Privacy-Preserving Multi-Objective Evolutionary Algorithms

Daniel Funke and Florian Kerschbaum*

Email: daniel.funke@ieee.org;florian.kerschbaum@sap.com

September 13, 2010

Abstract

Existing privacy-preserving evolutionary algorithms are limited to specific problems securing only cost function evaluation. This lack of functionality and security prevents their use for many security sensitive business optimization problems, such as our use case in collaborative supply chain management. We present a technique to construct privacy-preserving algorithms that address multi-objective problems and secure the entire algorithm including survivor selection. We improve performance over Yao's protocol for privacy-preserving algorithms and achieve solution quality only slightly inferior to the multi-objective evolutionary algorithm NSGA-II.

1 Introduction

Evolutionary algorithms (EAs) have proven to efficiently find effective solutions for many real-world optimization problems and are therefore widely employed in business practice. Nevertheless in many real-world business problems, such as our use case from collaborative production planning, the data is distributed across a number of parties. A natural objective for each of these parties is to protect their data, in particular if it is sensitive for their well-being, e.g. business secrets such as production costs or capacities.

Privacy-preserving evolutionary algorithms (PPEAs) [8, 26] combine the privacy of input data with the effectiveness and efficiency of EA by using ideas from secure computation (SC) [29]. SC is a cryptographic technique that allows a number of parties to jointly compute a function $y = f(\vec{x})$ on their combined input $\vec{x}$, such that each party only learns the result y, but nothing else about the input $\vec{x}$.

The proposals for PPEA in the literature [8, 26] suffer from several shortcomings in generality and security. The setup chosen in [26] is such that one

[*] Daniel Funke and Florian Kerschbaum are with SAP Research CEC Karlsruhe, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe.

party maintains the population of individuals while the other owns the cost function. As we will explain in more detail in our use case, this is generally not the case in collaborative business scenarios. A different setup was chosen in [8], but the second problem is also evident in this proposal. When SC is only applied to cost function evaluation, the selection of individuals from the population reveals significant information about the cost function despite the use of provably secure protocols. Imagine the following situation: Given two individuals differing only in one characteristic, assume one survives and one dies. From this observation one can immediately conclude (even if the cost function was computed privately) that the survivor's characteristic was superior. Due to the many individuals and rounds in EAs this information quickly accumulates.

We improve over this in two aspects. First, we use a quite general setup for distributed optimization problems. Each party owns some arbitrary parts of the cost function. In addition, we use a multi-objective evolutionary algorithm (MOEA), in which each party may follow its own objective function. This is the natural setting in collaborative supply chain management where each party wants to minimize its own cost, but has gained an understanding that it needs to align with its partners in order to promote the common good. While we describe specific algorithms for a particular use case in collaborative production planning, their design technique is applicable to the most general set of distributed optimization problems.

Second, we secure the generation and selection of individuals in the population. Our protocols are provably secure not only for the cost function evaluation, but for the entire EA. We nevertheless stress that our proposal is not a straightforward application of the construction by Yao [29]. This would be impractical for complex real-world use case due to communication effort and memory consumption. Instead we use a combination of several techniques which might also be of interest for other complex applications than EA. In order to underpin the practicality of our approach we report the evaluation results of our prototypical implementation.

When designing privacy-preserving protocols one must balance three objectives: security, performance, and quality. This paper's main contribution is a privacy-preserving multi-objective evolutionary algorithm (PPMOEA) that improves over the state-of-the-art as follows

- it is more *secure* than previous proposals for PPEA [8, 26]. We provide a proof of security for the entire algorithm and not just cost function evaluation.

- it is more *efficient* than general SC [29]. We reduce communication complexity from $\mathcal{O}\left(\eta(T^2Pm^2\lambda + TPm\lambda\log^2\lambda)\right)$ to $\mathcal{O}\left(\eta(TPm^2\lambda + \lambda^2\log^2\lambda + m\lambda\log^2\lambda)\right)$.[1]

- it is almost as *effective* as NSGA-II [3], one of the best known MOEAs [16]. We chose algorithms that are more efficiently implementable securely,

[1] number of planning periods T; number of products P; bit size of production quantity m; number of offspring λ; number of generations η

	Producer			Supplier		
	p_1	p_2	p_3	m_1	m_2	m_3
Opportunity cost oc_p	200	400	600	600	400	200
Capacity consumption c_p	1	3	6	6	3	1
Storage costs product sc_p	20			20		
Storage costs material sc_m	20			-		
Market demand d	100			-		
Production capacity c_t	100			100		

Table 1: Parameters of the considered use case.

since performance is key for practicality, but experimentally quantify the penalty.

The remainder of the paper is structured as follows: Section 2 introduces the use case that motivates our PPMOEA. The cryptographic tools are introduced in Section 3. A detailed description of the construction of our PPMOEA is presented in Section 4. Afterwards, its performance and quality (Section 5) are evaluated. Related work is presented in Section 6. Section 7 concludes this paper with a summary of our findings and an outlook on future work.

2 Use Case

Our use case is a finite horizon two-echelon collaborative production planning problem [4], i.e. companies along a two-level supply chain that wish to jointly optimize their production planning for a bounded time period. We limit the two echelons of the supply chain to comprise one party each. On the upstream echelon, a supplier $\mathcal{S}$ provides raw materials to a manufacturer $\mathcal{P}$ on the downstream echelon. Both abide an exclusive relationship, i.e. $\mathcal{P}$ only procures materials from $\mathcal{S}$ and is furthermore $\mathcal{S}$‘s single source of revenue. Moreover, we assume periodical shipping with neglected transportation times [4].

As an example assume that $\mathcal{S}$ supplies three ($P = 3$) materials $m_1 \dots m_3$ to $\mathcal{P}$, who produces three consumer products $p_1 \dots p_3$, with one unit of p_i requiring one unit of m_i to produce. Table 2 shows the further parameters of our use case. Note the opposing opportunity costs oc_p and capacity consumptions c_p for the supplier‘s and producer‘s commodities. This opposition causes tension in the parties' planning objectives, since each party is inclined to act rationally, minimizing only its own cost.

Considered costs Let T denote the planning horizon, i.e. the number of periods, and P the total number of products involved. Let the matrix $\mathbf{A}$ denote a combined production plan for both parties with entries $a_{\mathcal{S}_{t,p}}$ and $a_{\mathcal{P}_{t,p}}$ specifying the produced quantity of product p of the supplier and producer in period t, respectively. Further, let

- $S(\mathbf{A})$ denote the warehousing costs of a production plan $\mathbf{A}$. We consider two kinds of warehousing costs: a) sc_p - warehousing costs for a *product* p and b) sc_m the producer's expenses for warehousing *material* m.
- $O(\mathbf{A})$ denote the opportunity costs for a party. These costs arise if a party produces less goods then demanded by the market. These lost earnings are quantified by oc_p for product p.
- $C(\mathbf{A})$ denote the penalty costs for exceeding the available production capacity c_t in period t. Producing one unit of product p demands c_p capacity. The cost cp accounts for excess costs, e.g., overtime bonuses.[2]
- $U(\mathbf{A})$ denote the penalty costs on the producer's echelon for consuming more material in a production plan than there is on stock for a particular product. The costs for procuring extra material are quantified by up.[2]

We assume the market demand d to be constant in the example, but a varying demand adds no complexity to our protocol.

