Privacy-Preserving Implementation of Local Search Algorithms for Collaboratively Solving Assignment Problems in Time-Critical Contexts

Kevin Schuetz
Johannes Kepler University Linz
Linz, Austria
ORCID: 0000-0003-1569-0342

Christoph G. Schuetz
Johannes Kepler University Linz
Linz, Austria
ORCID: 0000-0002-0955-8647

Samuel Jaburek
Johannes Kepler University Linz
Linz, Austria
jaburek@dke.uni-linz.ac.at

***Abstract*—Solving real-world optimization problems often requires collaboration among multiple stakeholders. In air traffic flow management, for example, airlines must work together to prioritize individual flights in cases of reduced capacity in the air traffic network. However, when diverse parties are required to share sensitive information to collaboratively conduct optimization, trust becomes an issue. To alleviate those issues, privacy-preserving computation can be utilized to protect the confidential information of participants, which comes with a trade-off in terms of runtime performance. In time-critical contexts, privacy-preserving implementations of deterministic optimization algorithms may not be able to produce a result before the deadline. In this paper, we investigate the effectiveness of using variants of local search algorithms for the search of solutions to an optimization problem in conjunction with multi-party computation for the evaluation of those solutions. We argue that the proposed method using local search algorithms achieves good results in terms of the quality of the found solution while considerably reducing the run time with respect to a privacy-preserving deterministic solution.**

***Index Terms*—combinatorial optimization, heuristics, multi-party computation**

I. Introduction

Conducting optimization often requires collaboration among multiple stakeholders who must provide inputs for the optimization algorithm. For example, in supply chain planning, suppliers, manufacturers, logistics providers, and customers must collaborate to improve business performance [1]. In air traffic flow management (ATFM), airlines must collaborate when prioritizing flights to minimize delay costs in cases of reduced capacity in the air traffic network [2], which is an example of the *linear assignment problem*.

When diverse parties with vested interests are required to share sensitive information, e.g., costs, to collaboratively conduct optimization, trust becomes an issue. Participants in the optimization often want to conceal their preferences from other participants. If a third party provides a platform for conducting the optimization, sensitive information should be protected from an honest-but-curious platform provider or any intruders to the system.

979-8-3503-1458-8/23/$31.00 ©2023 IEEE

Privacy-preserving computation that protects the confidential information of participants may alleviate trust issues when collaboratively conducting optimization. Multi-party computation with secret sharing, for example, distributes the computation to multiple servers, where no server alone can decrypt the private inputs, and no trusted party conducts the optimization. Privacy protection, however, comes with a trade-off in terms of runtime performance.

In time-critical contexts, privacy-preserving implementations of deterministic optimization algorithms may not arrive at a solution within the deadline. In such contexts, the "correctness of a computation depends not only on the logical correctness but also on the time at which the results are produced" [3, p. 6]. Thus, a "software routine implementing the functionality of a task should complete its execution before the task deadline" [4, p. 8]. For example, in ATFM, flight prioritization in situations of reduced capacity must be finished within a relatively narrow time window [5] in order to allow for the operational changes to take effect and a compensation mechanism to be executed before the first flight's departure/arrival time [6].

In contrast to deterministic algorithms, evolutionary optimization algorithms can be terminated anytime and still yield a valid result, which makes them an ideal choice in a time-critical setting. Furthermore, previous work has demonstrated the relative efficiency of adopting a distributed architecture for privacy-preserving optimization [5] that employs evolutionary algorithms (in form of genetic algorithms) for the search of candidate solutions and MPC protocols for the evaluation of those candidate solutions when compared to an MPC implementation of deterministic algorithms [7].

In this paper, we investigate the potential of using local search algorithms for solving linear assignment problems in a privacy-preserving manner, building on the distributed architecture proposed by Schuetz *et al.* [5]. Local search algorithms employ an iterative approach to improve an existing solution by comparing the solution's quality with that of neighboring solutions, which are defined as solutions that differ only slightly from the existing solution [8]. In particular, we investigate the use of different variants of well-known

local search algorithms, which can be considered a kind of evolutionary optimization algorithm (cf. Simon [9]), instead of genetic algorithms for the search of candidate solutions. We show how to adapt existing local search algorithms and investigate the performance of those algorithms when using limited information regarding the fitness of the found solutions. We conduct experiments using realistic datasets from a real-world use case in ATFM [2].

The remainder of this paper is organized as follows. Section II provides background information regarding a real-world application in ATFM and relevant concepts in multi-party computation; Section II-C reviews related work. Section III introduces the framework for the privacy-preserving optimization with local search algorithms. Section IV describes the experimental setup, Section V the experimental results. Section VI discusses the framework and the experimental results. Section VII concludes the paper.

II. Background

In this section, we describe a real-world application of privacy-preserving optimization of assignments in the domain of air traffic flow management (ATFM) before summarizing the fundamentals of multi-party computation. We also review related work.

A. Real-World Application in Air Traffic Flow Management

Temporarily reduced capacity in the air traffic network and the ensuing congestion at airports are a major reason for flight delays, which result in additional costs for the operating airlines in terms of both cash flows and reputation with passengers. Different flights incur different (amounts of) costs when experiencing delay. Some flights can more easily tolerate additional delay, whereas other flights have more critical delay targets. Consequently, cost savings could be achieved by prioritizing flights according to their individual cost functions. To achieve the best result, such optimizations should be conducted across airlines, with each airline sharing its preference regarding the optimization. Airlines, however, are typically wary of sharing their preferences, which may allow to infer conclusions about an airline's cost structures, which in turn is confidential information for the airlines. We refer to Schuetz *et al.* [5], Gringinger *et al.* [6], and Schuetz *et al.* [2] for further information.

The SlotMachine project [10] aimed to develop a platform for privacy-preserving optimization of flight lists; we build on that project's proposed distributed architecture for our implementation and use that project's published datasets for our experiments. The optimization of flight lists can be seen as an assignment problem, where a mapping between two sets of items—flights and slots—must be found. If there are as many slots as flights—the SlotMachine project considered this case— the problem is a balanced linear assignment problem (see Section III-A for a more formal definition). The optimization may be coupled with a market mechanism that serves to promote equity and fairness regarding slot assignment over time [11]. In the SlotMachine project, airlines state their preferences regarding the optimization in terms of a preferred slot and "margins", i.e., the earliest slot time and the latest slot time to be assigned (see [2] for more information). Those preferences are then translated into a weight map.

