Privacy-Preserving Implementation of an Auction Mechanism for ATFM Slot Swapping

Paul Feichtenschlager*, Kevin Schuetz*, Samuel Jaburek*, Christoph G. Schuetz†, and Eduard Gringinger‡
*Johannes Kepler University Linz, Linz, Austria
{feichtenschlager,kschuetz,jaburek}@dke.uni-linz.ac.at
†Johannes Kepler University Linz, Linz, Austria
ORCID: 0000-0002-0955-8647
‡ Frequentis AG, Vienna, Austria
ORCID: 0000-0003-3897-3003

***Abstract*—Air traffic flow management (ATFM) regulations issued by the EUROCONTROL Network Manager (NM) during periods of reduced capacity in the European air traffic network typically result in flight delays and additional costs for airspace users (AUs). However, not all flights are equally impacted by these regulations, and AUs would like to prioritize flights based on their preferences while protecting the confidentiality of such information. Thus, in the SlotMachine project, we proposed a privacy-preserving marketplace for collaborative optimization of flight lists during ATFM regulations An auction mechanism incentivizes AUs to participate in the SlotMachine's optimization runs. The proposed implementation of the auction mechanism in a privacy-preserving manner employs a genetic algorithm in combination with multi-party computation (MPC), since a privacy-preserving implementation of a deterministic algorithm would not finish within the time constraints. Experiments using realistic synthetic datasets based on real-world samples demonstrate feasibility of the proposed implementation.**

***Index Terms*—air traffic flow management, ATFM regulation, flight prioritization, combinatorial auction, genetic algorithm, multi-party computation**

I. Introduction

In cases of reduced capacity in the European air traffic network, the EUROCONTROL Network Manager (NM) issues air traffic flow management (ATFM) regulations, which cause flight delays that typically result in additional costs for the airspace users (AUs) operating the flights affected by a regulation. Regarding the operational impact of a regulation, not all flights are alike: For some flights, incurring delay is more problematic than for others. Therefore, AUs would like to be able to prioritize individual flights according to their preferences, which can be stated in terms of wished time slot and margins (time not after and time not before). Those preferences, however, are confidential information that AUs want to protect not only from competitors but also from an honest-but-curious platform provider.

In the SlotMachine project [1], which was funded by the SESAR Joint Undertaking in the EU Horizon 2020 research program, we proposed a privacy-preserving marketplace that allows AUs to collaboratively optimize flight lists in case of a regulation while protecting the confidentiality of the preferences for the optimization (margins and priority of flights) submitted by the participating AUs. The employed market mechanism is key to incentivize AUs to participate in the optimization. In previous work [2], we proposed a market mechanism based on combinatorial auctions with delay credits to compensate airspace users that give up favorable ATFM slots. Airspace users can give up slots for flexible flights in the present to earn credits that can be spent to prioritize important flights in the future.

The privacy-preserving implementation of an auction mechanism employed in the proposed marketplace, under the given time constraints (cf. [3]), requires a heuristic approach: We employ a genetic algorithm together with multi-party computation (MPC) to find feasible exchanges that produce the highest global utility. The genetic algorithm iteratively looks for candidate solutions to the optimization problem, i.e., finding feasible exchanges according to the auction mechanism. Each iteration step produces a new population of candidate solutions. A separate component—the Privacy Engine— evaluates the candidate solutions based on the preferences submitted by the airspace users. Multi-party computation ensures that the confidentiality of those preferences is guaranteed; the preferences are not decrypted on the server. Rather, multi-party computation distributes computation on multiple nodes. Furthermore, the Privacy Engine does not necessarily return precise fitness values for the candidate solutions but may further obfuscate the returned information regarding the fitness of candidate solutions. We propose different obfuscation methods, which complicate the search for solutions with an acceptable fitness value but improve privacy protection.

To evaluate the proposed solution, we conducted experiments with genetic algorithms using different configurations over synthetic datasets that were generated based on samples of the margins of real-world flights for real-world regulations, received from experts from Swiss International Air Lines who participated in the SlotMachine project. We compared the found solutions using the various configurations with the deterministic optimum, which in a real-world operational setting cannot be found within the time limit. With respect to previous work [4], we specifically focus on the implementation of the proposed auction mechanism, which requires certain optimization techniques in order to yield good results. We also used more realistic datasets in our evaluation than in previous work. Our findings can serve as the basis for building a privacy-preserving marketplace for ATFM slot swapping using an auction-based mechanism that ensures equity and fairness over time.

The remainder of this paper is organized as follows. In Section II we provide relevant background information, including a review of related work. In Section III we briefly present the auction mechanism proposed by Schuetz et al. [2] for the SlotMachine platform. In Section IV we investigate the privacy-preserving implementation of the previously presented auction mechanism. In Section V we discuss the setup of experiments conducted for evaluation purposes, including generation of realistic synthetic datasets, configuration of the algorithm, and employed metrics. In Section VI we present the results of the experiments. In Section VII we discuss those results. In Section VIII we conclude the paper with a summary and an outlook on future work.

II. Background

In this section, we provide background information on ATFM slot swapping, the SlotMachine platform's architecture, and genetic algorithms, which the SlotMachine platform employ for optimization (in combination with multi-party computation); we also review related work.

A. Slot Swapping in Air Traffic Flow Management

If an airport runs at full capacity, even minor events can cause flight delays. When such a congestion occurs, a new assignment of flights to ATFM slots is required. Mapping two types of entities, e.g., flights and slots, while minimizing the cost or maximizing the utility of the assignment, is referred to as *assignment problem* [5]. In the original form, the assignment problem describes the task of assigning agents to jobs. No agent can be overloaded, and each task needs to be assigned to one agent [6]. In the ATFM context, each flight must be assigned to one slot, and no slot can be used more than once.

Currently, the Network Manager generally follows a *first-planned, first-served* (FPFS) approach: Flights are assigned new slots in the order that was originally planned. When applying the FPFS approach, the order of the flights stays roughly the same and thus is considered to be fair [7].

In practice, the costs incurred by delays are not linearly dependent on the duration of the delay. Some flights can tolerate delays more easily than others, as the underlying cost structure is different for each flight. In addition, the relationship between additional cost and the duration of the delay can typically be described through a step function. Up to a certain point in time, flights can be pushed back rather cheaply, whereas the costs rise sharply after such delay targets. Each flight can have one or more delay targets, the positions of the delay targets differing for each flight [7]. Figure 1 illustrates the difference between FPFS and utility-based or cost-based optimization. The flight list on top shows the original schedule of the flights. The flight list in the middle shows rescheduling of the flights according to the FPFS principle due to reduced capacity. The flight list on the bottom

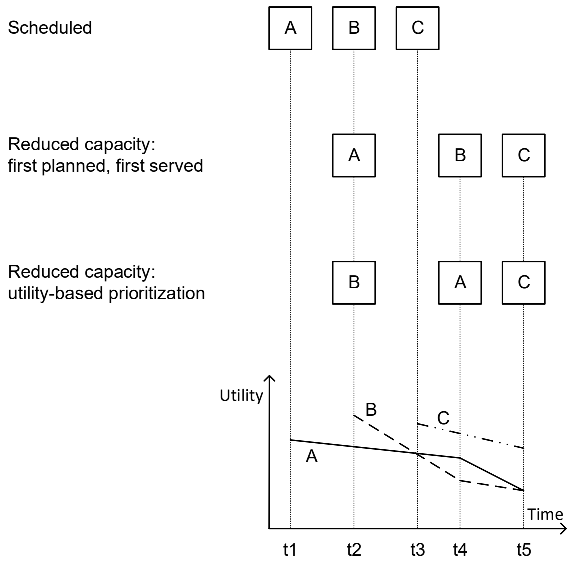

Fig. 1. Different options for flight prioritization in case of reduced capacity in the air traffic network. In case the original schedule cannot be kept due to reduced capacity, a utility-based prioritization of flights that takes into account the flight-specific utility functions produces a better solution in terms of global utility than a first planned, first served approach.

shows the optimal flight list given a utility function for each flight. Flight A has to endure only a modest decrease in utility (or increase in costs) in case of a delay until t4, after which there is a sharp drop in utility (or increase in costs). Flight B, on the other hand, already has a quite steep decline in utility. Flight C has only a flat decline in utility. Hence, it would be beneficial globally to give the first slot to Flight B instead of Flight A. Note that no flight can be assigned before its originally scheduled time slot.