As stated before, the supplier and producer pursue different objectives – minimizing their own cost – which are opposing due to the differing opportunity costs oc_p and capacity consumptions c_p. We thus compose the objective function $F(\mathbf{A})$ from the supplier's objective function $f_{\mathcal{S}}(\mathbf{A})$ and the producer's one $f_{\mathcal{P}}(\mathbf{A})$:

$$\min \vec{f} = F(\mathbf{A}) = \begin{pmatrix} f_{\mathcal{S}}(\mathbf{A}) \\ f_{\mathcal{P}}(\mathbf{A}) \end{pmatrix}.$$

Producer The producer's fitness function is a sum of the four costs

$$f_{\mathcal{P}}(\mathbf{A}) = \sum_{t=2}^{T} \left(C_t(\mathbf{A}) + \sum_{p=1}^{P} \left[S_{t,p}(\mathbf{A}) + O_{t,p}(\mathbf{A}) + U_{t,p}(\mathbf{A}) \right] \right). \qquad (1)$$

Due to space constraints, we only briefly present the details of the producer's warehousing cost and overcapacity penalty which exhibit all patterns later used in our optimizations. The producer has to account for both product and unused material warehousing. The warehousing cost $S_{t,p}$ for product p in period t is given by

$$\begin{aligned} S_{t,p}(\mathbf{A}) = sc_p \max &\left(0, \sum_{i=2}^{t} a_{\mathcal{P}_{i,p}} - \sum_{i=2}^{t} d_{i,p} \right) \\ + sc_m \max &\left(0, \sum_{i=1}^{t-1} a_{\mathcal{S}_{i,p}} - \sum_{i=2}^{t} a_{\mathcal{P}_{i,p}} \right). \end{aligned} \qquad (2)$$

The penalty costs C_t for excessive capacity usage in period t are

$$C_t(\mathbf{A}) = cp \max \left(0, \sum_{i=1}^{P} (c_p a_{\mathcal{P}_{t,i}}) - c_t \right). \qquad (3)$$

[2] Costs cp and up are parameters of the EA and not specific to the use case. For their values see Section 5.

Supplier The supplier is not constrained by a further upstream party and therefore no costs for excessive material consumption $U(\mathbf{A})$ arise. Its fitness function is consequently a composition of only three costs

$$f_{\mathcal{S}}(\mathbf{A}) = \sum_{t=1}^{T-1} \left(C_t(\mathbf{A}) + \sum_{p=1}^{P} [S_{t,p}(\mathbf{A}) + O_{t,p}(\mathbf{A})] \right) . \tag{4}$$

Using our exemplary numbers one can see how the traditional, decentralized production planning may lead to sub-optimal solutions [9]. In current business practice production planning is performed upstream: The producer generates a locally optimal production plan $\mathbf{A}_{\mathcal{P}}$ and orders materials accordingly. The supplier then strives to fulfill this order by a locally optimal production plan $\mathbf{A}_{\mathcal{S}}$. If the order cannot be completely satisfied, e.g. due to capacity constraints, the producer has to execute on a partial shipment, leading to a suboptimal production plan $\mathbf{A}'_{\mathcal{P}}$.

In our example, the producer incurs the lowest costs when producing 100 units of product p_1, resulting in an optimal capacity utilization of 100% and cost $f_{\mathcal{P}}(\mathbf{A}_{\mathcal{P}}) = 4\,500\,000$. He would then send an order to the supplier, but due to capacity constraints, $\mathcal{S}$ can merely supply 16 units of material m_1. As a consequence, the producer's costs are increased to $f_{\mathcal{P}}(\mathbf{A}'_{\mathcal{P}}) = 5\,256\,000$ with only 16% capacity utilization. The supplier's costs $f_{\mathcal{S}}(\mathbf{A}_{\mathcal{S}}) = 2\,268\,000$ add to the overall supply chain cost. Would they have instead used a collaborative planning approach, they would have attained the globally optimal production plan $\mathbf{A}$,

$$\mathbf{A} = \begin{array}{ccc|ccc} m_1 & m_2 & m_3 & p_1 & p_2 & p_3 \\ \hline 1 & 31 & 1 & 1 & 31 & 1 \end{array} .$$

This plan $\mathbf{A}$ yields 100% capacity utilization for both parties, producer's cost $f_{\mathcal{P}}(\mathbf{A}) = 4\,806\,000$ and supplier's cost $f_{\mathcal{S}}(\mathbf{A}) = 0$.[3] The overall supply chain costs are reduced by 40%.

While this greatly cost-reducing concept is long known, most companies are still very reluctant to share the necessary, sensitive data [21]. However, the improvements are not derived from the information sharing per se, but rather from the therefore improved decision making. SC allows for decision making upon a comprehensive data set, without jeopardizing the privacy of the parties contributing to this set.

3 Cryptographic Tools

3.1 Secure Computation

SC was introduced by Yao [29]. The problem is as follows: Two players, Alice and Bob, both know a joint function $y = f(a, b)$ on their combined input a

[3] The supplier incurs no cost, since a) he does not have to warehouse any goods $S(\mathbf{A}) = 0$, b) his capacity limit c_t is not exceeded $C(\mathbf{A}) = 0$, and c) the producer's orders are fulfilled $O(\mathbf{A}) = 0$.

(Alice) and b (Bob). They are both interested in the result y, but neither is willing to reveal its input. Note that a party may infer information about the other party's input based on y and their own input. This has to be accepted and is excluded from the security definition. Yao [29] constructed a protocol that achieves this for any function f.

Yao's protocol roughly proceeds as follows. Alice and Bob encode the function f as a Boolean circuit. Then Alice encrypts the circuit for each possible input of Bob. She ships the encrypted circuit to Bob which then also obtains the keys for his input using a technique called Oblivious Transfer (OT) [6]. OT ensures that Bob does not have to reveal his input.

Yao's protocol has been implemented [22] using a high-level programming language, but its general construction is too inefficient for complex problems as our PPMOEA. Instead we construct the necessary (optimized) circuits manually and apply further optimizations based on secret sharing and homomorphic encryption.

3.2 Secret Sharing

Let s be a secret known to neither Alice nor Bob. Both, Alice and Bob, hold a value (called share) $s^{(A)}$ and $s^{(B)}$, respectively, such that $s = f(s^{(A)}, s^{(B)})$ for some reconstruction function f, e.g. $s = s^{(A)} + s^{(B)} \bmod n$ [12]. A secret sharing is perfect if any share $s^{(A)}$ or $s^{(B)}$ does not reveal additional information about the secret s: $Pr(s) = Pr(s|s^{(A)}) = Pr(s|s^{(B)})$.