B. Multi-Party Computation

Privcacy-preserving computation performs computations in a way that protects the confidentiality of private input data without the need of a trusted third party. Different techniques enable privacy-preserving computations, e.g., MPC with secret sharing, homomorphic encryption, and zero-knowledge proofs. We refer to Cramer *et al.* [12] for a comprehensive introduction into the topic. According to Zhao *et al.* [13], MPC allows distributed computation of arbitrary functionality without requiring the participating parties to reveal their private inputs or outputs. Loruenser *et al.* [7] employ MPC and zero-knowledge proofs as a facilitator to guarantee that no central trusted authority is required and that the output is publicly verifiable. MPC comes with a certain performance penalty, which means that there is a trade-off between confidentiality and performance. The calculations are performed by MPC nodes, with each node being under the control of a different participant. Each MPC node receives a *share* of the inputs. Loruenser *et al.* [7] also encrypt each share using the public key of the MPC node that receives that share.

C. Related Work

Sakuma *et al.* [14] propose an MPC protocol for privacy-preserving combinatorial optimization using local search and genetic algorithms, the distributed traveling salesman problem being an example. Han *et al.* [15] use a similar architecture, combining MPC with genetic algorithms to facilitate rule discovery in data mining. Both works disclose the results of the cost function—Sakuma *et al.* [14] reveal the relative ordering of the individuals of a population whereas Han *et al.* [15] reveal absolute fitness values.

Liu *et al.* [16] emphasize that attackers in the context of optimization are interested in relative rankings rather than exact objective values. Therefore, Liu *et al.* [16] propose a federated data-driven optimization framework based on the Diffie-Hellman-assisted secure aggregation. Funke *et al.* [17] also highlighted that not only cost function evaluation has the potential to leak private input data, but that the selection of individuals from the population also threatens the confidentiality of the input data. Funke *et al.* [17] propose a technique based on additive-homomorphic encryption that does not reveal intermediate results of a privacy-preserving genetic algorithm. However, Schuetz *et al.* [5] highlight that the presented results are not immediately transferrable to the assignment problem or any other optimization problem.

Schuetz *et al.* [5] propose a system that evaluates the objective function of a genetic algorithm using MPC to solve the assignment problem of optimizing flight sequences in ATFM, which could also be adapted to other assignment problems in other domains. Schuetz *et al.* [5] propose different obfuscation methods to disguise actual fitness values. In our paper, by

switching from genetic algorithms to local search algorithms, we propose an improvement to this system that enhances security of private input data and performance, leaking less information regarding the dominance of individuals when conducting an optimization.

III. Framework

We now define the optimization problem and describe the general framework for privacy-preserving optimization. We then discuss methods for obfuscation of fitness, the specific local search algorithms that were considered in this paper, and the implementation of those algorithms.

A. Problem Formulation and Execution Model

Let us first define the notion of assignment and the corresponding optimization problem; the following definitions regarding assignments and assignment problems are based on Burkard *et al.* [18]. An assignment maps n items of one kind (e.g., flights) to n items of another kind (e.g., departure slots). In terms of graph theory, given a bipartite graph $G = (U, V; E)$, where the disjoint sets of vertices U and V represent the items to be mapped, and the set of edges $E \subseteq U \times V$ defines possible mappings between elements from U and V, an assignment is a *matching* $M \subseteq E$, which maps every $u \in U$ to exactly one $v \in V$ and no $v \in V$ to more than one $u \in U$. In a *balanced assignment problem*, which we consider in the following, the number of items in U is equal to the number of items in V, i.e., $|U| = |V|$, and an assignment then is a *perfect matching* $M \subseteq E$. An injective *weight function* $c : E \to \mathbb{R}$ then defines a weight (e.g., costs or utility) for each possible mapping, which allows to compare the *fitness* of different assignments.

An alternative representation of an assignment is the *permutation matrix* [18, p. 1], which lends itself to the implementation using MPC [7]. Thus, in an $n \times n$ permutation matrix $X = (x_{ij})$ for a balanced assignment problem where $|U| = |V| = n$, and the sets U and V are ordered, the element $x_{ij} = 1$ if, and only if, the i-th item of the set U maps to the j-th item of the set V, and otherwise $x_{ij} = 0$, for all $i, j \in \{1, \ldots, n\}$. The sum of each row in the permutation matrix is 1. The weight function c then becomes a *weight matrix* $C = (c_{ij})$, where each row comprises the weights of assigning the i-th item of the set U to the j-th item of the set V, for all $i, j \in \{1, \ldots, n\}$.

An optimization problem requires an objective function. Given an $n \times n$ permutation matrix $X = (x_{ij})$ and an $n \times n$ weight matrix $C = (c_{ij})$, where each c_{ij} represents the utility of the corresponding x_{ij}, the following objective function characterizes a *linear sum assignment problem* aiming to maximize the utility (fitness) of the assignment.

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} c_{ij}$$

In a collaborative setting, k participants work together to solve an optimization problem characterized by $G = (U, V; E)$, where each participant controls a set of items $U_i \subseteq U$, for every $i \in \{1, \ldots, k\}$, the sets $U_1, \ldots, U_k$ being pairwise disjoint. Each participant then contributes one or more rows of the weight matrix C as input to the optimization algorithm, indicating the respective participant's preferences regarding the optimization.

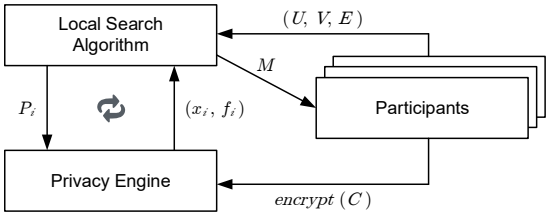

Fig. 1. Framework for privacy-preserving optimization

To preserve the confidentiality of the participant's preferences regarding the optimization, each participant encrypts their contributed rows of the weight matrix C, and the optimization algorithm employs a *Privacy Engine* to compute the fitness of different assignments in a privacy-preserving manner. Technology-wise, such privacy-preserving computation can be enabled via *homomorphic encryption*, which allows to perform computations over ciphertexts [19], or via secret sharing and MPC protocols (see Section II-B). Using MPC, each participant *secret-shares* their contributed rows of the weight matrix C, obtaining m shares of each row with the same number of columns as C, one share for each of the m MPC nodes actually conducting the computations. Each of those shares is also encrypted with the public key of the MPC node that receives the respective share. The Privacy Engine only coordinates the computations conducted by the MPC nodes, which are controlled by different entities and cannot decrypt the private inputs submitted by the participants.