Due to the varying cost structures, an FPFS approach is not always optimal for AUs in terms of additional costs [8]. In the User-Driven Prioritization Process (UDPP), an airline may rearrange its flights in an optimal order [9]. To increase the benefit, an airline may swap its own flights, based on the cost functions (or other criteria). Yet, since airlines are only operating a limited number of flights at an airport, the possibility of swapping is restricted. Therefore, the possibility to exchange slots between different airlines would increase the flexibility of AUs.

Based on the assumption that it is the AUs themselves that best know the preferences and underlying cost structures for the various flights, an overall more beneficial flight list could be generated when the optimization process takes into account the information of all AUs [10]. In such a case, optimizing a flight list may involve assigning some flights to less favorable slots compared to the FPFS approach, so a market mechanism needs to be put in place to compensate airlines that accept a later slot for flights. AUs receiving an improved slot allocation through the optimization have to compensate AUs accepting a displacement for their flights. Castelli et al. [11] name two factors for a market mechanism to be accepted. On the one hand, a non-negative payoff for each participant needs to be reached, and on the other hand, the mechanism needs to be fair over time [11]. Schuetz et al. [2] introduce an auction-based mechanism (see Section III), which we implement in this paper using a genetic algorithm.

B. SlotMachine Architecture

For a system to find an optimal flight list, the system requires information about the preferences of multiple participants. However, airlines are reluctant to make that information public or even entrust it to a third party. Loruenser et al. [3] propose the use of multi-party computation (MPC) to protect the AUs' preferences during the optimization, which can be combined with a genetic algorithm that looks for candidate solutions [4]. An MPC implementation of a deterministic algorithm, e.g., the Hungarian method (or algorithm) [12], would not finish within the time constraints of the ATFM use case [3]. A heuristic optimization algorithm, which separates the search for candidate solutions from the evaluation of the solutions may yield a result within a shorter time period but may not find the optimal solution—there is a trade-off between privacy protection and performance.

The SlotMachine system is a system based on MPC and genetic algorithms, the main components of the system being the Privacy Engine (PE) and the Heuristic Optimizer (Figure 2); the Controller coordinates communication between Network Manager, SlotMachine system, and AUs while also initializing the optimization runs. The Heuristic Optimizer iteratively looks for candidate solution to the optimization problem, the PE employs MPC to securely evaluate the candidate solutions found by the Heuristic Optimizer after each iteration step, the confidentiality of the preferences being protected through secret-sharing. MPC nodes that are hosted outside the system, potentially operated by different AUs, conduct the required computations. To further increase the security of the private inputs, different obfuscation methods may conceal the resulting fitness values from the Heuristic Optimizer. The PE returns the evaluated population, including obfuscated information about the individual fitness values, to the Heuristic Optimizer [4]. The Heuristic Optimizer estimates fitness values for all individuals based on the returned information, and uses a genetic algorithm or a local search algorithm to construct the next generation of candidate solutions. This iterative process continues until a termination criterion is met. The obfuscation methods can be divided into two groups, depending on whether sorting is required. This is important from a performance point of view since the need for sorting the entire population drastically reduces the performance in the context of a MPC. We refer to Section IV-C for a description of the different obfuscation methods.

C. Genetic Algorithms

A genetic algorithm (GA) is a metaheuristic based on the idea of natural selection in evolution, which can be employed to solve various optimization problems. The underlying concept of GAs—or, more generally, evolutionary algorithms—is that a population of candidate solutions for the problem is generated and then evaluated over multiple iterations (called generations). Based on the evaluation of the fitness of each candidate solution (individual) of a population, the GA generates a new population (the next generation). Figure 3 illustrates the principle of a GA.

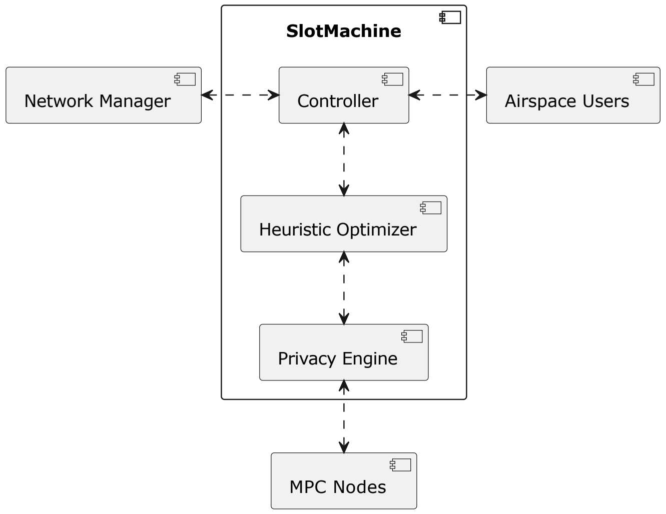

Fig. 2. The architecture of the SlotMachine platform (adapted from Schuetz et al. [13, p. 10])

Every individual in a population is characterized by several properties (chromosomes). A defined number of individuals (population size) makes up the population in a generation. Each solution in the generation is evaluated using a fitness function, taking into account the chromosomes of each individual. This evaluation is the basis for the composition of the next generation. Iteratively, a new generation is built based on the previously evaluated generation [14]. Solutions with a high fitness value (and their chromosomes) will more likely be present in the next generation. Thus, promising areas of the search space will likely be explored and the fitness of the solutions can be expected to rise over time [15].

To build a new generation, three concepts are involved. First, selection is applied to the candidate solutions. This concept is comparable to the survival of the fittest principle in biology [16]. Second, crossover is used to combine two or more solutions. Crossover selects a *locus* (position) on the chromosome, which determines which chromosomes are used from which solution [16, p. 5]. Chromosomes before the locus are used from one solution and chromosomes after the locus are used from the other solution [16, p. 8]; it is also possible to use more than one locus to combine two solutions. In biological terms, this approach is comparable to recombination [16]. Finally, mutation is applied, in which individual chromosomes are changed randomly. Mutation tackles the problem that two parents with similar chromosomes can only produce children with those chromosomes. The changes caused by mutation are rather small but help with exploring also deviating (off-shot) solutions and thereby increasing the diversity in the

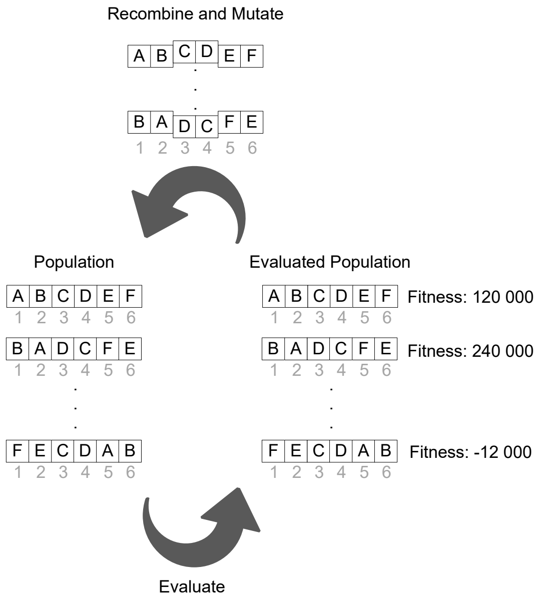

Fig. 3. Illustration of a genetic algorithm for the assignment problem

population [17]. When building new generations, a GA highly depends on randomness [18]. Due to the introduction of randomness, the results of GAs are not deterministic. Thus, two executions of a GA with the same input data do not necessarily lead to the same results.