3.3 Homomorphic Encryption

Homomorphic encryption is a modern encryption scheme where one operation on the ciphertexts produces an encryption of the result of a homomorphic operation on the plaintexts. In particular, we require the homomorphic operation to be addition (modulo a key-dependent constant). We used Paillier's encryption system [23] in the implementation. Let $E_X(x)$ denote the encryption of x with X's public key and $D_X(\cdot)$ the corresponding decryption with X's private key, then Paillier's encryption system has the following property:

$$D_X\left(E_X(x) \cdot E_X(y)\right) = x + y$$

With simple arithmetic the following property can be derived

$$D_X\left(E_X(x)^y\right) = x \cdot y$$

4 The Privacy-Preserving MOEA

4.1 Algorithm Overview

We follow the conventional structure of an EA as outlined in Algorithm 1. The initial population consists of μ random individuals (operation random).

Algorithm 1 Privacy-preserving MOEA

```
$\mathbb{A} \leftarrow \left\{ \forall i \leq \mu : \mathbf{A}_i \leftarrow \text{random}^{T \times P} \right\}$
**for** $\forall j \leq \eta$ **do**
    $\mathbb{O} \leftarrow \left\{ \forall i \leq \lambda : \mathbf{O}_i \leftarrow \text{mutate}\left( \mathbf{A}_{\lfloor \frac{i}{\lceil \frac{\lambda}{\mu} \rceil} \rfloor} \right) \right\}$
    $\mathbb{F} \leftarrow \left\{ \forall \mathbf{O} \in \mathbb{O} : \vec{f_i} \leftarrow F(\mathbf{O}) \right\}$
    $\mathbb{A} \leftarrow \begin{cases} \text{select}\,(\mathbb{O}, \mathbb{F}) & \textit{if non-elitist} \\ \text{select}\,(\mathbb{O} \cup \mathbb{A}, \mathbb{F}) & \textit{if elitist} \end{cases}$
**end for**
**return** $\mathbb{A}$
```

	Yao	**PPMOEA**
random	$TPm\mu$	0
mutate	$TPm\lambda$	0
F	$T^2Pm^2\lambda$	$TPm^2\lambda$
select	$TPm\lambda \log^2 \lambda$	$\lambda \log^2 \lambda(m + \lambda) + \mu(TPm + \lambda)$
Total per generation	$T^2Pm^2\lambda + TPm\lambda \log^2 \lambda$	$TPm^2\lambda + \lambda^2 \log^2 \lambda + m\lambda \log^2 \lambda$

Table 2: Communication complexity ($\mathcal{O}(\cdot)$)

Every individual produces $\lceil \frac{\mu}{\lambda} \rceil$ offspring (operation mutate), perturbing each individual with a small probability p_m by a Gaussian distributed value. We solely use mutation in our use case, but believe that other reproduction schemes including recombination are securely realizable with little additional effort. We then compute the offspring's fitness (operation F) and finally select μ survivors (operation select) either from the offspring only (non-elitist) or including the parent population (elitist) using a Pareto-optimal selection algorithm suitable for a multi-objective problem. The evolution iterates for a fixed number η of generations.

Our PPMOEA realizes each operation privacy-preservingly and not only cost function evaluation as previous PPEA. Furthermore all privacy-preserving operations are tied together, such that not even the result of any single operation will be known to any party, but only the result of the entire algorithm, i.e. the optimal production plan, will be revealed to both parties. We emphasize that the parties gain no sensitive information from this production plan, since they need this information to schedule orders and shipments.

In the remainder of the paper we describe the privacy-preserving implementation of the operations, but we start by explaining how we tie the individual operations in Section 4.2, such that no intermediate results are revealed. The principle we use is not limited to inter-operation use, but we also apply it in optimizing the operations. We will then describe the privacy-preserving realization of the operations in the following sections.

4.2 Construction of Secure Protocol

Each operation of the MOEA can be abstractly written as a function

$$y = f(x)$$

e.g. $\mathbb{A} = \mathrm{select}\,(\mathbb{O}, \mathbb{F})$. Using Yao's algorithm we could construct a privacy-preserving protocol between Alice (Supplier) and Bob (Producer) for any operation, but the results would be revealed and reused as inputs in the next operation. Instead we use secret sharing as follows in order to conceal the results.

We maintain the following invariant as pre- and post-condition of each step: Each variable x or y, i.e. input and output, are distributed as secret shares across Alice and Bob. For our basic data type of integer Alice has $x^{(A)}$ and Bob has $x^{(B)}$, such that $x = x^{(A)} + x^{(B)}$ (we omit the modulus for clarity) [12]. Our other data types, such as vectors and matrices, are simple compositions of integers and if the secret sharing of integers is perfect, their secret sharing is perfect as well. We can now write

$$y^{(A)} + y^{(B)} = f\left(x^{(A)} + x^{(B)}\right)$$

We intend to realize the function f using Yao's protocol which can only implement deterministic functions. However, in order for the security of the secret sharing to hold, the shares need to be chosen randomly. We therefore transform above equation to

$$y^{(B)} = f\left(x^{(A)} + x^{(B)}\right) - y^{(A)}$$

and define this as a new function f'

$$y^{(B)} = f'\left(x^{(A)}, y^{(A)}, x^{(B)}\right)$$

We now implement this function f' using Yao's protocol, such that only Bob will learn the result. In order to obtain the final result Alice and Bob simply exchange shares.

Our secret sharing scheme is linear, and linear operations, such as additions or multiplications with constants, can be performed locally on the shares. If Alice wants to add a number to a variable, she can simply add this number to her local share. No communication with Bob is required. Therefore this decomposition into smaller protocols may reduce the communication complexity if used cleverly, since otherwise the addition would have to be performed as part of the encrypted circuit in Yao's protocol. Table 4.1 gives details of the reduced communication complexity in our PPMOEA by application of the decomposition technique. We also reduce the memory consumption, since the size of the circuits in main memory is reduced. Both are important aspects of making secure computation protocols efficient.

4.2.1 Security Proof

We prove the security of our decomposition in the semi-honest model of Goldreich [7]. Loosely speaking, in the semi-honest model, parties follow the protocol, but keep a record of all messages in order to infer additional knowledge (passive attackers). This model very well corresponds to the motivation of the players in our business use case. For a security proof in this model we need to show the existence of a simulator that using local input $x^{(B)}$ and output $y^{(B)}$ generates messages – a view – that are computationally indistinguishable from the messages received during a real protocol execution. Assume we sequentially compose two functions f and g as described above.

$$\begin{aligned} y^{(B)} &= f\left(x^{(A)}, y^{(A)}, x^{(B)}\right) \\ z^{(B)} &= g\left(y^{(A)}, z^{(A)}, y^{(B)}\right) \end{aligned}$$

Let $h = f \circ g$ be the sequential composition.

Theorem 1. *Let $h = f \circ g$ be the sequential composition of two function f and g. If both f and g are implemented securely as described above, the combined protocol securely implements h in the semi-honest model.*

We stress that our composition theorem differs from Goldreich's in that we prove a secure protocol for h from secure protocols for f and g instead of assuming there exists already a secure protocol for h (with either f or g replaced). We write $VIEW_A^f$ for the view of Alice in the protocol for f and S_A^f for a simulator of this view. We write $VIEW_A^f \sim S_A^f$ if they are computationally indistinguishable.