Figure 1 illustrates the generic framework for privacy-preserving optimization using local search algorithms, based on the distributed architecture proposed by Schuetz *et al.* [5], which employs genetic algorithms in conjunction with MPC. The local search algorithm looks for candidate solutions, receiving public information regarding the items to be mapped (U and V) and any constraints over non-private inputs, which are formally represented by the set of possible mappings E, from the participants in the optimization run. The encrypted preferences for the optimization, i.e., the secret-shared weight matrix C, are used for the privacy-preserving evaluation of the fitness of the submitted solutions, which the *Privacy Engine* orchestrates. The participants receive as the result from the local search algorithm the assignment (M). We refer to Schuetz *et al.* [5] as well as Loruenser *et al.* [7] for more information regarding the implementation of the Privacy Engine using MPC and focus on the local search aspects in the following.

The actual optimization is an iterative process conducted by a local search algorithm in conjunction with the Privacy Engine, thus separating the search for candidate solutions from the evaluation of the fitness of those candidate solutions. In

each iteration, the local search algorithm submits a *population* of candidate solutions to the Privacy Engine for evaluation and receives a selected solution and a fitness value as the result. The selected solution and the fitness value are the basis for generating an altered population in the next iteration.

Algorithm 1 illustrates the general principle of local search algorithms for privacy-preserving optimization, the notation following the example of Simon [9]. The requires the set of possible mappings between U and V as well as a *neighborhood function* $h(x) : E \rightarrow \mathcal{P}(E)$ as input, with $\mathcal{P}(E)$ the set of all subsets of E. Given a candidate solution x (an assignment), $h(x)$ returns the set of other solutions that are similar to x, i.e., the neighborhood of x. The neighborhood of x consists of all solutions that can be obtained from x via a local transformation, which in the current implementation are those solutions that can be obtained from x by swapping the mapping of two elements from U, similar to a 2-*Exchange* (or 2-*Opt*) neighborhood for the traveling salesman problem [20]. The algorithm first initializes a solution x_0, which can be the original solution (if available), a randomly generated solution, or a solution generated using construction heuristics. This solution x_0 becomes the value of x, with f set to $-\infty$. Then, until a termination criterion is fulfilled, in each step i, the algorithm selects a set P_i from the neighborhood of x, submits that set to the Privacy Engine for evaluation, and receives one solution x_i from P_i along with a fitness value f_i. From an optimization perspective, the returned x_i ideally is the solution with the best fitness in the neighborhood and f_i that solution's actual fitness. Depending on the obfuscation method—aimed at reducing leakage of information regarding private inputs (see Section III-B)—x_i may only be *among the best solutions* in the neighborhood, and f_i may only be the average fitness of the best solutions. Based on f_i and the current value of f, the local search algorithm decides whether to update x with x_i and f with f_i for the next iteration step; otherwise, x remains unchanged and another set P_{i+1} from the neighborhood $h(x)$ is selected in the next step. When the termination criterion is fulfilled the algorithm returns the encountered solution x_j where the corresponding f_j was maximal.

Algorithm 1: Generic local search algorithm

```
E = the set of possible mappings between U and V
h(x) = a neighborhood function: h(x) : E → P(E)
Initialize a candidate solution x_0 ⊆ E
x ← x_0
f ← −∞
i ← 1
while not(termination criterion) do
    Select a set P_i of candidate solutions from the
      neighborhood h(x)
    Obtain a solution x_i ∈ P_i and a corresponding
      fitness value f_i from the Privacy Engine
    if x_i is accepted given f_i and f then
        x ← x_i
        f ← f_i
    end
    i ← i + 1
end
return solution x_j ∈ {x_1, ..., x_n} where f_j is max
```

B. Obfuscation Methods

To enhance the protection of private inputs, various obfuscation methods can be employed to conceal the true fitness value and the dominance of a solution—i.e., the relative fitness of a solution with respect to other solutions in a population—returned by the Privacy Engine. The obfuscation method determines the process for selecting the solution $x_i \in P_i$ returned by the Privacy Engine and obtaining the corresponding fitness value f_i. Figure 2 illustrates the principle of the different obfuscation methods.

The goal is to ensure that the local search algorithm is capable of finding good solutions while ensuring the confidentiality of sensitive input data by limiting the amount of information disclosed to the local search algorithm. The obfuscation method *best* serves as the baseline, revealing the most information regarding fitness and dominance of the solution x_i returned by the Privacy Engine: In each iteration, the Privacy Engine returns a solution $x_i \in P_i$ such that $\nexists x'_i \in P_i$ where $fitness(x'_i) > fitness(x_i)$, and the returned fitness value f_i is $fitness(x_i)$.

The obfuscation methods *top* and *above* conceal fitness value and dominance of the solution x_i returned by the Privacy Engine. Both methods construct a set $P'_i \subseteq P_i$ comprising the best solutions in a population P_i. For both methods, a threshold $0 < t < 1$ has to be defined, which determines the size of P'_i. For method *top*, t denotes the number of individuals relative to the size of P_i that are included in P'_i. For example, if $|P_i| = 100$ and $t = 0.05$ then P'_i consists of the five best solutions in P_i. For method *above*, t denotes the fitness threshold relative to the maximum fitness value in P_i that qualifies a candidate solution for inclusion in P'_i. For example, if the maximum fitness in a population P_i is 100 and $t = 0.95$ then $P'_i = \{x \in P_i : fitness(x) \geq 95\}$. For both methods, the condition $\forall x' \in P'_i : \nexists x \in P_i : x \notin P'_i \wedge fitness(x) > fitness(x')$ is true. An additional constraint for method *above* requires that P'_i is composed of at least three elements, which can be satisfied by dynamically lowering the threshold for an iteration if necessary.

Given a set P'_i constructed by the obfuscation methods *top* and *above*, the Privacy Engine returns a randomly chosen $x_i \in P'_i$ to the local search algorithm. The returned fitness value f_i is set to the average fitness values of the solutions in P'_i, i.e.,

$$f_i = \frac{1}{|P'_i|} \sum_{x \in P'i} fitness(x).$$

When using the obfuscation methods *top* and *above*, it is impossible to infer neither the actual fitness value of any given solution nor the dominance of one solution over another solution. Furthermore, given two distinct (but possibly overlapping) populations $P_i, P_j \subseteq E$ as well as the solutions

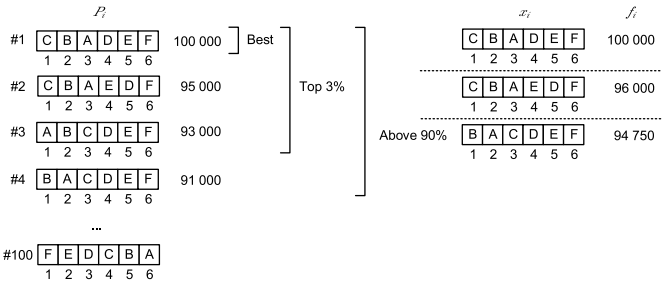

Fig. 2. Illustration of obfuscation methods. Note that rank and fitness values of the individual solutions in a population are known neither to the local search algorithm nor to the Privacy Engine but the computations are conducted by multiple MPC nodes together, no single node knowing the private inputs.