D. Related Work

Multiple works have explored privacy-preserving auction mechanisms in different contexts. Gao et al. [19] propose a *privacy-preserving auction scheme* (PPAS) for auctioning data commodities generated by IoT (internet of things) networks. Chen et al. [20] design a privacy-preserving spectrum auction mechanism utilizing cryptographic techniques. Similarily, Huang et al. [21] suggest PISA, a privacy-preserving integer comparison protocol to realize an auction-mechanism for spectrum allocation. Qiu et al. [22] focus on combinatorial spectrum auctions and presented PICASSO, based on homomorphic encryption, protecting privacy of bidders with regards to both bidding values and bundles. Zhai et al. [23] propose the ExPO auction mechanism to allocate electricity from swap stations to electric vehicles in microgrids, while ensuring privacy and individual utilities. In the context of cloud computing, Wang et al. [24] propose PADS, a privacy-preserving auction design mechanism for dynamic pricing of cloud resources, which protects private information in bids through differential privacy guarantees. Palmer et al. [25] suggest an algorithm for the development of a combinatorial auction circuit that computes the result of a combinatorial auction using garbled circuits, which provides privacy for all bids except the winning bid. The proposed privacy-preserving auction mechanism for ATFM slot swapping shares similarities with those previous works on privacy-preserving auctions, which also address security and privacy concerns using cryptographic techniques. Whether the related approaches are suitable in practice for the ATFM use case remains doubtful since the SlotMachine system employs a customized variant of combinatorial auctions. Nevertheless, future work will investigate the implementation of the SlotMachine auction mechanism directly using MPC and compare performance to the GA-based approach.

III. Auction Mechanism for ATFM Slot Swapping

In the following, we summarize the principles of the auction-based market mechanism for the SlotMachine platform proposed by Schuetz et al. [2]. We first describe the types of flights before discussing exchange for credits as a form of compensation of AUs that give up favorable slots.

The auction mechanism is based on combinatorial auctions, where bidders can bid for a bundle of objects but only want to receive one [26]. The application of combinatorial auctions in the aeronautics domain is not new: Rassenti et al. [27] show how combinatorial auctions can be employed for the assignment of airport time slots. In the proposed mechanism for the SlotMachine platform, AUs can bid for earlier ATFM slots for important flights. On the other hand, AUs with flexible flights can offer ATFM slots for auction. Based on the bids and "asks" submitted by the AUs, the optimization algorithm finds feasible exchanges with high utility. The bids and exchanges are the basis for computation of credits that airlines receive for accepting additional delay or that airlines have to pay in case of receiving a better slot. The idea is that airlines may accept later ATFM slots for certain flights to be able to reduce delays for priority flights in the future.

A. Types of Flights

In the context of the auction-based market mechanism, flights can be classified according to the bids and offers ("asks"), respectively, associated with the flight. The starting point is the time that the flight is assigned in the initial flight allocation. As described in Section II-A, the initial flight allocation follows an FPFS approach. The type of flight refers to whether the airline wants to receive an earlier slot than assigned or is open to pushing a flight further back. Flights for which neither is true are not participating in the optimization process and will keep their original position.

1) Flexible flights: Flights for which the AUs are willing to receive a later slot if compensated with credits are referred to as flexible flights. The margins within which flexible flights can be pushed forth or back are defined through a TimeNotAfter parameter provided as input to the optimization by the airlines. If the flight is assigned to a later ATFM slot than planned in the initial allocation, the AU operating this flight will receive credits as compensation. The number of credits depends on the amount of additional delay endured.

2) Priority flights: For some flights, AUs are willing to invest credits to receive better ATFM slots in an optimized flight list. Such flights are referred to as priority flights. AUs are bidding for one or more slots for those flights and define how many credits they are willing to pay at most for a certain ATFM slot. If the optimization assigns a better AFTM slot to the flight, the AU receiving the better slot must compensate the original "owner" of the slot. When other AUs are bidding for the same slot, not only the previous holder of the resource but also the other AUs who did not receive it must be compensated for the sake of fairness.

3) Priority flights with flexibility: Priority flights with flexibility can be seen as a combination of the first two types of flights. For priority flights with flexibility, AUs are bidding for better ATFM slots but at the same time are offering their current slot if no better slot can be obtained. When the flight receives a better slot, credits must be paid similarly to priority flights. On the other hand, the AU receives credits if the flight is assigned to a worse slot.

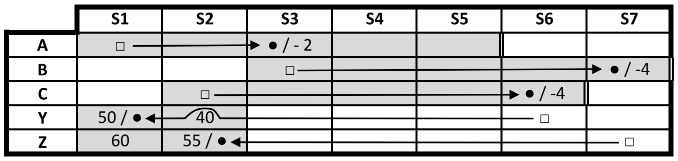

Fig. 4. Illustration of bids and "asks" for different flights as well as the optimal slot swaps between flights (adapted from Schuetz et al. [2]). A square (□) denotes a flight's originally assigned slot, a bullet (●) the assigned slot after swapping. The grey area denotes feasible exchanges, a double vertical line the time not after for flexible flights. Positive numbers indicate bids for better slots, arrows indicate slot swaps, and a negative number represents the number of slots that a flight is pushed back by an exchange.

Figure 4 shows an example of flexible flights (Flights A, B, and C) and priority flights (Flights Y and Z). Flight A (originally at Slot S1) would be willing to be pushed back to S5, Flight B (S3) to S7, and Flight C (S2) to S6. Flight Y should be allocated to S1 or (with lower preference) S2. Likewise, Flight Z should be allocated to S1 or (with lower preference) S2. The optimal solution would be to allocate Flight Y to S1 and Flight Z to S2 (maximizing the utilities expressed by the bids) while pushing back Flight A to S3, Flight B to S7, and Flight C to S6.

B. Credits

For compensating AUs, the market mechanism employs non-monetary delay credits. Those credits cannot be exchanged for real currency. Credits can only be used to bid for better ATFM slots. Therefore, the value of the credits is linked to the cost of delay that can be avoided through their usage. This value differs for each AU since the underlying cost structure varies. If the exchange rate of one credit is determined by one minute of delay reduction, the cost saving which goes along with it is a different amount of money for each airline.

We refer to Schuetz et al. [2] for a discussion of the market mechanism's properties and an explanation how the amount of credits to be exchanged as compensation and congestion fee, respectively, are computed. Intuitively, the amount of credits that an AU receives for a flight is proportional to the amount of additional delay accepted. For example, the AU operating Flight A in Figure 4 would receive two credits if pushed back two slots to S3. In turn, the amount of credits that the AU operating Flight Y would have to pay if receiving S1 would depend on the bid of 50 placed by the AU (but will likely be lower than that).