Proof. From using Yao's protocol we know that there exist $S_A^f \sim VIEW_A^f$, $S_A^g \sim VIEW_A^g$, $S_B^f \sim VIEW_B^f$ and $S_B^g \sim VIEW_B^g$. In case of Alice's view, observe that Alice does not obtain any output in either f or g and therefore the views of the protocols are independent. We construct the simulator S_A^h simply as a sequential composition of S_A^f and S_A^g. If one can distinguish S_A^h and $VIEW_A^h$, one can either distinguish S_A^f and $VIEW_A^f$ or S_A^g and $VIEW_A^g$. Bob receives $y^{(B)}$ in addition to the views $VIEW_B^f$ and $VIEW_B^g$. The key insight is that $y^{(B)}$ can be simulated independently by a uniform random source due to the perfect secret sharing with $y^{(A)}$ which remains unknown to Bob. Therefore there exists a simulator $S_B^g(z^{(B)})$ which produces a computationally indistinguishable output from $S_B^g(y^{(B)}, z^{(B)})$ which simulates the view during Yao's protocol. Since this simulator is independent from the simulator for protocol for f, the same arguments as for Alice hold. □

4.3 Population

Before we can describe the first privacy-preserving realizations of the initialization (operation random) and reproduction (operation mutate), we must describe the representation of the population. This representation must always be use case specific. Integer genotypes as necessitated by cryptographic techniques are well-suited for production planning problems, although little research has been conducted on the field of integer-based EA (see Section 6). Let $a_{t,p}$ denote the produced quantity of product p in period t. An individual is given by matrix

$$\mathbf{A} = \left(\begin{array}{cccccc} \multicolumn{3}{c}{\overbrace{\hphantom{a_{1,1} \quad \cdots \quad a_{1,P_\mathcal{S}}}}^{\text{Supplier}}} & \multicolumn{3}{c}{\overbrace{\hphantom{a_{1,P_\mathcal{S}+1} \quad \cdots \quad a_{1,P_\mathcal{S}+P_\mathcal{P}}}}^{\text{Producer}}} \\ a_{1,1} & \cdots & a_{1,P_\mathcal{S}} & a_{1,P_\mathcal{S}+1} & \cdots & a_{1,P_\mathcal{S}+P_\mathcal{P}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{T,1} & \cdots & a_{T,P_\mathcal{S}} & a_{T,P_\mathcal{S}+1} & \cdots & a_{T,P_\mathcal{S}+P_\mathcal{P}} \end{array} \right).$$

Individuals are the main inputs and outputs of the operations in our PPMOEA. Therefore they are maintained as secret shares throughout the entire algorithm and each party only possesses a share of every individual, denoted by $\mathbf{A}^{(A)}$ and $\mathbf{A}^{(B)}$.

Initialization Supplier and producer independently generate the initial population, i.e. μ local production plans. Each party chooses initial production quantities for their own products randomly from a uniform distribution. They use these values to initialize their shares for these parts of the individual and simply initialize the shares for the other party's plan with 0. No communication is required for this step and it results in a perfect secret sharing of the random, initial population.

Reproduction Reproduction may consists of mutation and recombination. We restrict ourselves to mutation, due to the high probability that in our use case the combination of two individuals results in prohibitively high penalty costs for excessive material consumption $U(\mathbf{A})$. We believe however, that crossover is suitable for privacy-preserving recombination. Both parties can either choose crossover points independently, select one publicly or perform the crossover entirely using Yao's protocol.

In our PPMOEA, *each* individual produces $\lceil \frac{\lambda}{\mu} \rceil$ offspring resulting in an ancestor population of λ individuals. We apply Gaussian perturbation to every production quantity with probability p_m [10]

$$\forall t, p: \; a'_{t,p} = a_{t,p} + \Delta \quad \Delta = \begin{cases} \lfloor \mathcal{N}(0, \sigma^2) \rceil & Pr = p_m \\ 0 & Pr = (1 - p_m) \end{cases}.$$

Stochastic rounding ($\lfloor \Delta \rceil$) ensures an unbiased tie-breaking if the fractional part of Δ is .5 [17].

We can use the linearity of the secret sharing scheme in order to realize this operation (almost) without any communication. For each production quantity a party is chosen that applies the perturbation to its share only. The other party does not modify its share. In order to prevent certain active attacks, such as enforcing its locally optimal production plan, we randomly choose the party to perform the perturbation using a pseudo-random function based on a jointly chosen seed.

The step size of the mutation is set by the standard deviation σ^2 of the normal distribution. This step size strongly influences the convergence of the evolution and can either be static throughout all generations or be adapted according to some predefined schedule, commonly producing better results [5]. Our PPMOEA can implement both and we compare the results in our evaluation.

4.4 Fitness Evaluation

As noted in the other proposals for PPEA fitness computation is the most sensitive operation. Since it is a simple arithmetic computation, we can implement it straightforwardly using Yao's protocol. Nevertheless in order to speed up the protocol, we employ a number of optimizations over simply implementing $F(\mathbb{A})$ as a circuit.

First note that $F(\mathbb{A})$ can be easily decomposed into λ computations of the fitness $F(\mathbf{A})$ of each individual. The same circuit can be used for each invocation of $F(\mathbf{A})$ and, since all invocations are using independent inputs and outputs, we can run them in parallel. Parallel execution allows computation and communication of different invocations to overlap, maximizing the CPU utilization of each party.

Furthermore we use the linearity of the secret sharing in order to reduce the size of the circuit that has to be computed using Yao's protocol. The size of the circuit dominates the communication cost. Due to space limitations we exemplify our techniques on the previously presented formulas for warehousing cost $S_{t,p}(\mathbf{A})$ as well as capacity penalty cost $C_t(\mathbf{A})$.

Precomputation Note that the innermost terms of the warehousing costs $S_{t,p}(\mathbf{A})$ are summations. These sums need to be computed first and can be precomputed before the input to Yao's protocol on the secret shares. Let $\mathcal{S}_{t,p}$ denote the accumulated production quantities for product p of the supplier in period t, $\sum_{i=1}^{t} a_{S_{i,p}}$, and $\mathcal{P}_{t,p}$ that of the producer: $\sum_{i=2}^{t} a_{P_{i,p}}$. Furthermore, let $\mathcal{D}_{t,p}$ be the aggregated demand of product p: $\sum_{i=2}^{t} D_{i,p}$.

We can then use the precomputed sums as input for Yao's protocol for fitness evaluation. This optimization does not only reduce the communication complexity to $\mathcal{O}(Pm^2)$ but also significantly reduces the constants hidden by the "big-O" notation.

Term rewriting The sums are not only used in the warehousing costs, but almost all cost types of our fitness function. Obviously we can reuse the precomputed terms for all cost types, but we exploit a further similarity.

