting users' individual privacy constraints in web personalization. In *International Conference on User Modeling*. Springer, 157–166.