Ball et al. [28] state that in order to be accepted in the long run, a market mechanism must be fair and beneficial for all participants. Similar requirements are stated by Castelli et al. [11]. The SlotMachine auction mechanism aims to spread the gain in overall utility between the participants in an equitable manner. Every AU should receive a share of the utility gain, including the AUs that receive a better slot but not the preferred slot(s) for their flight(s). AUs operating flights that are either accepting a worse ATFM slot or are not getting their wished slot will be compensated by AUs, the flights of which received the wished slot. For example, the AU operating Flight Z in Figure 4 would have to pay a compensation but would also receive a share of the utility gain of the AU operating Flight Y because Flight Z was not allocated its most preferred slot (S1). With the obtained credits, better ATFM slots can be obtained in the future. Furthermore, flights not participating in the optimization will not be affected and remain in their original position.

IV. Privacy-Preserving Implementation

In the following section, we present the extension developed for the SlotMachine system to optimize flight lists based on the presented auction mechanism. The preferences can still be captured in weight maps, as in the original implementation, but some things must be considered to capture the TimeNotAfter constraints properly.

A. Weight Maps

The preferences of the airlines need to be represented in a way that facilitates optimization. For every flight, a weight map can be created, indicating how valuable each slot is for the flight. However, the creation of such weight maps is time-consuming and difficult which is particularly disadvantageous in a time-critical real-world setting.

A much easier task is defining the preferences as margins [29]. In this context, three times are relevant for each flight: The TimeNotAfter determines the latest time a flight should be assigned, the TimeWished indicates the time an airline would prefer for a certain flight, and the TimeNotBefore is the earliest time an airline wants for a slot. In addition, a priority may indicate how valuable receiving the TimeWished for the flight would be for the airline [7].

Based on the margins, a weight map for each flight can be generated indicating the utility of assignments to the ATFM slots. In addition to the preferences (TimeWished, TimeNotAfter, TimeNotBefore) used by Schuetz et al. [4], for an auction-based market mechanism, the TimeAssigned is relevant. Depending on the type of flight, the weight map is shaped differently. In the following, for the three different types of flights, the shape of the distribution in the weight maps is explained.

1) Flexible Flights: Flexible flights have a TimeNotAfter that is later than the TimeWished. Within the window between the slot that was assigned initially and the TimeNotAfter, the

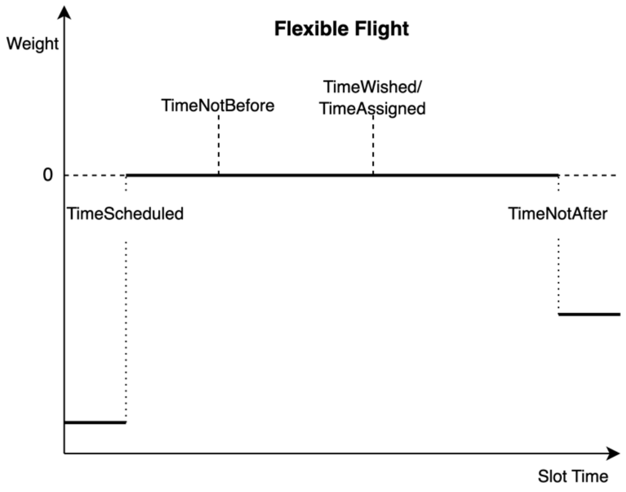

Fig. 5. Weight distribution of a flexible flight

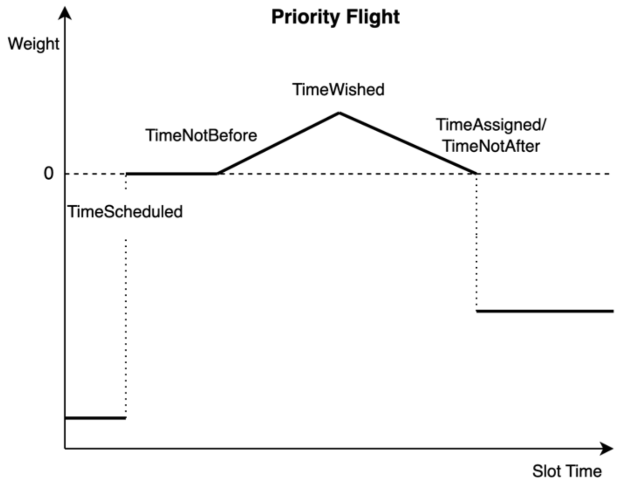

Fig. 6. Weight distribution of a priority flight without flexibility

flight can be pushed back. In the optimization, shifting is accepted up to the TimeNotAfter. For this reason, the weights are set to zero to represent the feasibility.

Alternatively, the weights could be set to a small negative number showing that a credit exchange is required for those slots. Therefore, the weights could decrease linearly with a small gradient. Since flexible flights are not bidding for better slots, the weight map contains no positive values. For this reason, the values from the TimeAssigned to the TimeScheduled are zero as well. Slots that are before the TimeScheduled and after the TimeNotAfter have (large) negative values. Those cases are constraint violations whereby the TimeScheduled is a hard constraint, i.e., no allocation before the TimeScheduled is allowed, and the TimeNotAfter is a soft constraint, i.e., no allocation after the TimeNotAfter is desired but the Network Manager would not complain. Figure 5 shows the weight function depending on the time of the slot for flexible flights.

2) Priority Flights: Priority flights have a TimeWished which is before the TimeAssigned. For priority flights, in addition to the margins, the number of credits an airline is willing to pay for the preferred slot is required. The weight at the TimeWished is equal to this value. From the TimeNotBefore to the TimeWished, the weights are increasing linearly—at least in the current proposal, which could be changed in the future—, starting from zero and peaking at the TimeWished. Between the TimeWished and the TimeAssigned, the weights are dropping, eventually reaching zero at the TimeAssigned. If the two constraints described earlier are violated, the weight map has a negative value. Figure 6 shows the weight function for priority flights. .

3) Priority Flights with Flexibility: The weight map is a combination of the first two types of flights. The TimeWished is before the TimeAssigned and the TimeNotAfter is after the TimeAssigned. The weight map is shaped similarly to priority flights before the TimeAssigned and like flexible flights after the TimeAssigned. Figure 7 shows the structure of the weight map for priority flights with flexibility.

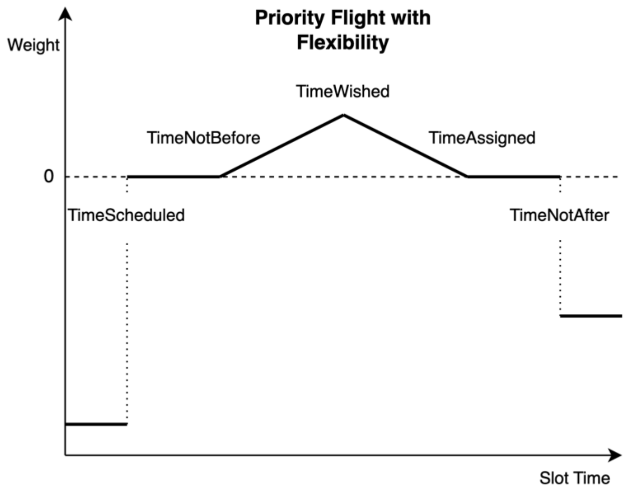

Fig. 7. Weight distribution of a priority flight with flexibility

B. Genetic Algorithm

As input data, the genetic algorithm receives a weight map containing bids and the TimeNotAfter for each flight, although a TimeNotAfter penalty can also be directly included in the weight map, which facilitates computation for the PE. With this information, the weight representation can be computed as introduced in Section IV-A. Based on those inputs, the Privacy Engine computes the fitness of a solution. To optimize the flight list based on the auction-based market mechanism, conceptually, the fitness function is as follows [4]. First, solutions that do not violate any margins should be ranked higher than solutions that violate margins. Second, the sum of the bids should be maximized. Third, the number of swaps could be minimized.