The warehousing cost $S_{t,p}(\mathbf{A})$ contains a lower bounded difference of the precomputed sums: $\max(0, \sum_{i=2}^{t} a_{\mathcal{P}_{i,p}} - \sum_{i=2}^{t} d_{i,p})$. The opportunity cost $O_{t,p}(\mathbf{A})$

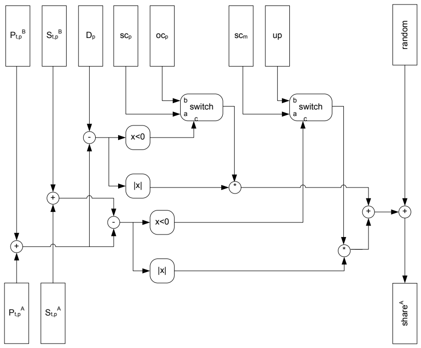


Figure 1: Schematic diagram for f_P - Producer stock, opportunity and penalty costs.

contains the lower bounded negative of this difference: $\max(0, \sum_{i=2}^{t} d_{i,p} - \sum_{i=2}^{t} a_{\mathcal{P}_{i,p}})$.

As a result, only either the warehousing cost or the opportunity cost contributes to the fitness; the other one being 0. We can therefore implement the circuit by computing the difference only once, comparing it to 0 and then selecting the appropriate cost. The same applies for material cost and deficit penalty cost.

Using both optimizations Equation (1) for the fitness function can be rewritten as

$$f_{\mathcal{P}}(\mathbf{A}) = \sum_{t=2}^{T} \left(C_t(\mathbf{A}) + \overbrace{\sum_{p=1}^{P} \left[\varphi_{t,p} |\mathcal{P}_{t,p} - \mathcal{D}_{t,p}| + \varkappa_{t,p} |\mathcal{P}_{t,p} - \mathcal{S}_{t-1,p}| \right]}^{\psi_{t,p}(\mathbf{A})} \right)$$

$$\varphi_{t,p} = \begin{cases} sc_p, & \mathcal{P}_{t,p} - \mathcal{D}_{t,p} > 0 \\ oc_p, & \mathcal{P}_{t,p} - \mathcal{D}_{t,p} < 0 \end{cases} \qquad \varkappa_{t,p} = \begin{cases} up, & \mathcal{P}_{t,p} - \mathcal{S}_{t-1,p} > 0 \\ sc_m, & \mathcal{P}_{t,p} - \mathcal{S}_{t-1,p} < 0. \end{cases} \tag{5}$$

Figure 1 depicts our circuit for $\psi_{t,p}(\mathbf{A})$. Let m denote the bit size of one production quantity. The circuit consists of $12m^2 + 64m - 10$ gates. For $m = 32$ bits this amounts to 14 326 gates. By term rewriting we reduced the number of expensive ($\mathcal{O}(m^2)$) multiplications from 4 to 2.

The precomputation of the sums does not apply to the capacity penalty cost $C_t(\mathbf{A})$, since its innermost term $c_p a_{P_{t,i}}$ is a product of two sensitive values that should be kept private. The resulting circuit's size grows with the number of products P and has $6Pm^2 + 7Pm - 5P + 12m^2 + 32m + 6$ gates. For $m = 32$ bits and $P = 6$ products the circuit has 51 496 gates. We show the impact of the lack of precomputation in our evaluation.

For the fitness computation of the entire population $\mathcal{O}(\lambda T)$ parallel invocations of Yao's protocol with circuits containing $\mathcal{O}(Pm^2)$ gates and $\mathcal{O}(\lambda TP)$

parallel invocations with $\mathcal{O}(m^2)$ gates are required with a total communication complexity of $\mathcal{O}(\lambda T P m^2)$.

4.5 Survivor Selection

The selection of the μ “best” individuals from the population is the final step for every generation. The selected individuals serve as parent population for the subsequent generation. Multi-objective optimization problems (MOPs) require a more complex notion of “better” than single objective ones. Classical methods scalarize the MOP by some (weighted) objective combination scheme, thus requiring knowledge about the relative magnitudes of the parties' fitness functions [28]. This is not desirable in a privacy-preserving setting, since it leaks information, and we therefore embrace Pareto's notion of “better” [2, p.10 et. seq.].

A vector $\vec{x} \in \mathbb{R}^n$ is called superior to, or not dominated by, $\vec{y} \in \mathbb{R}^n$ if it satisfies the Pareto-criterion:

$$\vec{x} \preceq \vec{y} := \forall i : x_i \leq y_i \wedge \exists i \leq n : x_i < y_i.$$

In a set of vectors Ω, the subset $\Omega' := \{\vec{x} \mid \nexists \vec{y} \in \Omega : \vec{y} \preceq \vec{x}\}$ is called the Pareto-optimal set and contains the optimal tradeoffs to the given problem. Pareto-sorting requires no domain knowledge and is independent from the different orders of magnitude of the single objectives.

Pareto-sorting is a rather complex task that can even dominate the cost without protection of privacy. The most efficient, non-privacy-preserving algorithms known are $\mathcal{O}(\lambda^2)$ [3]. We therefore choose to implement a traditional sorting algorithm for strictly ordered sets that is only likely to produce a Pareto-sorting. Subsequent to the sorting the non-dominated individuals are then likely occupying the first θ ranks. The first μ individuals are selected for survival; if $\theta > \mu$ optimal individuals will be lost, if $\theta < \mu$ non-optimal ones will survive in order to keep a steady-state population.

Sorting For a privacy-preserving realization we require a sorting algorithm that is *oblivious* to the outcome of the performed comparisons, i.e. the operations performed after comparing x_i to x_j are exactly the same for the case $x_i < x_j$ and $x_i > x_j$. Sorting networks [19, p. 228 et. seq.] are an ideal candidate with this property. A sorting network is a sequence of compare-and-exchange (CX) operations to sort any given input $x_1 \ldots x_\lambda$. A sorting network is characterized by two properties: *size* refers to the total number of CX operations; *depth* denotes the maximum number of comparators a x_i has to pass through. The most efficient practical sorting networks follow the odd-even merger construction of Batcher [1], resulting in networks of size $\mathcal{O}(\lambda \log^2 \lambda)$ and depth $\mathcal{O}(\log^2 \lambda)$ [1, 19, p. 228 et. seq.].

A CX operation compares two fitness values x and y (using the Pareto criterion) and if necessary exchanges their position, such that the non-dominated one occupies the lower rank. We decompose the sorting network by secret sharing and only implement each CX operation as an invocation of Yao's protocol using the same circuit. The CX operations on one level of the network can be performed in parallel.

Our use case is a two objective problem. We implement the Pareto criterion for the two fitness vectors $\vec{f_x}$ and $\vec{f_y}$ with the following logical formula: $(f_{x_1} < f_{y_1} \wedge f_{x_2} \leq f_{y_2}) \vee (f_{x_1} \leq f_{y_1} \wedge f_{x_2} < f_{y_2})$. We further optimize our integer comparison, such that it produces two output bits: one for the $<$ and one for the $=$ case. The circuit then has only $10m + 12$ gates.

The purpose of privately implementing sorting is to privately implement selection, i.e. the parties should not learn which individuals survive, since this reveals significant information about the private cost functions. Consequently not only the fitness values need to be sorted privately, but also the corresponding individuals. Unfortunately the size (number of bits) of an individual is quite large ($TPm \gg \lambda$), significantly increasing the communication complexity of a CX operation. We therefore realize another optimization by a level of indirection using index variables. An index variable i is a λ-bit string with only one bit $i[\kappa]$ set. At the start of the sorting this bit κ is the index of individual x in the original, unsorted population, hence index variable. After the sorting the lower ranked index variables contain the indices of the "better" individuals.

