Journal of Artificial Intelligence Research 57 (2016) 621-660 Submitted 9/16; published 12/16

P-SyncBB: A Privacy Preserving Branch and Bound DCOP Algorithm

Tal Grinshpoun TALGR@ARIEL.AC.IL
Ariel University
Ariel, Israel

Tamir Tassa TAMIRTA@OPENU.AC.IL
The Open University
Ra'anana, Israel

Abstract

Distributed constraint optimization problems enable the representation of many combinatorial problems that are distributed by nature. An important motivation for such problems is to preserve the privacy of the participating agents during the solving process. The present paper introduces a novel privacy-preserving branch and bound algorithm for this purpose. The proposed algorithm, P-SyncBB, preserves constraint, topology and decision privacy. The algorithm requires secure solutions to several multi-party computation problems. Consequently, appropriate novel secure protocols are devised and analyzed. An extensive experimental evaluation on different benchmarks, problem sizes, and constraint densities shows that P-SyncBB exhibits superior performance to other privacy-preserving complete DCOP algorithms.

1. Introduction

Distributed constraint optimization (DCOP) (Hirayama & Yokoo, 1997; Modi, Shen, Tambe, & Yokoo, 2005) is a general model for representing and solving distributed combinatorial problems, in which the variables of the problem are owned by different agents. The ability of the model to represent real-world problems such as meeting scheduling (Modi & Veloso, 2004), sensor nets (Zhang, Xing, Wang, & Wittenburg, 2005), and vehicle routing (Léauté & Faltings, 2011), resulted in the development of various complete DCOP algorithms. Most of those algorithms, such as SyncBB (Hirayama & Yokoo, 1997), ADOPT (Modi et al., 2005), NCBB (Chechetka & Sycara, 2006), AFB (Gershman, Meisels, & Zivan, 2009), and BnB-ADOPT (Yeoh, Felner, & Koenig, 2010), are based on *search*. Other paradigms for solving DCOPs include grouping of sub-problems (OptAPO, Mailler & Lesser, 2004) and dynamic programming (DPOP, Petcu & Faltings, 2005).

One of the main motivations for solving constraint problems in a distributed manner is that of *privacy*. The term privacy is quite broad, a fact that gave rise to several categorizations of the different types of privacy (Faltings, Léauté, & Petcu, 2008; Greenstadt, Grosz, & Smith, 2007; Grinshpoun, 2012). In this paper we relate to the categorization of Léauté and Faltings (2013) that distinguish between agent privacy, topology privacy, constraint privacy, and decision privacy.

Most research on DCOP privacy focused on constraint privacy, which means that constraint information should only be known to the agents whose variables are involved in the constraints. Some work has focused on measuring the extent of constraint privacy loss. Most

notably is the work of Maheswaran, Pearce, Bowring, Varakantham, and Tambe (2006) who proposed the VPS framework that was initially used to measure the constraint privacy loss in SyncBB and OptAPO. Later, VPS was also applied to the DPOP and ADOPT algorithms (Greenstadt, Pearce, & Tambe, 2006). Doshi, Matsui, Silaghi, Yokoo, and Zanker (2008) proposed to inject privacy-loss as a criterion to the problem solving process. Some previous work was also directed towards reducing constraint privacy loss. Most effort in the development of privacy-preserving search algorithms focused on DCSP, which is the *satisfaction* variant of DCOP (Nissim & Zivan, 2005; Silaghi & Mitra, 2004; Yokoo, Suzuki, & Hirayama, 2005). The work of Silaghi and Mitra (2004) addressed both satisfaction and optimization problems. However, the proposed solution is strictly limited to small scale problems since it depends on an exhaustive search over all possible assignments. Several privacy-preserving versions of the dynamic programming algorithm, DPOP, were proposed in the past (Greenstadt et al., 2007; Silaghi, Faltings, & Petcu, 2006). Recently, Léauté and Faltings (2013) proposed several versions of DPOP that provide strong privacy guarantees. While these versions are aimed for DCSPs, some of them may be also applicable to DCOPs. Considering a different aspect of constraint privacy, researchers have addressed problems in which the nature of a constraint is distributed among the constrained agents. Solutions to such problems include the PEAV formulation (Maheswaran, Tambe, Bowring, Pearce, & Varakantham, 2004) and *asymmetric* DCOPs (Grinshpoun, Grubshtein, Zivan, Netzer, & Meisels, 2013). Here, we restrict ourselves to the traditional symmetric DCOPs.