$x_i \in P_i$ and $x_j \in P_j$ returned by the Privacy Engine for the respective populations, dominance of x_i over x_j cannot be inferred with certainty even if $f_i > f_j$.

The choice of obfuscation method also influences the complexity of the MPC computations. Schuetz *et al.* [5] report that ranking candidate solutions in MPC is time-intensive since ranking requires many iterations and cannot be parallelized whereas collecting the candidate solutions into buckets based on fitness value is comparatively faster. Of the proposed obfuscation methods, only the method *top* requires ranking the solutions in a population. For methods *best* and *above*, finding out the maximum fitness and comparing each solution's individual fitness to that maximum fitness suffices.

C. Local Search Algorithms

We selected five popular local search algorithms and adapted them for use in the proposed framework. The algorithms follow a *best improvement* strategy as opposed to a *first improvement* strategy, i.e., the algorithm selects the best solution of the neighborhood—or, in our case, an approximation thereof—rather than the first solution to be an improvement [20, p. 193]. Algorithms following a *first improvement* strategy cannot be used in the proposed framework since the population P_i that is submitted to the Privacy Engine in each iteration must contain a minimum number of distinct candidate solutions to ensure the protection of privacy of the input data as well as to reduce the overhead caused by communication between local search algorithm and Privacy Engine.

In the following, we briefly introduce the implemented algorithms. The candidate solution and the fitness value returned by the Privacy Engine in any given iteration are denoted with x_i and f_i, respectively. Likewise, the current reference solution and the associated fitness value are denoted with x and f. For all algorithms, $f \leftarrow f_i$ if and only if $x \leftarrow x_i$. The algorithms differ with regard to the conditions for updating the current reference solution x.

1) Hill Climbing (HC): This algorithm updates the current solution x with the returned candidate solution x_i if and only if $f_i \geq f$. Therefore, HC would not even require disclosure of actual or obfuscated fitness values at all, as the disclosure of the result of the evaluation of the condition (a boolean expression) to the local search algorithm would suffice.

Algorithm 2: Simulated annealing

E = the set of possible mappings between U and V
$h(x)$ = a neighborhood function: $h(x) : E \to \mathcal{P}(E)$
$\alpha(t, i)$ = a cooling function: $\alpha(t, i) : \mathbb{R}_{>0} \times \mathbb{N} \to \mathbb{R}_{>0}$
t_0 = an initial temperature > 0
Initialize a candidate solution $x_0 \subseteq E$
$x \leftarrow x_0$
$f \leftarrow -\infty$
$t \leftarrow t_0$
$i \leftarrow 1$
while not(termination criterion) **do**
  Select a set P_i of candidate solutions from the neighborhood $h(x)$
  Obtain a solution $x_i \in P_i$ and a corresponding fitness value f_i from the Privacy Engine
  if $f_i \geq f$ **then**
    $x \leftarrow x_i$
    $f \leftarrow f_i$
  else
    Choose a random number $r \in [0, 1]$
    if $r < e^{-\frac{f - f_i}{t}}$ **then**
      $x \leftarrow x_i$
      $f \leftarrow f_i$
    end
  end
  $i \leftarrow i + 1$
  $t \leftarrow \alpha(t, i)$
end
return solution $x_j \in \{x_1, \ldots, x_n\}$ where f_j is max

2) Step Counting Hill Climbing (SCHC): This algorithm is a *threshold local search algorithm* [20, p. 433] where a candidate solution x_i is *also* accepted if $f_i \geq t$, where t denotes a threshold in terms of an absolute fitness value. Every n iterations, t is updated with the current value of f.

3) Great Deluge (GD): Another *threshold local search algorithm*, GD raises t every iteration by a fraction of the first fitness value returned by the Privacy Engine, i.e., f_1, unlike with SCHC, where t is updated periodically.

4) Simulated Annealing (SA): Algorithm 2 shows a basic algorithm for simulated annealing based on Simon [9, p. 226]. The algorithm requires as input an initial temperature t_0 and a cooling function $\alpha : \mathbb{R}_{>0} \times \mathbb{N} \to \mathbb{R}_{>0}$, which returns a new temperature depending on the current temperature t and iteration number i. If the fitness f_i returned by the Privacy Engine is less than the fitness of the current reference solution, the returned solution x_i may still be considered to replace the current reference soltuion: A random number $r \in [0, 1]$ is generated and compared to the degree of deterioration of the current temperature. After each iteration, t is lowered according to the cooling function.

5) Tabu Search (TS): Our TS implementation accepts any candidate solution x_i returned by the Privacy Engine as long as

$x_i \notin T$ or $f_i \geq f$, where T is the *tabu list*. After each iteration, the tabu list is expanded by the population, i.e., $T \leftarrow T \cup P_i$. If the size of T exceeds a configurable maximum size the oldest elements are removed from the list.

D. Implementation

We implemented the privacy-preserving variants of the described local search algorithms by modifying the source code of Version 8.14 of the OptaPlanner framework[1] [21]. OptaPlanner is a framework for constraint satisfaction solving using optimization algorithms to find solutions to planning problems, including assignment problems, but also other problems such as the traveling salesman problem and the knapsack problem, for example. OptaPlanner is written in Java and released under an open-source license; the development is sponsored by RedHat.

We adapted OptaPlanner by adding an interface for the evaluation of candidate solutions, which can be used to integrate the Privacy Engine into the optimization process[2]. The modified OptaPlanner library was integrated into the *Heuristic Optimizer* component of the SlotMachine platform [5], the source code of which is available online [23][3], providing a REST endpoint for initializing and starting optimizations. The PE was simulated for the experiments since the focus of this paper is on the evaluation of the performance of local search algorithms when using limited information regarding the fitness of candidate solutions.

IV. Experimental Setup

In the following, we describe the experimental setup for the investigation of the performance of local search algorithms. We first describe the datasets before presenting the different employed configurations of the algorithms and the metrics used to evaluate performance.