In the current implementation, we do not aim to minimize the number of slots since the other two criteria are more important. The representation of those criteria is achieved through a fitness function. The fitness value for each solution is calculated based on the weight map and the TimeNotAfter. The weight map acts as a proxy for the bids. In the computation of the fitness, the weights of each flight for the assigned slot are summed up. In addition, the violations of the TimeNotAfter constraints are recorded. Finally, the fitness of a slot assignment is calculated by subtracting the number of constraint violations multiplied by the weight of the penalty violation, from the cumulated values in the weight map.

Formally, the assignment problem can be characterized as a graph G, linking a set of Flights U to a set of slots V, where the set of edges E represents the valid assignments, i.e., assignments not violating a soft or a hard constraint, and w a function representing the weight maps of the flights. We base the following fomulation of the problem on Burkard et al. [5].

$$G = (U, V, E, w)$$

Hence, the set of edges E is a relation between U and V.

$$E \subseteq U \times V$$

The total injective function w represents the weight maps of all flights. For each assignment of a flight to a slot, a fitness value is defined independently of the feasibility of the assignment, although the TimeScheduled and the TimeNotAfter constraints could also be directly encoded into w.

$$w : U \times V \to \mathbb{Z}$$

A solution is represented through a total injective function ϕ assigning each flight in U to a slot in V.

$$\phi : U \to V$$

The fitness of a solution is defined as the sum of the fitness of the assignments of all flights in U. For each assignment of a flight $u \in U$ to a slot $v \in V$ that is not part of the set E, and thus violates a constraint, the individual fitness of the assignment is defined as -100 000—a value which could also be changed. If the pair (u, v) is part of E, the fitness is defined by the the weight, represented by the result of the function w for u and v as arguments. Algorithm 1 presents the procedure to calculate the fitness of a candidate solution ϕ, considering the set of valid assignments E and the weight map w, as well as a penalty p, which in our case was set to 100 000.

To ensure that the influence of constraint violation is greater than the influence of the sum of bidding, the weight of the penalty was determined at a height to always extends the sum of bids. Therefore, the fitness of a solution with a constraint violation always results in a negative overall fitness. For example, the swapping in Figure 8 violates the TimeNotAfter of Flight A, and thus the solution has a value lower than -100 000, which is lower than any solution containing only valid assignments.

If the inputs that are provided by AUs are bids and TimeNotAfter constraints as described by the auction mechanism, the fitness function of the genetic algorithm corresponds to Algorithm 1. In practice, to facilitate computation of a solution's fitness by the Privacy Engine, and as input for the Hungarian method, the penalty can already be included in the weight map for slots after a flight's TimeNotAfter margin.

Algorithm 1: Fitness function

```
Data: a candidate solution $\phi : U \to V$, a weight map
      $w : U \times V \to \mathbb{Z}$, the set $E \in U \times V$ of valid
      mappings between $U$ and $V$, and a penalty
      $p \in \mathbb{N}$ for invalid assignments
Result: the fitness value $f$ of $\phi$ given $w$, $E$, and $p$
1 $f \leftarrow 0$
2 for each $u, v$ such that $\phi(u) = v$ do
3     if $(u, v) \in E$ then
4         $f \leftarrow f + w(u, v)$
5     else
6         $f \leftarrow f - p$
7     end
8 end
9 return f
```

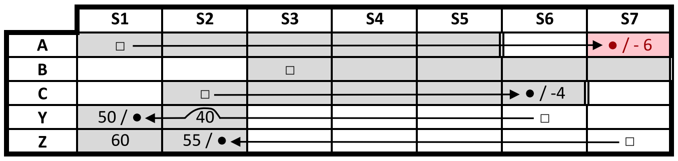

Fig. 8. An example of slot swapping that results in the violation of margins and thus constitutes an infeasible exchange.

We employed the Jenetics framework [30] to implement the genetic algorithm in Java. Jenetics provides implementations of common recombination and mutation operators as well as selectors and efficient encodings for common optimization problems, including the assignment problem. By using Jenetics, we could easily experiment with a multitude of configurations of the genetic algorithm.

C. Obfuscation Methods

Disclosing actual fitness values to the Heuristic Optimizer during the optimization potentially threatens the confidentiality of the private input data provided by AUs. For this reason, different obfuscation methods were developed to disguise actual fitness values and improve protection of confidentiality of private input data. Depending on the obfuscation method, the Heuristic Optimizer receives limited information about the fitness values of individuals within a population from the Privacy Engine. Based on this information, the Heuristic Optimizer estimates fitness values for each solution. The estimated fitness values are the foundation for selection and recombination by the GA.

In the following we briefly present the obfuscation methods. The obfuscation methods can be divided into two groups, depending on whether they require sorting of candidate solutions during evaluation of a population. This is important from a performance perspective, as sorting cannot be parallelized in the context of MPC, which has a negative effect on running time.

1) Order: When applying the Order obfuscation method, the Heuristic Optimizer receives the order of the individuals along with a flag that indicates if a new maximum fitness has been reached in this generation with regards to the whole optimization process. The Heuristic Optimizer then estimates the maximum and minimum fitness value in the generation and applies a linear or logarithmic function to estimate fitness values for each individual. To estimate the maximum fitness values, the optimization starts with an arbitrary initial maximum fitness which is estimated for the first generation. In subsequent iterations, this value is increased by 1 each time the Heuristic Optimizer receives the information that a new actual maximum fitness value has been achieved with regards to the whole optimization process. This way, the GA is able to accurately determine the generation in which the maximum fitness has been achieved. The minimum fitness value is estimated to be roughly the maximum fitness minus two times the maximum fitness. This approach is employed for all obfuscation methods. Based on the estimated maximum and minimum fitness values, a linear or logarithmic function is applied to estimate fitness values between the maximum and minimum for all individuals according to the disclosed order of individuals. Although this method hides actual fitness values, it still discloses a significant amount of information regarding dominance of individuals. Also, as this method requires sorting during evaluation, it is not optimal from a performance point of view.

2) Order Quantiles: When applying the obfuscation method Order Quantiles an additional parameter is required indicating over how many buckets the individuals should be spread. The ordered generation then gets put in the buckets containing the same amount. If ten buckets are used, the first bucket contains the best 10% solutions. The Heuristic Optimizer receives the mapping of individuals to buckets and estimates a fitness value for each bucket and the contained solutions, similar to the approach described for the obfuscation method Order, so that all candidate solutions in the same bucket are assigned the same estimated fitness value. This approach increases the confidentiality of fitness data and is thus practically applicable, although this method still requires sorting.

3) Top Individuals: When applying the top individuals obfuscation method the individuals with the highest fitness values are disclosed to the Heuristic Optimizer. This method is a special case of the Order Quantiles method. A parameter determines how many percent of a generation are marked as top individuals. Neither is the exact order of the best individuals nor of the rest revealed. However, sorting is still required.

4) Above Threshold: When applying the above threshold method all individuals whose actual fitness surpasses a threshold are disclosed to the Heuristic Optimizer. The threshold value is determined by a configurable parameter relative to the maximum fitness of the population. When the parameter is 70%, and the maximum fitness is 1000, values with a fitness value over 700 are disclosed. This method is comparable to the Top Individuals method regarding the amount of disclosed information, but does not require sorting.