In the present paper we devise a novel DCOP algorithm that preserves constraint, topology and decision privacy. The new algorithm, called P-SyncBB, is based on SyncBB (Hirayama & Yokoo, 1997), which is the most basic DCOP search algorithm. We chose that algorithm since the search paradigm is applicable for a large variety of problems. Contrary to that, in DPOP (Petcu & Faltings, 2005) the size of messages grows exponentially in the induced width of the pseudo-tree structure that DPOP uses (the base of this exponentiation is the domain size); this fact hinders the applicability of DPOP for dense problems (Gershman, Zivan, Grinshpoun, Grubshtein, & Meisels, 2008) and problems with large domain sizes. OptAPO is also inappropriate for this purpose since it inherently involves comprehensive information sharing throughout the problem solving process, a fact that is counterproductive when considering privacy aspects. Moreover, OptAPO was shown to have rather poor runtime performance (Grinshpoun & Meisels, 2008).

In the course of developing the privacy-preserving algorithm we encountered several secure computation problems. In the main problem, a group of agents needs to compare the sum of private inputs held by those agents against an upper bound which is held by another agent, such that none of them learns information on neither the sum nor the private inputs of his peers. We introduce here a novel protocol for solving that problem. In addition, we devise a novel protocol for solving Yao's millionaires' problem (Yao, 1982) that does not depend on costly public key encryption and decryption.

We conduct extensive experimental evaluation on different benchmarks, problem sizes, and constraint densities, in order to compare the performance of P-SyncBB to the basic SyncBB (for assessing the price of privacy) and for comparing P-SyncBB to the privacy-preserving complete DCOP algorithm, P-DPOP$^{(+)}$, of Léauté and Faltings (2013). Our experiments show that while the price of privacy is significant (the runtime of P-SyncBB

is an order of magnitude higher than that of SyncBB), P-SyncBB exhibits superior performance to P-DPOP$^{(+)}$.

The plan of the paper is as follows. After some preliminaries (Section 2), the new privacy-preserving search algorithm is presented in Section 3. Several secure protocols are invoked during the run of the algorithm. These protocols are introduced in Section 4, along with analysis of their privacy properties. Section 5 includes communication and computational cost analysis of the security protocols, as well as experimental evaluation of the overall performance of P-SyncBB. Conclusions are drawn in Section 6.

This article has evolved from a paper that was published at AAMAS conference (Grinshpoun & Tassa, 2014). The proposed P-SyncBB algorithm of this extended version is an improved version of the algorithm that was presented in AAMAS. The present version of P-SyncBB offers decision privacy (while the earlier conference version did not). In addition, it solves the instances of the millionaires' problem (Section 4.2.3) by invoking newly developed and much more efficient sub-protocols. Furthermore, this extended version includes complete proofs, privacy analysis, detailed evaluation of the communication and computational costs, and a much more thorough experimental evaluation.

2. Preliminaries

Herein we provide the necessary preliminaries. Section 2.1 includes some background on the DCOP model; in Sections 2.2 and 2.3 we discuss the security and privacy notions that we adopt herein; and in Section 2.4 we provide the required background on probabilistic homomorphic encryption functions, which play an important role in our algorithms.

2.1 Distributed Constraint Optimization

A Distributed Constraint Optimization Problem (DCOP, Hirayama & Yokoo, 1997) is a tuple $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} >$ where $\mathcal{A}$ is a set of agents $A_1, A_2, \ldots, A_n$, $\mathcal{X}$ is a set of variables $X_1, X_2, \ldots, X_m$, $\mathcal{D}$ is a set of finite domains $D_1, D_2, \ldots, D_m$, and $\mathcal{R}$ is a set of relations (constraints). Each variable X_i takes values in the domain D_i, and it is held by a single agent. Each constraint $C \in \mathcal{R}$ defines a non-negative cost for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times \cdots \times D_{i_k} \rightarrow \Omega_{\mathcal{R}} := [0, q] \cup \{\infty\}$, for some $1 \leq i_1 < \cdots < i_k \leq m$. Constraints of infinite cost are called hard; they represent combinations of assignments that are strictly forbidden. Constraints of a finite cost are called soft. We assume hereinafter that q is a publicly known upper bound on the cost of the soft constraints.

An *assignment* is a pair including a variable, and a value from that variable's domain. We denote by a_i the value assigned to the variable X_i. A *partial assignment* (PA) is a set of assignments in which each variable appears at most once. A constraint $C \in \mathcal{R}$ is applicable to a PA if all variables that are constrained by C are included in the PA. The cost of a PA is the sum of all applicable constraints to the PA. A full assignment is a partial assignment that includes all of the variables. The goal in Constraint Optimization Problems in general, and in DCOPs in particular, is to find a full assignment of minimal cost.