A. Datasets

For the evaluation of privacy-preserving local search algorithms, we employ datasets from the real-world use case of flight prioritization in ATFM (see Section II-A) published by the SlotMachine project [23]. Those datasets can be divided into three groups.

The first group comprises 27 datasets (Datasets 1–27) with 100 flights each, simulating different scenarios regarding the margin width, the concentration of the time wished, and the priorities. Those datasets were generated using the dataset generator published by the SlotMachine project. We refer to Schuetz *et al.* [25] for more detailed description of those scenarios.

The second group comprises 20 datasets (Datasets 28–47) that were generated based on preliminary samples of real-world preferences provided by an airline [25]. Among this group of datasets, there are datasets with 100 flights (Datasets 28-37) and 150 flights (Datasets 38-47). We employ the same datasets used by Schuetz *et al.* [5].

The last group of datasets comprises 10 datasets (Datasets 48-57), which were generated by the SlotMachine project based on more extensive, refined samples of real-world preferences provided by an airline [23].

B. Configurations

In addition to looking at the performance of different algorithms and fitness methods, we chose the size of the population as a parameter for the experiments. For every combination of algorithm and fitness method, one configuration with a small population size of 100, and another configuration with a larger population size of 250, were included in the experiments. Overall, we considered 30 configurations.

Due to the non-deterministic nature of some of the algorithms and fitness methods, we conducted three optimization runs of each configuration over each dataset. All experiments were configured to terminate after 500 iteration steps, or when the optimal solution, which was returned by the Hungarian algorithm, was found.

In the following, we describe the values of the parameters specific to the presented algorithms (see Section III-C) and obfuscation methods (see Section III-B), respectively.

For SCHC, n was 10, so that $t \leftarrow f$ every 10 iterations. For GD, the fraction of f_1 that t increases was 0.005, so that after every iteration $t \leftarrow t + (f_1 \times 0.005)$, where f_1 is the fitness value returned by the Privacy Engine in the first iteration. For SA, the starting temperature t_0 was 1000. For TS, the size of the tabu list was also 1000.

For the fitness method *above*, t was 0.995, considering that the observed range of fitness values of any given population P_i in preliminary tests tended to be quite narrow due to the similarity of the candidate solutions. For the fitness method *top*, t was 0.03.

C. Metrics

We were mainly interested in the fitness of the solution found after 500 iterations using a certain configuration of local search algorithm relative to the fitness of the optimal solution found by the deterministic *Hungarian algorithm* [26] for the same dataset. Furthermore, we were interested in the convergence of the local search algorithms. Thus, we looked at the relative fitness of the solutions returned in each iteration step to determine a number of iterations after which reasonably good solutions are found.

V. Experimental Results

In this section, we summarize the results of the performance experiments[4] introduced in Section IV. We present a comparison of the results regarding the investigated algorithms (see Section III-C), the fitness methods (see Section III-B), and the size of the population. Configurations of algorithm, obfuscation method and population size are denoted using the following schema: *(algorithm, obfuscation-method, population*

[1] https://www.optaplanner.org/

[2] The source code can be found online [22].

[3] The source code of the adapted component can also be found online [24].

[4] The datasets and experimental results can be found online [27].

size). For example, *(HC, best, 250)* denotes a configuration with the algorithm HC, the obfuscation method *best*, and a population size of 250. Whenever results are aggregated over different configurations, we use "_" as a wildcard. For example, *(HC,_,_)* means that the results are aggregated over all configurations that use the algorithm HC.

A. Performance

1) Algorithms: The experiments have shown that the choice of algorithm has a rather small effect on the quality of the solutions. The average relative fitness values over all datasets ranged from 94.30% for *(HC,_,_)* to 96.18% for *(SCHC,_,_)*, with an average over all datasets and configurations of 95.35%. Table I summarizes the results analyzed by algorithm.

2) Obfuscation methods: Regarding the employed obfuscation method, the results have shown that *(_,best,_)* on average finds solutions with considerably higher relative fitness values than the other methods, with an average relative fitness value of 99.17%. This result is expected, as *best* is the only fitness method that guarantees to disclose the candidate solution with the highest actual fitness value in every iteration (see Section III-B). However, although *best* discloses more information about dominance of candidate solutions, when paired with algorithms like HC or TS this obfuscation method could still be realized without revealing actual fitness values to the LSA. Table II summarizes the results by obfuscation method.

3) Population size: A larger population size tends to yield better solutions given after a fixed amount of iterations. However, the difference of the average relative fitness is less than 1% point, and a trade-off between the quality of the result solution and the run time has to be taken into account with regards to the cardinality of P. Table III shows the results.

Figure 3 plots the mean relative fitness value for each dataset per algorithm, obfuscation method, and population size, respectively. This also hints at the difficulty of the three different groups of datasets (see IV-A).

4) Outliers: Even with an average relative fitness over all datasets of 95.35%, one optimization that was executed as part of the experiments marked the minimum with a relative fitness value of 63.64%. This result was obtained in the first repetition of the Dataset 51 with the configuration *(SA, top, 250)*. However, the result does not seem to be due to the difficulty of this dataset, as the average relative fitness value for this dataset is 92.27%, with 22.81% of all datasets having an average relative fitness value below that fitness. Compared to that, *(SA, top, 250)* reached an average relative fitness over all repetitions for this dataset of 76.27%, with the maximum being 85.04%. The following analysis by configuration will confirm that especially the obfuscation method *top* combined with a population size of 250 lead to an increased dispersion of the achieved fitness values.

5) Variance: Table IV shows the average relative fitness values over all datasets for all 30 configurations. The best solutions were found with the configuration combining the algorithm TS with the fitness method *best* and a population size of 250. This is in line with the general observations presented in Section V-A.