5) Fitness Range Quantiles: When applying the Fitness Range Quantiles fitness method, the individuals of a generation are spread in buckets. Each bucket contains individuals within a specific fitness range. An input parameter defines how many buckets are used. The distance between the highest and the lowest values is separated into equally long ranges. Each range then is assigned to one bucket. Contrary to the order quantiles approach, not every bucket contains the same number of individuals.

D. Privacy Engine

In the SlotMachine system, the Privacy Engine is responsible for coordinating the MPC. The component receives encrypted inputs about the preferences for each AU at the beginning of an optimization. In every iteration, the Heuristic Optimizer transmits a population of candidate solutions to the Privacy Engine for evaluation. The Privacy Engine uses the encrypted inputs and the MPC Nodes, which are hosted outside the system, to cooperatively evaluate the fitness functions for the individuals of the population to produce the required information according to the chosen obfuscation method (see section IV-C). For the work at hand, MPC and the function of the Privacy Engine were simulated by considering only the amount of information that would be disclosed by the Privacy Engine (for a given obfuscation method) during the optimization. We refer to Loruenser et al. [3] and Schuetz et al. [4] for further information about the Privacy Engine and the MPC employed in the SlotMachine system.

V. Experimental Setup

This section will detail the setup for the conducted performance experiments. We will first describe the creation of synthetic datasets that were used as inputs for the experiments. Afterwards, the details regarding the configuration parameters of the GA will be presented, before we conclude with a brief documentation of the metrics chosen for further evaluation of the results.

A. Datasets

For the experiments, we generated synthetic datasets with 100 flights and slots, based on data samples of margins for flights in real-world regulations, annotated by experts from Swiss International Air Lines. Note, however, that real datasets in a production system will likely differ, since we only have data from one airline and some of the samples are the result of using an automatic tool, with no further checks by human experts. The provided preferences were analyzed regarding the widths of the margins, i.e. the time window between the time not before and the time not after. Table II gives the absolute and relative frequencies that a margin falls into a certain interval in minutes in the provided real-world samples of preferences. In addition, the probabilities that a certain slot is wished by a given number of flights in the provided preferences have been analyzed. Table III presents the absolute and relative frequencies that a slot is wished by a certain number of flights. We also investigated the differences between the time wished and the time not before. Table I shows the absolute and relative frequencies of the difference between those points in time in the real-world samples of preferences. Figure 9 shows, for one example dataset modeled after the provided real-world samples, the distribution of flights with regards to the differences between the time wished and the time not after to the time assigned.

TABLE I
ABSOLUTE (f) AND RELATIVE (p) FREQUENCIES OF DIFFERENCE (Δ) IN MINUTES BETWEEN THE TIME WISHED AND THE TIME NOT BEFORE IN THE REAL-WORLD SAMPLES OF PREFERENCES

Δ	f	p
0 or 1	40	0.1375
2	8	0.0275
3	8	0.0275
4	12	0.0412
5	223	0.7663

TABLE II
ABSOLUTE (f) AND RELATIVE (p) FREQUENCIES OF MARGIN WIDTHS FALLING INTO A CERTAIN INTERVAL IN MINUTES IN THE REAL-WORLD SAMPLES OF PREFERENES

Interval	f	p
$[0, 10]$	49	0.1684
$[11, 20]$	54	0.1856
$[21, 30]$	66	0.2268
$[31, 40]$	36	0.1237
$[41, 50]$	21	0.0722
$[51, 60]$	12	0.0412
$[61, 70]$	14	0.0481
$[71, 80]$	6	0.0206
$[81, 90]$	3	0.0103
$[91, 100]$	2	0.0069
$[101, 110]$	1	0.0034
$[111, 120]$	2	0.0069
$[131, 140]$	1	0.0034
$[151, 160]$	1	0.0034
$[171, 180]$	2	0.0069
$[181, 190]$	2	0.0069
$[191, 200]$	2	0.0069
$[211, 220]$	1	0.0034
$[221, 230]$	1	0.0034
$[251, 260]$	3	0.0103
$[261, 270]$	1	0.0034
$[271, 280]$	1	0.0034
$[281, 290]$	4	0.0137
$[291, 300]$	2	0.0069
$[321, 330]$	1	0.0034
$[571, 580]$	1	0.0034
$[701, 1100]$	2	0.0069

TABLE III
ABSOLUTE (f) AND RELATIVE (p) FREQUENCIES OF A TIME BEING WISHED BY n FLIGHTS IN THE REAL-WORLD SAMPLES OF PREFERENCES

n	f	p
1	239	0.9019
2	26	0.0981

B. Configurations

We executed different test runs as part of our work. In between the test runs, changes to the implementation and

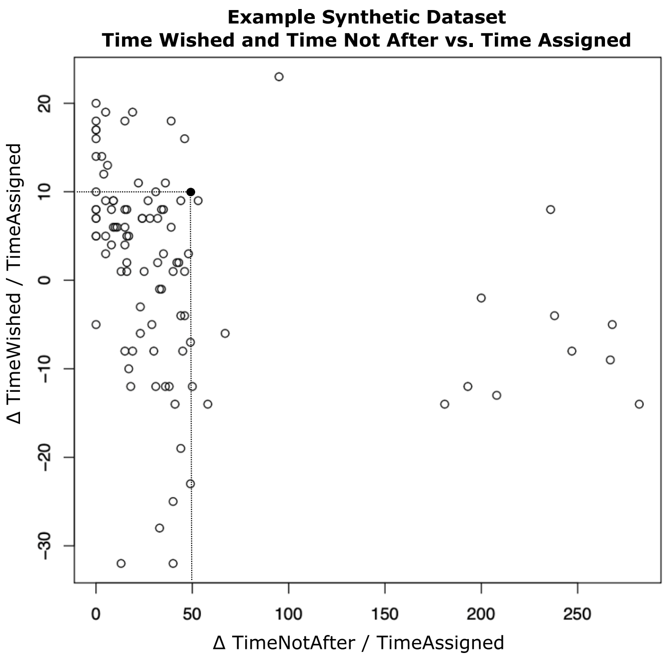

Fig. 9. Time wished and time not after relative to the time assigned (in the original flight list) for each flight in a representative example synthetic dataset employed in the experiments. The black point was added for illustration purposes, denoting a flight where the time assigned is 50 minutes before the time not after and 10 minutes after the time wished.

adjustments to the configuration of the GA have been realised to increase the performance. The results presented in Section VI cover the final experimental run. This experimental run has been designed to compare the performance of different obfuscation approaches when combined with an auction-based market mechanism. Therefore, we chose three obfuscation methods (Actual Values, Fitness Range Quantiles, Order Quantiles). The method Actual Values has been chosen as a baseline for comparison and practically represents no obfuscation of fitness values, whereby the estimated fitness values of solutions correspond to their actual fitness values (refer to Section IV-B regarding the fitness function). Fitness Range Quantiles represents obfuscation methods that do not require sorting, the number of buckets was set to 20. Order Quantiles represents obfuscation methods that require sorting and the number of quantiles was set to 10. See section IV-C for more details about the obfuscation methods.

We repeated each test five times and used the mean results for further analysis. Each test execution has been configured to terminate after a maximum of 500 generations.