For simplicity, we assume that each agent holds exactly one variable, i.e., $n = m$. We let n denote hereinafter the number of agents and the number of variables. We also concentrate on binary DCOPs, in which all constraints are binary, i.e., they refer to exactly

two variables. Such constraints take the form $C_{i,j} : D_i \times D_j \rightarrow \Omega_{\mathcal{R}}$. These assumptions are customary in DCOP literature (Modi et al., 2005; Petcu & Faltings, 2005).

Consider a full assignment which is acceptable, in the sense that it does not violate any hard constraint. Then its cost is at most $\binom{n}{2}q$. Hence, if we set the cost of each of the hard constraints to $q_\infty := \binom{n}{2}q + 1$, there is no need to distinguish between the two types of constraints. If the cost of the full assignment that the DCOP algorithm produces is at most $\binom{n}{2}q$, then that full assignment is an acceptable solution that does not violate any of the hard constraints. Otherwise, it is an unacceptable solution. If the DCOP algorithm is a complete algorithm then such a finding would imply that the problem has no acceptable solution.

2.2 Security Notions

A secure multi-party protocol is a protocol that enables several parties (or agents) to jointly compute a function over their inputs, while keeping these inputs private. Namely, given n agents, $A_1, \ldots, A_n$, where each agent A_i holds a private input x_i, $1 \leq i \leq n$, they wish to compute a function f over those private inputs, $y = f(x_1, \ldots, x_n)$, where the functionality f is publicly known, while keeping their private inputs secret. Such a protocol is considered *perfectly secure* if it does not reveal to any of the agents, say A_i, any information on the private inputs of his peers (x_j, $1 \leq j \neq i \leq n$) beyond what is implied by his own input x_i and the final output y of the protocol.[1] This notion of security coincides with the disclosure of *semi-private information*, a concept that was suggested in the context of distributed constraint satisfaction (Faltings et al., 2008).

One of the earliest secure multi-party protocols was presented by Yao (1982). He considered a problem that later became known as Yao's millionaires' problem. In that problem there are two players, each one holding a private integer representing his wealth. They wish to determine which one of them is richer (i.e., holds a larger integer) without revealing the actual private integers. There are several solutions for that problem, all of which rely on costly operations like oblivious transfer (Rabin, 1981; Even, Goldreich, & Lempel, 1985), or public-key encryption.

Multi-party computation was implemented in many contexts. For example, Lindell and Pinkas (2000) showed how to securely build an ID3 decision tree when the training set is distributed horizontally. The problem of distributed association rule mining was studied in the vertical setting, where each party holds a different set of attributes (Schuster, Wolff, & Gilburd, 2004; Vaidya & Clifton, 2002; Zhan, Matwin, & Chang, 2005), and in the horizontal setting (Kantarcioglu & Clifton., 2004; Tassa, 2014). Another context is of privacy-preserving multi-party protocols for computing k-anonymized views of distributed datasets (Jiang & Clifton, 2006; Tassa & Cohen, 2013; Tassa & Gudes, 2012; Zhong, Yang, & Wright, 2005). Techniques of multi-party computation were also implemented in order to perform privacy preserving collaborative filtering over distributed data (Jeckmans, Tang, & Hartel, 2012; Yakut & Polat, 2012). In recent years, multi-party computation was applied also in the context of DCOPs (Grinshpoun & Tassa, 2014; Léauté & Faltings, 2013; Tassa, Zivan, & Grinshpoun, 2015, 2016).

1. We adopt here the convention of relating to agents by the pronoun "he".

We assume herein that the parties are semi-honest; namely, they follow the prescribed protocol but try to glean more information than allowed from the protocol transcript. We also assume that they do not collude. Such assumptions are common in most studies that design secure multi-party protocols for practical problems, and in particular they were made in all above mentioned studies. The reader is referred to the work of Jeckmans et al. (2012), Jiang and Clifton (2006), Lindell and Pinkas (2000), Schuster et al. (2004), Zhong et al. (2005) for a discussion and justification of those assumptions in practical applications of multi-party computation.

We also note that when looking for practical solutions, some relaxations of the notion of perfect privacy are usually inevitable, provided that the excess information is deemed benign (Grinshpoun & Tassa, 2014; Kantarcioglu & Clifton., 2004; Léauté & Faltings, 2013; Tassa, 2014; Tassa & Cohen, 2013; Tassa et al., 2015; Tassa & Gudes, 2012; Vaidya & Clifton, 2002; Yakut & Polat, 2012; Zhong et al., 2005). For each of the security protocols that we present in this paper, we bound the excess information that it may leak to the interacting agents and explain why such leakage of information is benign, or how it may be reduced.