TABLE I
MEAN AND MINIMUM RELATIVE FITNESS OF SOLUTIONS FOUND BY EACH ALGORITHM (IN % OF THE OPTIMAL SOLUTION'S FITNESS)

	Mean	**Min**
Hill Climbing (HC)	94.30	71.46
Step Counting Hill Climbing (SCHC)	96.18	80.70
Great Deluge (GD)	95.87	79.77
Simulated Annealing (SA)	94.57	63.64
Tabu Search (TS)	95.85	79.74

TABLE II
MEAN AND MINIMUM RELATIVE FITNESS OF SOLUTIONS FOUND USING DIFFERENT METHODS FOR FITNESS OBFUSCATION (IN % OF THE OPTIMAL SOLUTION'S FITNESS)

	Mean	**Min**
best	99.17	95.60
top	94.58	63.64
above	92.32	74.32

For real-world applications, consistency may be as important as average performance. The standard deviations of the average mean fitness values of the datasets for the obfuscation methods *best, above, and top*[5], are 0.0078, 0.0464 and 0.0445, respectively. The range of average relative fitness values of the configurations matching *(_, best, 250)* is from 99.34% to 99.47%. Combinations of the obfuscation method *best* with the smaller population size have a similar narrow range for the different algorithms. Looking at the other obfuscation methods, the picture is different. Average relative fitness values ranged from 90.17% to 92.03% for *(_, best, 125)*, and from 93.04% to 94.47% for *(_, above, 250)*. The range for the fitness method *top* is even broader. Average relative fitness values ranged from 92.96% to 96.19% for *(_, top, 125)*, and from 91.28% to 96.17% for *(_, top, 250)*. We assume that the increased range is due to the random selection mechanism and fitness obfuscation applied by the PE for these two fitness methods (see Section III-B). The obfuscation method *top* is the only one for which the range of mean relative fitness values increases with population size. Furthermore, the standard deviation of the mean relative fitness values of datasets increases slightly from 0.0426, when aggregated by *(_, top, 125)*, to 0.0467 for *(_, top, 250)*, whereas standard deviation drops for all other obfuscation methods. An increased population size even leads to worse results for some combinations including the fitness method *top*. Note that with an increased population size, the size of P' increases proportionally, and the stochastic element is amplified (see Section III-B).

When making similar comparisons regarding the algorithms, HC has the broadest range, from 90.17% for *(HC, above,*

[5]This means that the relative fitness values were aggregated for every dataset by *(_, method,_)*, where *method* is *best, above, or top*, respectively. Standard deviation was calculated based on the average relative fitness value for each dataset.

TABLE III
MEAN AND MINIMUM RELATIVE FITNESS OF SOLUTIONS FOUND USING DIFFERENT NEIGHBORHOOD SIZES (IN % OF THE OPTIMAL SOLUTION'S FITNESS)

	Mean	**Min**
250	95.77	63.64
100	94.94	74.32

TABLE IV
MEAN RELATIVE FITNESS OF SOLUTIONS PER COMBINATION OF ALGORITHM, OBFUSCATION METHOD, AND NEIGHBORHOOD SIZE (IN % OF THE OPTIMAL SOLUTION'S FITNESS)

	best		**above**		**top**	
	100	**250**	**100**	**250**	**100**	**250**
HC	99.03	99.34	90.17	93.04	92.96	91.28
SCHC	99.05	99.36	92.03	94.47	96.19	95.97
GD	98.82	99.46	91.29	93.72	95.89	96.02
SA	98.96	99.37	90.56	93.19	93.32	92.02
TS	98.82	99.47	91.10	93.62	95.92	96.17

125) to 99.34% for the configuration *(HC, best, 250)*. HC also has the highest standard deviation of average relative fitness values of the datasets (0.0413), when results are aggregated by *(HC,_,_)*. Combined with the same fitness methods and population sizes, the algorithm SCHC reached a minimum average relative fitness value of 92.03% for *(SCHC, above, 125)* and maximum of 99.36% for *(SCHC, best, 250)*, with the lowest standard deviation of the average relative fitness values of the datasets of 0.0272, when results are aggregated by *(SCHC,_,_)*.

Regarding population size, mean relative fitness values ranged from 90.17% to 99.05% for all configurations with smaller population and from 91.28% to 99.47% for the configurations with larger population. Standard deviations of average relative fitness values of the datasets are 0.0367 and 0.0292 for the smaller and larger population, respectively.

B. Convergence

We also analyzed the fitness evolution of the different configurations, primarily to determine how many iterations are required before optimizations converge to sufficiently optimal solutions. Figure 4 presents, for each configuration, the mean absolute fitness value in each iteration, and the mean theoretical maximum fitness over all datasets. This shows that near-optimal solutions were usually found after significantly fewer than the 500 iterations performed. As an example, the configuration (SCHC, *best*, 250) surpassed 95% of the theoretical maximum fitness on average after 158 iterations for the Datasets 28–57, and after only 25 iterations for the Datasets 1–27.

VI. DISCUSSION

Regarding the performance in terms of quality of the found solutions, the choice of local search algorithm does not seem to have a significant influence. Convergence was also not significantly affected by the choice of algorithm. On the other hand, a larger population size seems to lead to better results, although in this regard we have to note a performance penalty in terms of run time: Referring to MPC implementations for sorting and classifying solutions of an assignment problem in a population [7], we note that the larger the population size, the higher the run time. The disadvantage in run time, however, is offset by generally faster convergence: Fewer iterations are needed to arrive at the same result.

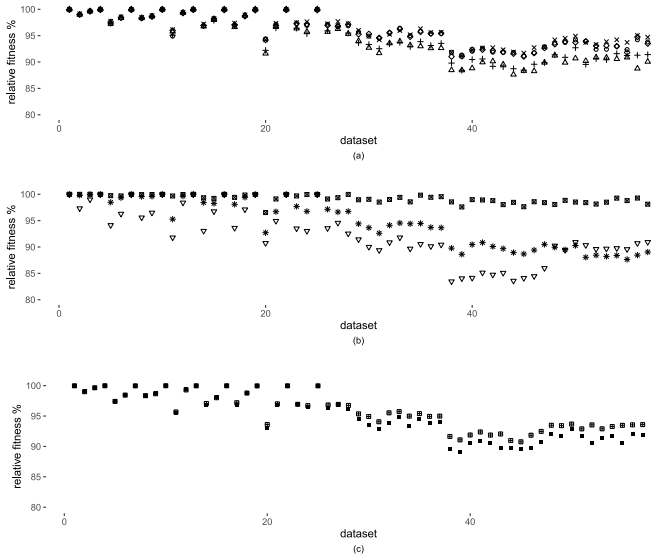

Fig. 3. The mean relative fitness value for each dataset by (a) algorithm (HC △, SCHC ×, GD ○, SA +, TS ◇), (b) obfuscation method (best ☒, above ▽, top ⋆), and (c) population size (100 ■, 250 ⊞).

When limiting the disclosure of information regarding fitness and dominance, the quality of the results tends to be lower when more randomness is involved in the selection of the solution to be returned by the Privacy Engine, i.e., when the P_i' selected by the obfuscation methods *top* and *above* comprises more solutions.