The index variables are secretly shared bitwise (modulo 2) between Alice and Bob, such that $i = 2^\kappa = i^{(A)} \oplus i^{(B)}$. We then only sort and exchange the index variables in a CX operation instead of the entire individuals. This reduces the communication complexity for sorting from $\mathcal{O}(TPm)$ to $\mathcal{O}(\lambda)$, but adds another subsequent selection operation using the index variables and individuals which we describe next. Each CX circuit then has only $66m + 10\lambda + 10$ gates.

The total communication complexity of the sorting network following Batcher's construction is $\mathcal{O}(\lambda \log^2(\lambda)(m + \lambda))$.

Selection Subsequent to the execution of the sorting network, the indices of the μ best individuals are likely to reside in ranks $1 \ldots \mu$. A separate protocol is required for using an index variable to select the corresponding individual. Implementing this straightforwardly as a circuit in Yao's protocol completely annihilates the advantage from using index variables in the first place. Instead we employ a more sophisticated technique.

Alice and Bob secretly share index variable $i = 2^\kappa = i^{(A)} \oplus i^{(B)}$ and population $\mathbb{A} = \mathbb{A}^{(A)} + \mathbb{A}^{(B)}$. We present a protocol to select Alice's share $\mathbf{A}_\kappa^{(A)}$. Later, this protocol is run again with the roles of Alice and Bob interchanged selecting Bob's share $\mathbf{A}_\kappa^{(B)}$. This pair of protocols is run μ times, once for each survivor.

Alice chooses a public, private key-pair $E_A(\cdot)$ in Paillier's homomorphic encryption scheme. She encrypts each bit $i^{(A)}[j]$ of her share of the index variable i and sends all ciphertexts to Bob.

Bob performs the following algorithm: For each share $\mathbf{A}_j^{(B)}$ of an individual he reconstructs the (unshared) index variable bit $i[j]$ in homomorphically encrypted form ($E_A(i[j])$) by using his plaintext share of the index variable (recall that these bits are shared "exclusive-or" while modulus of the encryption is a

Protocol 2 Survivor selection with homomorphic encryption

Input: λ encrypted index variable bits $E_A(i^{(A)}[j])$ $(i = 2^\kappa)$
Output: Encrypted share $E_A(s^{(A)})$ of the individual selected by i

```
$R_B \leftarrow (\mathcal{U}(0, q))^{T \times P}$
for $j \leq \lambda$ do
  if $i^{(B)}[j] = 1$ then
    $E_A(i[j]) \leftarrow E_A\left(i^{(A)}[j]\right)^{-1} \cdot E_A(1)$
      $= E_A\left(-i^{(A)}[j] + 1\right)$
  else
    $E_A(i[j]) \leftarrow E_A\left(i^{(A)}[j]\right)$
  end if $\{i[j] \mathrel{\widehat{=}} i^{(A)}[j] \oplus i^{(B)}[j];\ i = 2^\kappa\ \kappa \in [0, \lambda]\}$
  $O_j \leftarrow \mathbf{A}_j - R_B$
  $E_A(c_j) \leftarrow E_A(i[j])^{O_j} = E_A\left(i[j] \cdot O_j\right)$
end for$\{E_A(c_\kappa) = E_A(\mathbf{A}_\kappa - R_B); \forall j \neq \kappa : E_A(c_j) = E_A(\mathbf{0})\}$
$E_A(s^{(A)}) \leftarrow \prod_{j=1}^{\lambda} E_A(c_j) = E_A\left(\sum_{j=1}^{\lambda}\left(i[j] \cdot O_j\right)\right)$
```

product of two large primes). If the individual is the selected one, i.e. $j = \kappa$, then he now has an encryption of 1 ($E_A(1)$), otherwise of 0 ($E_A(0)$). In our decomposition technique all shares need to be randomized by an uniform R_B, which Bob can subtract from his plaintext share. Bob then homomorphically multiplies the encrypted index bit with the plaintext of his randomized share of the individual. Since only one index bit is set, the (homomorphic) sum of products of all his shares equals the share of the selected individual. Bob sends the (encrypted) share to Alice. The detailed steps are shown in Protocol 2.

After the protocol Alice has $s^{(A)} = D_A(E_A(s^A))$ and Bob has R_B, such that $\mathbf{A}_\kappa^{(B)} = s^{(A)} + R_B$. Alice and Bob then run the same protocol with the roles of Alice and Bob interchanged. Alice obtains R_A and Bob $s^{(B)}$ ($\mathbf{A}_\kappa^{(A)} = s^{(B)} + R_A$). They compute new shares $\mathbf{A}_\kappa'^{(A)} = s^{(A)} + R_A$ and $\mathbf{A}_\kappa'^{(B)} = s^{(B)} + R_B$ of the selected individual, respectively. They have obtained a fresh, perfect secret sharing

$$\mathbf{A}_\kappa'^{(A)} + \mathbf{A}_\kappa'^{(B)} = s^{(A)} + R_A + s^{(B)} + R_B = \mathbf{A}_\kappa^{(B)} + \mathbf{A}_\kappa^{(A)} = \mathbf{A}_\kappa.$$

Note that Bob only receives ciphertexts as messages and no output and Alice's view corresponds to our decomposition technique. Therefore the security proof for this protocol does not significantly differ from that of our decomposition technique.

The execution of the protocol pairs can be parallelized. All μ invocations of the first protocol of each pair are performed concurrently, afterwards the μ second protocols are performed concurrently. The total communication complexity for this selection is $\mathcal{O}(\mu(\lambda + TPm))$.

5 Evaluation

We experimentally evaluated the solution quality and performance of our PPMOEA in comparison to NSGA-II [3]. Security is ascertained by proof (Section 4.2.1).

Solution quality For evaluation of solution quality we follow the proposals from literature and adopt four of the metrics proposed by Zitzler et al. [30]. Let $\mathbb{F}$ denote the set of the final population‘s fitness vectors. The average distance of each individual in $\mathbb{F}$ to the true Pareto front of the problem is measured by metric $\mathcal{M}_1^*(\mathbb{F})$. A low value indicates good convergence of the evolution. Metric $\mathcal{M}_3^*(\mathbb{F})$ is an indicator for the spread of the identified Pareto-optimal set by computing the maximal distance between any two solutions in $\mathbb{F}$. Another metric for the spread of a population and its distribution along the identified Pareto-front is $\mathcal{S}(\mathbb{F})$. It measures the size of the space covered by population $\mathbb{F}$. For both metrics a high value is desirable, as it indicates a diverse population. To compare the relative quality of two fitness vector sets $\mathbb{F}'$ and $\mathbb{F}''$, Zitzler et al. [30] propose metric $\mathcal{C}(\mathbb{F}', \mathbb{F}'')$ as the number of individuals in $\mathbb{F}''$ which are dominated by or equal to an individual in $\mathbb{F}'$. The better population is characterized by less dominated solutions. All metrics are normalized to interval $[0, 1]$ and displayed in box plots which identify minium, maximum, median, first and third quartile [30].