2.3 Privacy Notions

Léauté and Faltings (2013) have distinguished between four notions of private information in DCSPs. Those notions are also relevant in the context of DCOPs, whence we adopt them herein:

- *Agent privacy* – hiding from each agent the identity or even the existence of other agents with which he is not constrained.
- *Topology privacy* – hiding from each agent the topological structures in the constraint graph in which he is not involved. Namely, each agent should be aware of the nodes of variables with which his variable is constrained, and the edges that connect his node to those nodes, but nothing else on the graph structure.
- *Constraint privacy* – hiding from each agent the constraints in which he is not involved. Namely, agent A_k should not know anything about $C_{i,j}(\cdot,\cdot)$ if $k \notin \{i,j\}$.
- *Decision privacy* – hiding from each agent the final assignments to other variables.

Grinshpoun (2012) has considered two additional notions of private information in DCOPs – *domain privacy* and *algorithmic privacy*. As for domain privacy, the standard DCOP model usually assumes that the domains of all variables are fully known. (Agents can always implicitly *remove* certain values from their domain by the use of *unary* constraints; but then the problem of recovering the essential set of possible values for their variable becomes a problem of constraint privacy.) An exception to that is the *Open COP* model (Faltings & Macho-Gonzalez, 2005), which is not in the scope of the present research. As for algorithmic privacy, it is only relevant to algorithms that involve internal parameters which affect their performance, such as DSA (Zhang et al., 2005). Consequently, we do not relate to these two types of privacy.

We also note that while Léauté and Faltings (2013) only referred to decision privacy, i.e., final assignments, other researchers also considered the privacy of assignments during

the search process (Brito, Meisels, Meseguer, & Zivan, 2009). However, the work of Brito et al. (2009) was in the context of DCSPs, in which the first assignment during the search process that satisfies all the constraints is actually the final assignment, i.e., the decision. So the aspiration to preserve assignment privacy in that context was actually a means to preserve decision privacy, hence we do not explicitly relate to assignment privacy.

Out of the four privacy notions of Léauté and Faltings (2013), constraint privacy, has drawn the most attention in past research (Silaghi & Mitra, 2004; Maheswaran et al., 2006; Doshi et al., 2008). In fact, most of the privacy types in the categorization of Greenstadt et al. (2007) are actually variations of constraint privacy.

2.4 On Probabilistic Homomorphic Encryption Functions

A cipher is called public-key (or asymmetric) if its encryption function $\mathcal{E}(\cdot)$ of a plaintext depends on one key, K_e, which is publicly known, while the corresponding decryption function $\mathcal{E}^{-1}(\cdot)$ of a ciphertext depends on a private key K_d that is known only to the owner of the cipher, and K_d's derivation from K_e is computationally hard.

A cipher is called (additively) *homomorphic* if for every two plaintexts, m_1 and m_2, $\mathcal{E}(m_1 + m_2) = \mathcal{E}(m_1) \cdot \mathcal{E}(m_2)$. When the encryption function is randomized (in the sense that $\mathcal{E}(m)$ depends on m as well as on a random string), $\mathcal{E}$ is called *probabilistic*. Hence, a probabilistic encryption function is a one-to-many mapping (every plaintext m has many encryptions $m' = \mathcal{E}(m)$), while the corresponding decryption function is a many-to-one mapping (all possible encryptions m' of the same plaintext m are mapped by $\mathcal{E}^{-1}(\cdot)$ to the same m).

The semantically secure Paillier cipher (Paillier, 1999) is a public-key cipher that is both homomorphic and probabilistic.

3. Privacy Preserving DCOP Search

Synchronous Branch-and-Bound (SyncBB, Hirayama & Yokoo, 1997) was the first complete algorithm that was developed for solving DCOPs. SyncBB operates in a completely sequential manner, a fact that inherently renders its synchronous behavior. It is also the most basic *search* algorithm for solving DCOPs, and many more sophisticated DCOP search algorithms, such as NCBB (Chechetka & Sycara, 2006) and AFB (Gershman et al., 2009), are actually enhancements of SyncBB. Taking into account these comprehensions and considering the clear information flow within its search process, SyncBB is an ideal candidate to serve as the basis of a new privacy-preserving DCOP search algorithm.

The objective in this context is to allow the interacting agents to solve a given DCOP while maintaining constraint, topology and decision privacy. We first overview the original, non-privacy-preserving SyncBB algorithm (Hirayama & Yokoo, 1997), and then proceed to present the Privacy-preserving Synchronous Branch-and-Bound algorithm (P-SyncBB).