Compared to the privacy-preserving implementation based on GAs from the SlotMachine project [5], which requires disclosure of the dominance of multiple solutions, revealing only one solution per iteration, as in our approach with local search algorithms, is clearly preferable from a security point of view (cf. [17], [28]). Local search also seems to require fewer iterations to converge than a GA. Protection of private inputs could be further strengthened by including zero-knowledge proofs to allow for verification that the Privacy Engine indeed conducts optimization as specified (cf. [5], [7]). Furthermore, Schuetz *et al.* [5] propose a permissioned blockchain to store tamper-proof audit logs that could be opened in case there are doubts regarding the Privacy Engine's impartiality, which could be incorporated in our approach as well.

VII. SUMMARY AND FUTURE WORK

We demonstrated that common local search algorithms can be adapted for privacy-preserving optimization of assignment problems. To this end, we separate the search for solutions from the evaluation of those solutions, the evaluation being

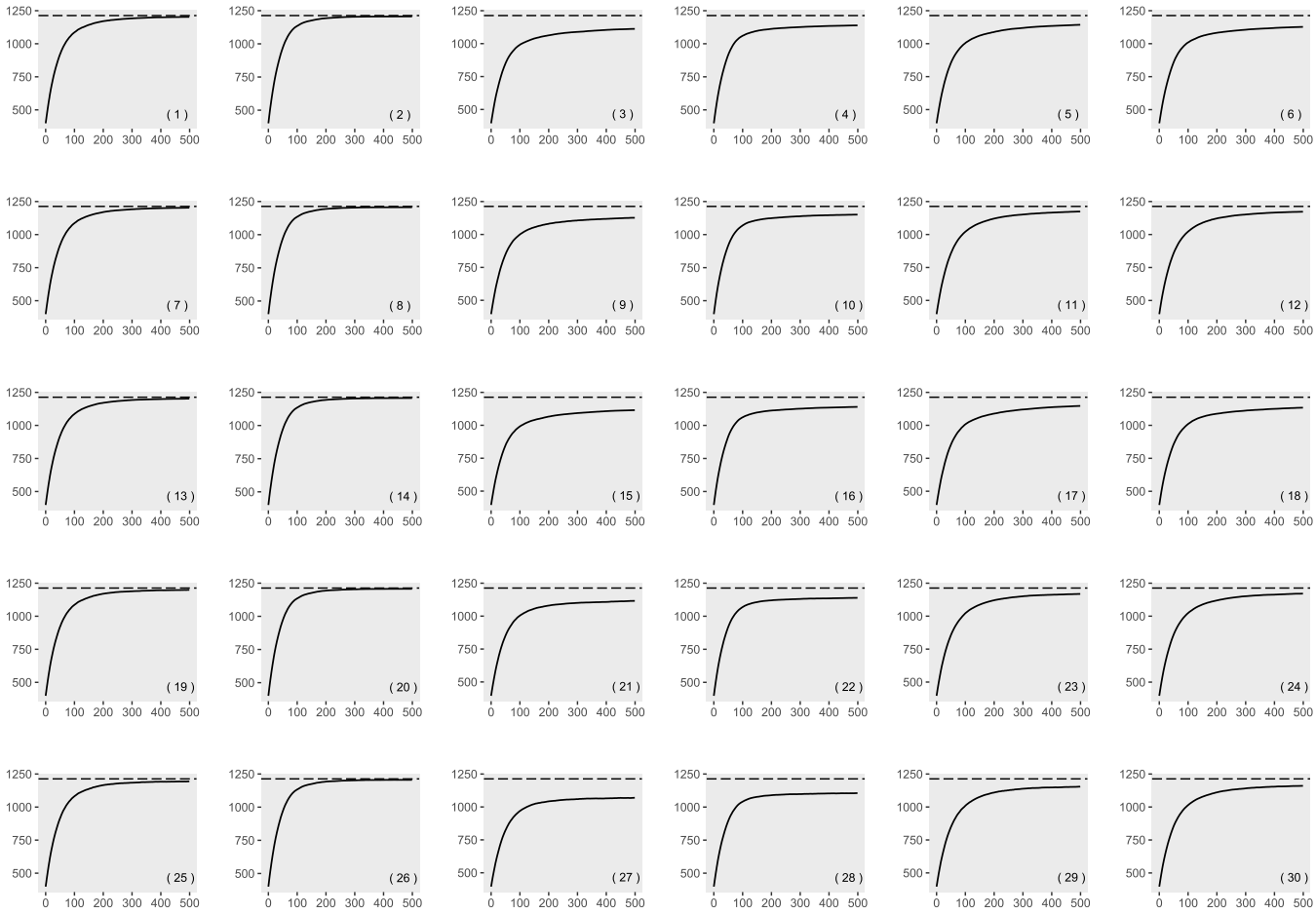

Fig. 4. A solid line indicates for each configuration over all datasets the mean fitness value of the solution found in each iteration. A dashed line indicates the mean fitness value of the solutions found by the Hungarian algorithm for all datasets. Each row shows the results of one algorithm (HC, SCHC, SA, GD, TS, in that order). The first two columns on the left are results of configurations with the obfuscation method *best*, the two middle columns with method *above*, and the two columns on the right with method *top*. Population size alternates by column, starting with the smaller population (100) in the first column, and the larger population (250) in the second column. In particular, the charts show results for the following configurations: (1) HC, best, 100; (2) HC, best, 250; (3) HC, above, 100; (4) HC, above, 250; (5) HC, top, 100; (6) HC, top, 250; (7) SCHC, best, 100; (8) SCHC, best, 250; (9) SCHC, above, 100; (10) SCHC, above, 250; (11) SCHC, top, 100; (12) SCHC, top, 250; (13) SA, best, 100; (14) SA, best, 250; (15) SA, above, 100; (16) SA, above, 250; (17) SA, top, 100; (18) SA, top, 250; (19) GD, best, 100; (20) GD, best, 250; (21) GD, above, 100; (22) GD, above, 250; (23) GD, top, 100; (24) GD, top, 250; (25) TS, best, 100; (26) TS, best, 250; (27) TS, above, 100; (28) TS, above, 250; (29) TS, top, 100; (30) TS, top, 250.

conducted by a Privacy Engine as proposed by Schuetz *et al.* [5]. Different obfuscation methods contribute to the protection of sensitive inputs by minimizing the information that is revealed. Future work will investigate applicability of the proposed framework to the generalized assignment problem [29] but also other optimization problems, e.g., traveling salesman problem or knapsack problem. Future work will also investigate the use of more modern techniques for the search of solutions, e.g., ant colony or particle swarm optimization. Furthermore, future work will implement the specific computations required for privacy-preserving evaluation of the fitness of solutions using MPC protocols. In this paper, we referred to MPC implementations for sorting and classifying solutions [7] to estimate the run time performance of the presented approach.