In our experiments we varied population sizes (μ and λ) and number of generations (η). Furthermore we ran different variants of the algorithm by using elitist ($\mu + \lambda$) and non-elitist (μ, λ) selection schemes as well as static and adaptive mutation step sizes and penalty factors (σ^2, cp, up). All reported results are the average of 10 runs of each algorithm.[4] In summary our experiments cover 80 different parameter configurations:

$$\overbrace{\begin{pmatrix} 50 \\ 100 \\ 500 \\ 1000 \\ 2000 \end{pmatrix}}^{\eta} \times \overbrace{\begin{pmatrix} (2;5) \\ (3;6) \\ (5;35) \\ (10;50) \end{pmatrix}}^{(\mu;\lambda)} \times \begin{pmatrix} (\mu,\lambda) \\ (\mu+\lambda) \end{pmatrix} \times \begin{pmatrix} \sigma^2{=}1, \\ cp{=}1000, \\ up{=}10000 \\ \hline \sigma^2{=}5 \ldots 1, \\ cp{=}10 \ldots 5000, \\ up{=}100 \ldots 20000 \end{pmatrix} \tag{6}$$

We first analyzed the parameter choices for our PPMOEA by evaluating only its metric scores. Due to space constraints we cannot report detailed figures, but our experiments indicate that 500 iterations are a good trade-off between population convergence and algorithm runtime in all tested population sizes. Not much solution quality is gained by increasing the number of iterations, whereas less generations (100) – while still producing a good median convergence for the larger population sizes ($(5; 35)$ and $(10; 50)$) – result in a significantly higher variance of metric $\mathcal{M}_1^*(\mathbb{F})$. Elitist ($\mu + \lambda$) and non-elitist ($\mu, \lambda$) selection produce similar scores for metric $\mathcal{M}_1^*(\mathbb{F})$ (within the margin of error). However, elitism consistently resulted in a slightly lower score for metrics $\mathcal{M}_3^*(\mathbb{F})$ and

[4] NSGA-II always uses elitist selection.

Test case	Parameter settings
1	$\eta = 100$ $(\mu, \lambda) = (5; 35)$ $\sigma^2 = 1$ $cp = 1000$ $up = 10000$
2	$\eta = 500$ $(\mu, \lambda) = (5; 35)$ $\sigma^2 = 1$ $cp = 1000$ $up = 10000$
3	$\eta = 500$ $(\mu, \lambda) = (5; 35)$ $\sigma^2 = 5 \dots 1$ $cp = 10 \dots 5000$ $up = 100 \dots 20000$
4	$\eta = 1000$ $(\mu, \lambda) = (5; 35)$ $\sigma^2 = 1$ $cp = 1000$ $up = 10000$

Table 3: Test cases for in-depth comparison of NSGA-II and PPMOEA

	Prep	Comp	Comm	Sync	Other	**Total**	
Initialization	-	0.02	-	0.02	15.01	**15.08**	**0.49%**
Mutation	-	0.03	-	-	0.01	**0.04**	**0.00%**
Fitness	0.08	308.94	166.72	1 468.84	809.93	**2 754.51**	**89.74%**
Sorting	0.01	100.26	59.48	0.91	98.46	**259.11**	**8.44%**
Selection	-	16.42	18.14	-	6.05	**40.61**	**1.32%**
Total	**0.09**	**425.67**	**244.34**	**1 469.76**	**929.47**	**3 069.35**	
	0.00%	**13.87%**	**7.96%**	**47.89%**	**30.28%**		*values in [s]*

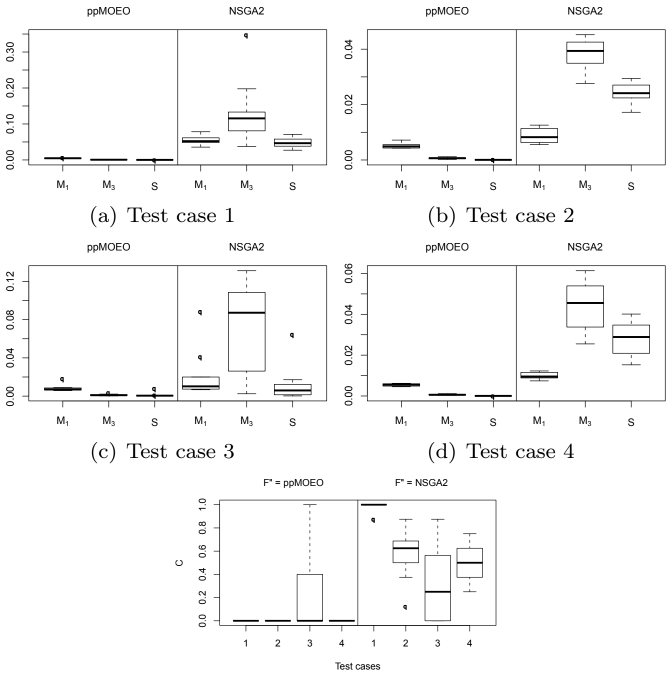
Table 4: Algorithm runtime broken down into preparation, computation, communication and synchronization.

$\mathcal{S}^*(\mathbb{F})$. Therefore we consider only non-elitist selection for our PPMOEA in the remainder of the paper.

From the entire parameter space we selected four configurations that broadly cover the range of solution quality of our PPMOEA (see Table 5). In these four configurations we compared our PPMOEA to NSGA-II. Figures 2(a) through 2(d) show the metrics $\mathcal{M}_1^*(\mathbb{F})$, $\mathcal{M}_3^*(\mathbb{F})$ and $\mathcal{S}(\mathbb{F})$ for both algorithms for the respective test cases. Figure 2(e) shows the metric $\mathcal{C}(\mathbb{F}', \mathbb{F}'')$ for all test cases. In all four test cases our PPMOEA has better convergence but poorer spread and distribution than NSGA-II. In metric $\mathcal{M}_1^*(\mathbb{F})$ our PPMOEA outperforms NSGA-II by a factor of 3.75, whereas NSGA-II outperforms PPMOEA by a factor of 124.87 and 421.83 in metrics $\mathcal{M}_3^*(\mathbb{F})$ and $\mathcal{S}(\mathbb{F})$. NSGA-II considers the density of the population surrounding an individual in its selection algorithm [3]. We believe that the lack of such a diversity operator in PPMOEA explains the poor spread of the identified solutions, but also contributes to the slightly better convergence due to the more focused evolution.

Runtime Our runtime measurements are based on test case 3 and are performed on two servers connected by Gigabit Ethernet, each equipped with four dual-core 2.6GHz CPUs and 16GB of main memory.

Table 5 summarizes the runtime of PPMOEA for one generation with pop-



(a) Test case 1

(b) Test case 2

(c) Test case 3

(d) Test case 4

(e) $\mathcal{C}(\mathbb{F}', \mathbb{F}'')$

Figure 2: Solution quality comparison of NSGA-II and PPMOEA. Figures (a) through (d) show metrics $\mathcal{M}_1^*(\mathbb{F})$, $\mathcal{M}_3^*(\mathbb{F})$ and $\mathcal{S}(\mathbb{F})$ for the respective test cases. Figure (e) shows the metric $\mathcal{C}(\mathbb{F}', \mathbb{F}'')$ for all four test cases.

ulation size $(\mu; \lambda) = (5; 35)$. It requires ≈ 52 minutes. In comparison NSGA-II computes *all* 500 generations in 10.62s; a factor of $\approx 150\,000$ faster. As anticipated privacy protection incurs a very high performance penalty which justifies our optimizations even if they may sacrifice solution quality.