Despite the obfuscation methods, the remaining configuration was the same for all test executions in the final run. The GA was configured with a population size of 500, so that each generation consists of 500 individuals; i.e. candidate solutions. For the selection of survivors, a truncation selector was applied to select the 10 individuals with the highest fitness values in each generation. Likewise, a truncation selector was used to select 10 individuals for recombination to produce the offspring. 95% of each newly created population have been

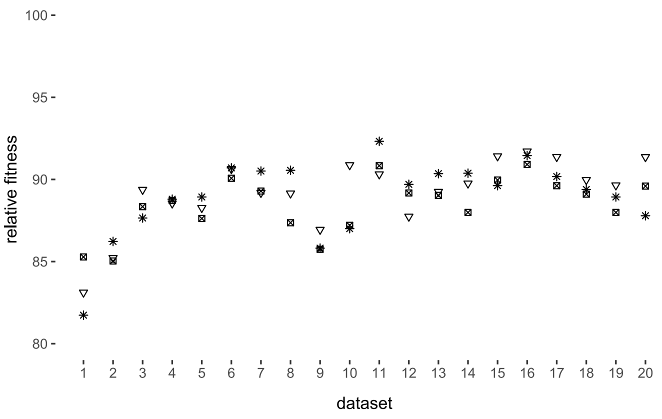

Fig. 10. The mean relative fitness in percent of the optimal solution's fitness found over five runs with 500 generations for each dataset and obfuscation method (actual values ▽, fitness range quantiles ⊠, and order quantiles ⋆)

created as offspring, the remaining 5% as survivors. Regarding altering during evolution, a swap mutator was used with a probability of 15% which randomly swapped the slots of two flights in a solution. Additionally, the offspring was randomly altered with a probability of 99% using the same mutation approach.

C. Metrics

We investigate *performance* of the genetic algorithm in its various configurations, measured by the fitness of the found solution relative to the optimal solution returned by the deterministic Hungarian method. We are further interested in the *convergence* of the genetic algorithm: After a certain number of generations, the genetic algorithm will hardly make improvements. The fewer generations it takes to find a good solution, the better in terms of running time.

VI. Experimental Results

In the following, we present the results of the experiments. More detailed results as well as links to the data can be found in the online appendix [31].

A. Performance

This section presents the results from the conducted performance experiments. First, we will give details regarding the achieved fitness values relative to the optimum fitness as determined by the Hungarian algorithm.

Table IV shows the mean and minimum fitness values for each combination of dataset and obfuscation method, relative to the optimum fitness of the respective dataset. The values represent the mean and minimum fitness value of the best solution found after 500 generations.

Overall, the method actual values achieved the best results with a mean relative fitness over all datasets of 89.19%. This method has been added as a baseline for comparison and in fact represents no obfuscation of fitness methods by using the actual fitness values of each solution as the basis for evolution by the GA. Therefore, it was expected that this approach performs best, as information is not limited by it.

TABLE IV
Mean and Minimum Relative Fitness of the Solutions Found over Five Runs with 500 Generations per Obfuscation Method (Actual Values, Fitness Range Quantiles, Order Quantiles) and Dataset in % of the Optimal Solution's Fitness

Dataset	Actual Values		Fitness Range Quant.		Order Quant.	
	Mean	Min	Mean	Min	Mean	Min
1	83.12	78.04	85.28	82.43	81.73	78.85
2	85.22	81.93	85.03	82.60	86.23	82.89
3	89.37	85.70	88.34	87.54	87.65	86.19
4	88.54	84.66	88.72	85.93	88.80	86.49
5	88.27	87.29	87.62	83.52	88.93	87.54
6	90.64	88.14	90.07	88.50	90.70	86.53
7	89.19	87.29	89.29	86.30	90.51	87.17
8	89.15	84.89	87.36	84.85	90.56	89.37
9	86.94	84.41	85.75	83.77	85.82	83.34
10	90.88	89.49	87.21	83.47	87.00	85.67
11	90.32	89.67	90.84	88.44	92.32	90.76
12	87.74	86.27	89.18	84.87	89.70	88.06
13	89.25	85.99	89.03	85.94	90.35	87.07
14	89.76	87.79	87.99	82.10	90.38	89.34
15	91.41	91.24	89.96	86.37	89.62	87.13
16	91.71	90.33	90.91	89.77	91.45	88.32
17	91.37	88.95	89.62	85.89	90.18	89.11
18	89.97	86.68	89.10	85.92	89.39	87.85
19	89.64	88.34	87.99	85.76	88.93	87.64
20	91.37	89.48	89.59	88.52	87.79	85.51

However, the obfuscation methods Fitness Range Quantiles and Order Quantiles performed only slightly worse, with mean relative fitness values over all datasets of 88.14% and 88.90%, respectively. This shows that sufficiently good average performance can still be achieved, even if information about fitness values and dominance of individuals is reduced significantly.

However, in real-world settings, consistent performance might be equally important as the average results. The minimum relative fitness achieved by the methods Actual Values and Order Quantiles over all datasets and experiments was roughly 78%. Surprisingly, the highest minimum relative fitness was achieved by the obfuscation method Fitness Range Quantiles, topping the other two methods by nearly 4% with a value of 82.10%. Extended experiments might be required to further investigate this result.

Figure 10 also visualizes the mean performance for each dataset and obfuscation method. The figure confirms that although Actual Values has been found to achieve the best overall performance, there are various instances in which the mean performance for a dataset with the obfuscation methods Fitness Range Quantiles and Order Quantiles surpass this method.

B. Convergence

Figure 11 shows, for each obfuscation method included in the final test run, the mean absolute fitness value over all datasets in each of the 500 maximum iterations. A dashed horizontal line indicates the mean optimum fitness over all datasets. It can be seen that solutions converge to sufficiently good regions before the maximum iterations are reached. Although the steepness of the curves indicate that further improvements are still possible after 500 generations, opti-

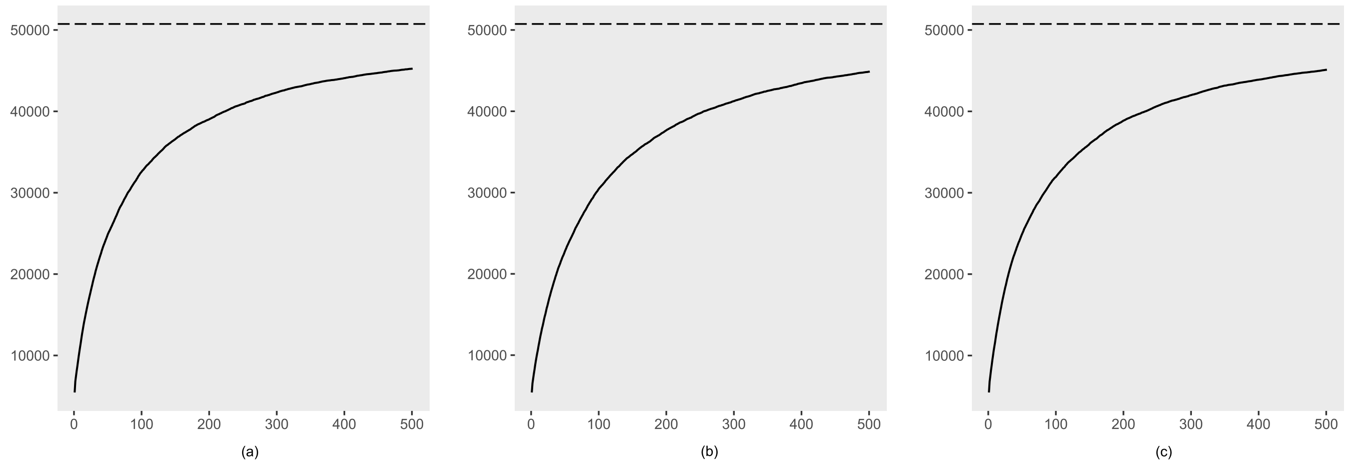

Fig. 11. The solid line indicates the mean fitness value of the solution found in each generation, aggregated over all datasets and repetitions, using the obfuscation methods (a) actual Values, (b) fitness range quantiles, and (c) order quantiles. The dashed line indicates the mean fitness value of the solutions found by the Hungarian algorithm, aggregated over all datasets.

mizations might be terminated after less than 500 iterations in time-critical settings and still produce satisfactory results. Furthermore, the convergence of obfuscation methods with limited information does not seem to be inferior to the results with actual fitness values.