3.1 Synchronous Branch and Bound

The SyncBB algorithm assumes a static public ordering of the agents, $A_1, \ldots, A_n$. The search space of the problem is traversed by each agent assigning a value to its variable and passing the *current partial assignment* (CPA) to the next agent in the order, along with

the current cost of the CPA. After an agent completes assigning all values in the domain to his variable, he *backtracks*, i.e., he sends the CPA back to the preceding agent. To prevent exhaustive traversal of the entire search space, the agents maintain an *upper bound*, which is the cost of the best solution that was found thus far. The algorithm performs continuous comparisons between the costs of partial assignments and the current upper bound, in order to *prune* the search space. Algorithm 1 presents the pseudo-code of SyncBB with *value ordering*. We focus on a version of SyncBB that includes value ordering because such ordering is also needed for the privacy-preserving algorithm that we present here, P-SyncBB.

The run of SyncBB starts by the agents initializing the upper bound B to infinity (**init**, line 1). Then, agent A_1 sets the cost of the (currently empty) CPA, $Cost$, to zero and runs the **assign_CPA** procedure (lines 2-4). In that procedure, A_1 simply chooses a value from his variable's domain (**assign_CPA**, line 14), assigns it to his variable in the CPA (line 18), and sends a **CPA_MSG** with the updated CPA to agent A_2 (line 25; the third parameter in that message is a cost value that always equals zero when the sender is A_1).

When agent A_k, $k \geq 2$, receives the **CPA_MSG**, he first updates his data with the received partial assignment and cost (lines 5-6). Next, he performs value ordering according to the added cost to the CPA that is incurred by each value in his domain (lines 7-8). The cost that A_k's assignment adds is

$$x_k = \sum_{i=1}^{k-1} C_{i,k}(a_i, a_k)\,, \tag{1}$$

where a_i, a_k are the assignments of the variables that are governed by agents A_i and A_k, respectively. Hence, for each $v \in D_k$, agent A_k computes $x_k(v) = \sum_{i=1}^{k-1} C_{i,k}(a_i, v)$ and then he orders those values so that the sequence of added costs $x_k(v)$ is non-decreasing. Note that agent A_1 never receives a **CPA_MSG**, so he does not perform value ordering. Finally, the agent runs the **assign_CPA** procedure (line 9).

The **assign_CPA** procedure begins with the agent choosing the next value from the pre-ordered domain (line 14). In case no more values remain, the agent backtracks (lines 15-16). Otherwise, A_k assigns to his variable X_k the next value v from his domain D_k (line 18). At this stage, the agent reaches a pruning decision point (line 19), in which he computes the cost of the augmented CPA ($Cost+x_k$) and checks whether this cost is greater than or equal to the upper bound. If so, then the pruning of this branch of the search space is achieved by backtracking (line 20). Otherwise, if the agent is the last one (A_n), a new best solution is found. In that case, the agent A_n broadcasts a **NEW_SOLUTION_MSG** (line 22), and backtracks in order to continue the traversal of the search space (line 23). All agents receive the **NEW_SOLUTION_MSG** and update the new best solution, $B\_Solution$, and the new bound, B, accordingly (lines 12-13). Back to the **assign_CPA** procedure, any agent which is not the last one sends a **CPA_MSG** to the subsequent agent (line 25).

Finally, we discuss the **backtrack** procedure. In the case of A_1, a backtrack means that the entire search space was traversed, and so the algorithm terminates (lines 26-27). Any other agent just removes his variable from the CPA (line 29) and sends a **BACK-TRACK_MSG** to the preceding agent (line 30). An agent that receives a **BACK-TRACK_MSG** removes his value from the CPA (line 10) and continues the search by assigning the next value in order (line 11).

Algorithm 1 – SyncBB (executed by agent A_k)

```
procedure init
 1: B ← ∞
 2: if k = 1 do
 3:    Cost ← 0
 4:    assign_CPA()

when received (CPA_MSG,PA,PA_Cost) do
 5: CPA ← PA
 6: Cost ← PA_Cost
 7: for each v ∈ D_k do compute x_k(v)
 8: order the values v ∈ D_k so that the sequence x_k(v) is non-decreasing
 9: assign_CPA()

when received (BACKTRACK_MSG) do
10: remove X_k from CPA
11: assign_CPA()

when received (NEW_SOLUTION_MSG,Solution,Sol_Cost) do
12: B_Solution ← Solution
13: B ← Sol_Cost

procedure assign_CPA
14: choose next value v in order from D_k
15: if no value exists do
16:    backtrack()
17: else
18:    X_k ← v
19:    if Cost + x_k ≥ B do
20:       backtrack()
21:    else if k = n do
22:       broadcast(NEW_SOLUTION_MSG,CPA,Cost + x_k)
23:       backtrack()
24:    else
25:       send(CPA_MSG,CPA,Cost + x_k) to A_{k+1}

procedure backtrack
26: if k = 1 do
27:    broadcast(TERMINATE)
28: else
29:    remove X_k from CPA
30:    send(BACKTRACK_MSG) to A_{k−1}
```