ACKNOWLEDGMENT

This work was conducted as part of the SlotMachine project. This project received funding from the SESAR Joint Undertaking under grant agreement No 890456 under the European Union's Horizon 2020 research and innovation program. The views expressed in this paper are those of the authors.

References

[1] A. Vereecke and S. Muylle, "Performance improvement through supply chain collaboration in europe," *International Journal of Operations & Production Management*, vol. 26, no. 11, pp. 1176–1198, 2006, DOI:10.1108/01443570610705818.

[2] C. G. Schuetz, E. Gringinger, N. Pilon, and T. Lorünser, "A privacy-preserving marketplace for air traffic flow management slot configuration," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, DOI: 10.1109/DASC52595.2021.9594401.

[3] K. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, 1994, DOI: 10.1109/5.259423.

[4] T. Mitra, J. Teich, and L. Thiele, "Time-critical systems design: A survey," *IEEE Design & Test*, vol. 35, no. 2, pp. 8–26, 2018, DOI: 10.1109/MDAT.2018.2794204.

[5] C. G. Schuetz, T. Lorünser, S. Jaburek, K. Schuetz, F. Wohner, R. Karl, and E. Gringinger, "A distributed architecture for privacy-preserving optimization using genetic algorithms and multi-party computation," in *CoopIS 2022*, ser. LNCS, vol. 13591. Springer, 2022, pp. 168–185, DOI: 10.1007/978-3-031-17834-4_10.

[6] E. Gringinger, S. Ruiz, and C. G. Schuetz, "Business and economic concepts for a privacy-preserving marketplace for atfm slots," in *2022 Integrated Communication, Navigation and Surveillance Conference (ICNS)*, 2022, DOI: 10.1109/ICNS54818.2022.9771484.

[7] T. Loruenser, F. Wohner, and S. Krenn, "A verifiable multiparty computation solver for the linear assignment problem: And applications to air traffic management," in *Proceedings of the 2022 on Cloud Computing Security Workshop (CCSW)*, 2022, pp. 41–51, DOI:10.1145/3560810.3564263.

[8] W. Michiels, E. Aarts, and J. Korst, *Theoretical Aspects of Local Search*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[9] D. Simon, *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

[10] CORDIS – EU Research Results: A privacy-preserving marketplace for slot management. [Online]. Available: https://doi.org/10.3030/890456

[11] C. G. Schuetz, S. Ruiz, E. Gringinger, C. Fabianek, and T. Loruenser, "An auction-based mechanism for a privacy-preserving marketplace for atfm slots," in *Proceedings of the 33rd Congress of the International Council of the Aeronautical Sciences*, 2022, https://www.icas.org/ICAS_ARCHIVE/ICAS2022/data/preview/ICAS2022_0693.htm.

[12] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015, DOI: 10.1017/CBO9781107337756.

[13] C. Zhao, S. Zhao, M. Zhao, Z. Chen, C.-Z. Gao, H. Li, and Y.-a. Tan, "Secure multi-party computation: theory, practice and applications," *Information Sciences*, vol. 476, pp. 357–372, 2019, DOI: 10.1016/j.ins.2018.10.024.

[14] J. Sakuma and S. Kobayashi, "A genetic algorithm for privacy preserving combinatorial optimization," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, 2007, pp. 1372–1379, DOI: 10.1145/1276958.1277214.

[15] S. Han and W. K. Ng, "Privacy-preserving genetic algorithms for rule discovery," in *DaWaK 2007*, ser. LNISA. Springer, 2007, pp. 407–417, DOI: 10.1007/978-3-540-74553-2_38.

[16] Q. Liu, Y. Yan, P. Ligeti, and Y. Jin, "A secure federated data-driven evolutionary multi-objective optimization algorithm," 2022, DOI: 10.48550/ARXIV.2210.08295.

[17] D. Funke and F. Kerschbaum, "Privacy-preserving multi-objective evolutionary algorithms," in *PPSN 2010*, ser. LNCS, vol. 6239. Springer, 2010, pp. 41–50, DOI: 10.1007/978-3-642-15871-1_5.

[18] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.

[19] X. Yi, R. Paulet, and E. Bertino, *Homomorphic Encryption and Applications*. Springer, 2014.

[20] J. Hromkovič, *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer, 2001, DOI:10.1007/978-3-662-05269-3.

[21] KIE Community, "GitHub Repository kiegroup/optaplanner – Branch 8.14.x." [Online]. Available: https://github.com/kiegroup/optaplanner/tree/8.14.x

[22] K. Schuetz, "SlotMachine: Privacy-Preserving OptaPlanner." [Online]. Available: https://github.com/jku-win-dke/optaplanner/tree/Slotmachine

[23] C. G. Schuetz, S. Jaburek, and K. Schuetz, "SlotMachine: Heuristic Optimizer." [Online]. Available: https://jku-win-dke.github.io/SlotMachine-Optimizer/

[24] ——, "Slotmachine: Heuristic optimizer." [Online]. Available: https://github.com/jku-win-dke/SlotMachine-Optimizer/tree/kschuetz

[25] C. G. Schuetz, T. Lorünser, S. Jaburek, K. Schuetz, F. Wohner, R. Karl, and E. Gringinger, "A distributed architecture for privacy-preserving optimization using genetic algorithms and multi-party computation – Appendix," 2022. [Online]. Available: http://files.dke.uni-linz.ac.at/publications/schu22c/appendix.pdf

[26] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[27] K. Schuetz, C. G. Schuetz, and S. Jaburek, "Privacy-Preserving Implementation of Local Search Algorithms for Solving Assignment Problems in Time-Critical Real-World Applications," 4 2023, DOI: 10.6084/m9.figshare.21867903.v2. [Online]. Available: https://doi.org/10.6084/m9.figshare.21867903.v2

[28] M.-C. Silaghi and V. Rajeshirke, "The effect of policies for selecting the solution of a DisCSP on privacy loss," in *AAMAS '04*, 2004, pp. 1396–1397.

[29] P. Chu and J. Beasley, "A genetic algorithm for the generalised assignment problem," *Computers & Operations Research*, vol. 24, no. 1, pp. 17–23, 1997, DOI:10.1016/S0305-0548(96)00032-9.