VII. Discussion

The experiments show that using the SlotMachine approach to optimization of flight lists can also be applied in conjunction with the auction-based mechanism. The bids can be translated into weight maps, with infeasible exchanges being penalized with a large negative fitness. The obtained results have a high utility compared to the optimum that could be found by the heuristic Hungarian method. The privacy-preserving implementation of the Hungarian method has a considerable computational overhead [3]. The performance improvements of using a genetic algorithm for searching the solutions and multi-party computation for evaluating the solutions in an iterative manner works considerably faster, and has the advantage that the genetic algorithm can be aborted anytime and still return a result [4]. We can therefore conclude that the SlotMachine architecture can also be used in the future implementations of a production-ready system that employs an auction-based mechanism.

VIII. Summary and Future Work

We showed that a genetic algorithm together with multi-party computation (MPC) may serve to find feasible exchanges with high global utility for an auction-based market mechanism for ATFM slot swapping while preserving privacy of the confidential inputs of AUs. We conducted experiments using realistic synthetic datasets based on preferences for flights in real-world regulations provided by experts from Swiss International Air Lines.

Future work will investigate performance improvements for the proposed approach. Furthermore, future work will investigate market dynamics over time and risks to privacy as well as potential exploits that enables AUs to cheat the system by misrepresenting their utilities.

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] “Slotmachine: A privacy-preserving marketplace for slot management,” https://doi.org/10.3030/890456.

[2] 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 *ICAS 2022*, 2022, https://www.icas.org/ICAS_ARCHIVE/ICAS2022/data/preview/ICAS2022_0693.htm.

[3] T. Lorünser, F. Wohner, and S. Krenn, “A verifiable multiparty computation solver for the linear assignment problem: And applications to air traffic management,” in *CCSW 2022*, 2022, pp. 41–51.

[4] 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, M. Sellami, P. Ceravolo, H. A. Reijers, W. Gaaloul, and H. Panetto, Eds., vol. 13591. Springer, 2022, pp. 168–185.

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

[6] D. G. Cattrysse and L. N. Van Wassenhove, “A survey of algorithms for the generalized assignment problem,” *European journal of operational research*, vol. 60, no. 3, pp. 260–272, 1992.

[7] 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, pp. 1–11.

[8] N. Pilon, L. Guichard, Z. Bazso, G. Murgese, and M. Carré, “User-driven prioritisation process (udpp) from advanced experimental to pre-operational validation environment,” *Journal of Air Transport Management*, vol. 97, p. 102124, 2021.

[9] N. Pilon, A. Cook, S. Ruiz, A. Bujor, and L. Castelli, “Improved flexibility and equity for airspace users during demand-capacity imbalance-an introduction to the user-driven prioritisation process,” *Sixth SESAR Innovation Days*, 2016.

[10] S. Ruiz, L. Guichard, and N. Pilon, “Optimal delay allocation under high flexibility conditions during demand-capacity imbalance,” in *Proc. 7th SESAR Innov. Days*, 2017, p. 28.

[11] L. Castelli, R. Pesenti, and A. Ranieri, “The design of a market mechanism to allocate air traffic flow management slots,” *Transportation research part C: Emerging technologies*, vol. 19, no. 5, pp. 931–943, 2011.

[12] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[13] SlotMachine Consortium, *D2.2 – System Design Document*, 2021, https://doi.org/10.3030/890456 $\rightarrow$ Results $\rightarrow$ System Design Document.

[14] M. Kumar, D. Husain, N. Upreti, D. Gupta *et al.*, “Genetic algorithm: Review and application,” *Available at SSRN 3529843*, 2010.

[15] J. H. Holland, “Genetic algorithms,” *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992.

[16] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[17] K. Sastry, D. Goldberg, and G. Kendall, “Genetic algorithms,” *Search methodologies: Introductory tutorials in optimization and decision support techniques*, pp. 97–125, 2005.

[18] P. Krömer, V. Snáel, and I. Zelinka, “Randomness and chaos in genetic algorithms and differential evolution,” in *INCOS 2013*. IEEE, 2013, pp. 196–201.

[19] W. Gao, W. Yu, F. Liang, W. G. Hatcher, and C. Lu, “Privacy-preserving auction for big data trading using homomorphic encryption,” *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 2, pp. 776–791, 2020.

[20] Y. Chen, Z. Ma, Q. Wang, J. Huang, X. Tian, and Q. Zhang, “Privacy-preserving spectrum auction design: Challenges, solutions, and research directions,” *IEEE Wireless Communications*, vol. 26, no. 5, pp. 142–150, 2019.

[21] Q. Huang, Y. Gui, F. Wu, G. Chen, and Q. Zhang, “A general privacy-preserving auction mechanism for secondary spectrum markets,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1881–1893, 2016.

[22] F. Qiu, F. Wu, X. Gao, and G. Chen, “A privacy-preserving combinatorial auction mechanism for spectrum redistribution,” in *IPCCC 2017*, 2017, pp. 1–8.

[23] H. Zhai, S. Chen, and D. An, “Expo: Exponential-based privacy preserving online auction for electric vehicles demand response in microgrid,” in *SKG 2017*, 2017, pp. 126–131.

[24] J. Xu, B. Palanisamy, Y. Tang, and S. Madhu Kumar, “Pads: Privacy-preserving auction design for allocating dynamically priced cloud resources,” in *CIC 2017*, 2017, pp. 87–96.

[25] B. Palmer, K. Bubendorfer, and I. Welch, “Development and evaluation of a secure, privacy preserving combinatorial auction,” in *AISC ’11*, ser. AISC ’11. AUS: Australian Computer Society, Inc., 2011, p. 67–76.

[26] S. De Vries and R. V. Vohra, “Combinatorial auctions: A survey,” *INFORMS Journal on computing*, vol. 15, no. 3, pp. 284–309, 2003.

[27] S. J. Rassenti, V. L. Smith, and R. L. Bulfin, “A combinatorial auction mechanism for airport time slot allocation,” *The Bell Journal of Economics*, pp. 402–417, 1982.

[28] M. O. Ball, C.-Y. Chen, R. Hoffman, and T. Vossen, *Collaborative decision making in air traffic management: Current and future research directions*. Springer, 2001.

[29] C. G. Schuetz, E. Gringinger, N. Pilon, and T. Lorünser, “A privacy-preserving marketplace for air traffic flow management slot configuration,” in *DASC 2021*. IEEE, 2021, pp. 1–9.

[30] F. Wilhelmstötter, “Jenetics library user’s manual 7.1,” 2022. [Online]. Available: https://jenetics.io/manual/manual-7.1.0.pdf

[31] P. Feichtenschlager, K. Schuetz, S. Jaburek, C. G. Schuetz, and E. Gringinger, “Privacy-preserving implementation of an auction mechanism for atfm slot swapping – Appendix.” [Online]. Available: http://files.dke.uni-linz.ac.at/publications/schu23c/appendix.pdf