3.2 Privacy Preserving Synchronous Branch and Bound

When considering a version of SyncBB that preserves privacy, one must pay special attention to the upper bound. In SyncBB, the upper bound is the most fundamental piece of information during the problem solving process, and it is publicly known to all agents. The effectiveness of the algorithm lies in the continuous comparisons of the costs of partial

assignments with the current upper bound, in order to prune the search space. Curious agents may collect information on costs of partial assignments in attempt to infer information on private constraints of other agents. This is a major source of trouble from the perspective of constraint privacy. Following that, the most fundamental task in the design of the privacy-preserving version, P-SyncBB, is to separate between information regarding the costs of partial or full assignments and the upper bound, while still enabling pruning of the search space. P-SyncBB achieves such a separation as described below.

P-SyncBB, like SyncBB, assumes a static public ordering of the agents, $A_1, \ldots, A_n$. Namely, all agents know each other and the position of each agent in the selected order. P-SyncBB allows only one agent, A_1, to know the upper bound. The selection of A_1 for that purpose is made because, according to the original SyncBB algorithm, A_1 never sees the assignments of the other agents. Knowing the upper bound while remaining oblivious of the assignments that determined that upper bound does not enable extracting information on unknown constraint costs. As for the other agents, A_k, $2 \leq k \leq n$, each of them receives throughout the search process CPAs from his preceding agent; consequently, each of them knows at each stage the assignments of the agents that precede him in the order. To prevent those agents from extracting information on unknown constraint costs, P-SyncBB prevents them from gaining knowledge on the costs of CPAs and on the current upper bound.

In the original SyncBB algorithm, the cost of the CPA is passed between the agents along with the CPA. Contrary to that, in P-SyncBB the cost of the CPA must not be known to any agent, since this cost may leak private constraint information. Consequently, at each pruning decision point, the agent A_k that holds a CPA which corresponds to the current assignments to $X_1, \ldots, X_k$, must compare between the cost of that CPA and the upper bound in a secure manner, i.e., without learning neither the cost of the CPA nor the upper bound. Such a non-trivial computational goal is achieved by restraining the information flow in the algorithm and by introducing secure protocols for summation and comparison.

A direct consequence of preserving constraint privacy is that P-SyncBB preserves also topology privacy. Indeed, as the communication patterns between agents during P-SyncBB are determined solely by the static public ordering of the agents, and not by the topology (as is the case with other privacy-preserving algorithms, e.g. (Tassa et al., 2015), which, consequently, need to apply specifically designed tools for hiding topological information), and as it is impossible to distinguish between zero cost binary constraints (as is the case with non-adjacent pairs of agents in the constraint graph) and positive cost binary constraints, topology privacy is respected.

Finally, P-SyncBB also preserves decision privacy. The main cryptographic tool that enables achieving that goal is the Paillier cryptosystem (see Section 2.4). Each agent A_k, $2 \leq k \leq n$, generates a key-pair in a Paillier cryptosystem, and informs A_1 of the corresponding public key. Let $\mathcal{E}_k(\cdot)$ denote the encryption function in A_k's cipher. A_k will use that cipher during P-SyncBB in order to store with A_1 the current assignment to his variable X_k. After A_1 identifies the stage in the search where the optimal assignment was found (while remaining oblivious of the actual assignments to all other variables in that optimal assignment), A_1 sends back to each of the other agents the assignment of that agent's variable in the optimal solution, encrypted with that agent's cipher. That way, each of the

agents gets, at the completion of P-SyncBB, his assignment in the optimal solution, but he remains oblivious of the assignments of his peer agents in that optimal solution.

The pseudo-code of P-SyncBB is given in Algorithm 2 (which is broken into two boxes due to its length) and described in detail next. The secure protocols that Algorithm 2 invokes are described in Section 4.

Algorithm 2 – P-SyncBB (executed by agent A_k) – first part

```
procedure init
 1: if k = 1 do
 2:    B ← ∞
 3:    assign_CPA()

when received (CPA_MSG,PA) do
 4: CPA ← PA
 5: for each v ∈ D_k do compute x_k(v)
 6: order the values v ∈ D_k so that the sequence x_k(v) is non-decreasing
 7: ComputedCPA ← false
 8: assign_CPA()

when received (BACKTRACK_MSG) do
 9: remove X_k from CPA
10: assign_CPA()

when received (CHECK_SOLUTION_MSG) do
11: Cost ← invoke Protocol 3 to securely compute the solution's cost (i.e., Σ_{i=2}^{n} x_i)
12: if Cost < B do
13:    B ← Cost
14:    CurrentlyBest ← true
15: else
16:    CurrentlyBest ← false
17: broadcast(REQUEST_CURRENT_ASSIGNMENT_MSG)

when received (REQUEST_CURRENT_ASSIGNMENT_MSG) do
18: z_k ← E_k(X_k)
19: send(CURRENT_ASSIGNMENT_MSG,k,z_k) to A_1
20: if k = n do
21:    backtrack()

when received (CURRENT_ASSIGNMENT_MSG,j,z_j) do
22: if CurrentlyBest = true do
23:    CA(j) ← z_j
```