Our decomposition technique enables simple parallelization of several protocol invocations. This is indispensable, since the aggregated CPU time used is ≈ 220 minutes (compared to ≈ 52 minutes runtime).

Recall that we precompute sums for the partial cost function $\psi_{t,p}(\mathbf{A})$ while not for $C_t(\mathbf{A})$. On average the ψ-circuit requires 2.38s for computation and 1.97s for communication. The C_t-circuit requires 4.48s and 3.74s, respectively. The two circuits differ (almost) only by the P additions in $C_t(\mathbf{A})$ with the ψ-circuit being even slightly larger. Precomputation reduces both measures by $\approx 47\%$.

Almost 90% of the runtime (≈ 45 minutes) is spent on fitness computation which has already been secured in the other proposals for PPEA. Our novel privacy-preserving complementary operations of an EA only contribute the remaining 10%. We attribute this superior performance to two factors: First, motivated by our real-world use case we chose a rather complex fitness function comprised of many and complex – e.g. maximum and product – arithmetic operations. Second, we carefully designed the other operations with runtime performance in mind and based on results from literature [13, 14, 15].

6 Related Work

The first PPEAs have been proposed in [26, 8]. The former presents a PPEA for combinatorial problems such as traveling salesman problem (TSP). The latter proposes a PPEA for rule discovery in distributed datasets.

Our PPMOEA improves [26] by supporting even the most general distributed setting, even for their use case of combinatorial problems. This is of practical interest, e.g. for a group of companies that wish to align their logistics in order to jointly ship goods. Han and Ng [8] consider arbitrarily partitioned datasets in their work.

Both proposals rely only on homomorphic encryption for security and therefore do not support general cost functions, but only permit scalar products. Our PPMOEA using Yao's protocol supports arbitrary functions.

Both proposals also reveal the result of the cost function (Han and Ng [8] reveal the absolute value while Sakuma and Kobayashi [26] reveal their relative ordering). This allows espionage by inferences from local input and output. Our PPMOEA only reveals the final result.

MOEAs have been studied extensively [2]. State of the art algorithms include NSGA-II [3], SPEA2 [31] and PAES [18]. All use a Pareto-dominance based selection scheme (similar to our PPMOEA), enhanced by some diversity operation (lacking in PPMOEA).

Integer-based EAs and their mutation operators have received less research. Schwefel [27] proposes to perturb the offspring by a binomial distributed value.

Rudolph [25] studies maximum entropy probability distributions for integer EAs. The use of stochastically rounded Gaussian distributed perturbations as used in PPMOEA is suggested by Hugosson et al. [10, 17].

Several implementations of SC exist. Malkhi et al. [22] where the first with a compiler for Yao's protocol. An optimized design for an alogrithm in genomic computing is presented in [11]. Optimizations built into the compiler – free "exclusive-or" gates [20] and security in the malicious model [24] – have also been proposed.

7 Conclusions

This paper presents a privacy-preserving multi-objective evolutionary algorithm capable of solving distributed multi-objective optimization problems between two parties. We elaborate on a real-world use case from collaborative supply chain management which involves sensitive information, such as mission-critical business secrets.

Our PPMOEA reveals only the optimal solution after the evolution, thus preventing inferences from intermediate results. We introduce several optimizations, including a general decomposition technique for Yao's garbled circuits. As a result the communication complexity is significantly reduced and computation can be parallelized.

Our experimental results show that performance remains the critical parameter. On the one hand, since cost function evaluation accounts for the majority of the runtime, less complex fitness functions that lead to similar solution quality should be evaluated in future work. On the other hand, since our novel privacy-preserving selection algorithm is comparatively efficient, but the diversity of our solutions is poor compared to NSGA-II (while convergence is similar), securely implementing a diversity operator in the selection algorithm remains future work.

References

[1] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the 1968 AFIPS Spring Joint Computer Conference*, 1968.

[2] C. Coello Coello, G. Lamont, and D. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Springer, 2007.

[3] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, 2000.

[4] A. Diponegoro and B. Sarker. Finite horizon planning for a production system with permitted shortage and fixed-interval deliveries. *Computers and Operations Research*, 33(8):2387–2404, 2006.

[5] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[6] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[7] O. Goldreich. *Foundations of Cryptography*, volume 2: Basic Applications. Cambridge University Press, 2004.

[8] S. Han and W. Ng. Privacy-preserving genetic algorithms for rule discovery. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery*, 2007.

[9] T. Hosoda and S. Disney. The governing dynamics of supply chains: The impact of altruistic behaviour. *Automatica*, 42(8):1301–1309, 2006.

[10] J. Hugosson, E. Hermberg, A. Brabazon, and M. O'Neil. An investigation of the mutation operator using different representations in grammatical evolution. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, 2007.

[11] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *Proceedings of the Symposium on Security and Privacy*, 2008.

[12] E. Karnin, J. Greene, and M. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29(1):35–41, 1983.

[13] F. Kerschbaum. Practical privacy-preserving benchmarking. In *Proceedings of the IFIP International Information Security Conference*, 2008.

[14] F. Kerschbaum, D. Dahlmeier, A. Schröpfer, and D. Biswas. On the practical importance of communication complexity for secure multi-party computation protocols. In *Proceedings of the ACM Symposium on Applied Computing*, 2009.

[15] F. Kerschbaum, N. Oertel, and L. Weiss Ferreira Chaves. Privacy-preserving computation of benchmarks on item-level data using RFID. In *Proceedings of the ACM Conference on Wireless Network Security*, 2010.

[16] V. Khare, X. Yao, and K. Deb. Performance scaling of multi-objective evolutionary algorithms. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, 2003.

[17] W. Klein Haneveld and M. van der Vlerk. Stochastic integer programming: General models and algorithms. *Annals of Operations Research*, 85:39–57, 1999.

[18] J. Knowles and D. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[19] D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1998.

[20] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, 2008.

[21] H. Lee and S. Whang. Information sharing in a supply chain. *International Journal of Manufacturing Technology and Management*, 1(1):79–93, 2000.

[22] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - a secure two-party computation system. In *Proceedings of the USENIX Security Symposium*, 2004.

[23] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT 99*, 1999.

[24] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology - ASIACRYPT 2009*, 2009.

[25] G. Rudolph. An evolutionary algorithm for integer programming. In *Proceedings of the Conference on Parallel Problem Solving from Nature*, 1994.

[26] J. Sakuma and S. Kobayashi. A genetic algorithm for privacy preserving combinatorial optimization. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, 2007.

[27] H. Schwefel. *Numerical optimization of computer models.* Wiley, 1981.

[28] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[29] A. C. Yao. Protocols for secure computations. In *Proceedings of the Symposium on Foundations of Computer Science*, 1982.

[30] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[31] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report TIK-Report 103, ETH Zurich, 2001.