The run of P-SyncBB starts by agent A_1 initializing the upper bound B to infinity and running the **assign_CPA** procedure (**init**, lines 2-3). In that procedure, A_1 simply chooses a value from his variable's domain (**assign_CPA**, line 24), assigns it to his variable in the CPA (line 31), and sends a **CPA_MSG** with the updated CPA to agent A_2 (line 32).

The handling of a **CPA_MSG** is exactly as in SyncBB, except that the cost of the CPA is not passed with that message. Therefore, in line 7 A_k sets the flag *ComputedCPA* to **false**; this flag indicates whether an agent has already computed the cost of the CPA

Algorithm 2 – P-SyncBB (executed by agent A_k) – second part

procedure assign_CPA
24: choose next value v in order from D_k
25: **if** no value exists **do**
26: backtrack()
27: **else if** $k = n$ **do**
28: $X_k \leftarrow v$
29: send(**CHECK_SOLUTION_MSG**) to A_1
30: **else if** $k < 4$ **do**
31: $X_k \leftarrow v$
32: send(**CPA_MSG**,CPA) to A_{k+1}
33: **else**
34: **if** $ComputedCPA =$ **false do**
35: invoke Protocol 4 to securely compute $CPA.cost$ (i.e., $\sum_{i=2}^{k-1} x_i$)
36: $ComputedCPA \leftarrow$ **true**
37: $ShouldBacktrack \leftarrow$ invoke Protocol 5 to securely check whether $CPA.cost + x_k \geq B$
38: **if** $ShouldBacktrack =$ **true do**
39: backtrack()
40: **else**
41: $X_k \leftarrow v$
42: send(**CPA_MSG**,CPA) to A_{k+1}

procedure backtrack
43: **if** $k = 1$ **do**
44: **for all** $1 \leq j \leq n$ **do**
45: $CA(j) = CA(j) \cdot \mathcal{E}_j(0)$
46: send(**ASSIGNMENT_IN_SOLUTION_MSG**,$CA(j)$) to A_j
47: **else**
48: remove X_k from CPA
49: send(**BACKTRACK_MSG**) to A_{k-1}

when received (**ASSIGNMENT_IN_SOLUTION_MSG**,y) **do**
50: $X_k \leftarrow \mathcal{E}_k^{-1}(y)$
51: Terminate

that was received from his preceding agent. After that, the agent runs the **assign_CPA** procedure (line 8).

The **assign_CPA** procedure begins with the agent choosing the next value from the pre-ordered domain (line 24). In case no more values remain, the agent backtracks (lines 25-26). Any agent which is not the last one (A_n) skips to line 30. We will relate to agent A_n (lines 27-29) later. If $k < 4$ then we do not perform pruning; hence, in that case A_k just assigns to his variable X_k the next value v from his domain D_k and then sends a **CPA_MSG** to the succeeding agent (lines 30-32). Otherwise ($k \geq 4$), A_k proceeds as follows: if the *ComputedCPA* flag is **false**, A_k invokes a secure computation of the CPA's cost, i.e., $\sum_{i=2}^{k-1} x_i$, where x_i are as in Eq. (1); namely, $x_i = \sum_{j=1}^{i-1} C_{j,i}(a_j, a_i)$ is the sum of costs of binary constraints that relate to agent A_i and his preceding agents in the order. After completing this, the agent sets the flag to **true** (lines 34-36). The secure computation in line 35 is performed by invoking Protocol 4 (Section 4.1). The outcome

of this computation is that A_k holds one additive share of the CPA's cost, while A_1 holds another share. Each of those shares is completely random and thus reveals no information on the CPA's cost. Hence, the actual cost is never disclosed to any single agent. A detailed description of how this is achieved is given in Section 4.1. (We start pruning only when $k \geq 4$ since Protocol 4 is relevant only for such values of k.)

Note that the usage of the *ComputedCPA* flag ensures that the cost of any given CPA (i.e., the sum of constraints incurred by assignments to $X_1, \ldots, X_{k-1}$) is computed only once (Protocol 4). Whenever A_k *changes* his assignment to X_k, only Protocol 5 is invoked, in order to compare the cost of the augmented CPA (involving $X_1, \ldots, X_{k-1}, X_k$) to the current upper bound. Moreover, as A_k knows in advance x_k for each value in D_k, then if two consecutive values from D_k add the same cost x_k, the pruning check may be avoided. (This is discarded from the pseudo-code for clarity reasons.)

At this stage, the agent reaches a pruning decision point. In the original SyncBB, this means checking whether the cost of the CPA combined with A_k's added cost x_k is greater than or equal to the upper bound. In P-SyncBB, this check is performed without A_k knowing neither the cost of the CPA nor the upper bound. This is achieved by invoking Protocol 5 (Section 4.2) in line 37. In case the output of Protocol 5 is **true**, the agent backtracks (lines 38-39). Otherwise, he assigns his variable in the CPA and sends a **CPA_MSG** to the subsequent agent (lines 41-42).

The **backtrack** procedure is exactly as in SyncBB for agents A_k, $k > 1$ (lines 47-49). However, for A_1 it requires a special termination procedure (lines 43-46), since a backtrack for A_1 means that the exhaustive search has finished and it is now necessary to inform each of the agents of their assignment in the optimal solution that was found. We defer the description of this special treatment to a later stage, as it depends on ingredients of the algorithm that are yet to be discussed.

Now, we return to the case of the last agent A_n (lines 27-29). In this case, the CPA is a full assignment, so a **CHECK_SOLUTION_MSG** is sent to agent A_1. When agent A_1 receives the **CHECK_SOLUTION_MSG**, he invokes Protocol 3 (Section 4.1) with $k = n$ in order to obtain the cost of the candidate solution (line 11). The cost of the CPA equals the sum of the private values x_k, $k \geq 2$, as defined in Eq. (1), where x_k is known only to A_k. (Note that the fact that A_1 computes the sum of private inputs which are held by other agents demonstrates the information separation in the protocol.) In case the cost of the new solution, $Cost$, is lower than the existing upper bound, B, the new upper bound is updated (line 13). In addition, A_1 sets a flag $CurrentlyBest$ to indicate whether the current full assignment is the best one that was found so far or not (lines 14 and 16). Finally, A_1 sends to all agents the message **REQUEST_CURRENT_ASSIGNMENT_MSG**.

When receiving the **REQUEST_CURRENT_ASSIGNMENT_MSG** message, each agent encrypts his current assignment with his own Paillier public key (line 18), and sends it with the message **CURRENT_ASSIGNMENT_MSG** back to A_1 (line 19). In the case of the last agent A_n, he also backtracks in order to proceed with the search (lines 20-21).

When A_1 receives the message **CURRENT_ASSIGNMENT_MSG** from an agent A_j, he stores the received value in the jth entry of a special array CA, but only if the current full assignment is the best one that was found so far, and thus has the potential of becoming, eventually, the sought-after optimal solution (lines 22-23). Otherwise (if the flag $CurrentlyBest$ is **false**), A_1 ignores those messages. Note that A_1 cannot decrypt

$CA(j)$, as it is encrypted under $\mathcal{E}_j$, an encryption that only A_j can decrypt. He stores those messages only to be used at the end of the search, as we describe below. We also note that since $\mathcal{E}_j$ are probabilistic encryption functions, A_1 cannot even identify equalities between the assignment values in different **CURRENT_ASSIGNMENT_MSG** messages that are received from the same agent A_j during the algorithm.

We now return to the description of lines 43-46 in the **backtrack** procedure. A_1 keeps the array CA so that at the completion of the search he can forward to A_j, $1 \leq j \leq n$, the value $CA(j)$ which holds an encryption under $\mathcal{E}_j$ of A_j's assignment in the optimal solution that was found. Consequently, A_j can decrypt it and find the sought-after assignment for his variable. If A_1 had sent to A_j the value $CA(j)$ as it was originally received from A_j, then A_j would have been able to recognize the stage in the search in which he generated that encrypted value, and thus infer the assignments of all preceding agents in the optimal solution. Indeed, such an identification is possible since $\mathcal{E}_j$ is a probabilistic encryption function and, hence, every encrypted value will appear, with almost certainty, only once in the course of the algorithm. In order to prevent A_j from making such inferences, and relying on the fact that $\mathcal{E}_j$ is public-key and homomorphic, A_1 multiplies $CA(j)$ with a fresh random encryption of 0 (line 45). Such an operation alters the ciphertext, but does not alter the underlying plaintext since, owing to homomorphism, $\mathcal{E}_j^{-1}(\mathcal{E}_j(x) \cdot \mathcal{E}_j(0)) = x$. Then, A_1 sends to each agent A_j the message **ASSIGNMENT_IN_SOLUTION_MSG** with the modified value $CA(j)$, $1 \leq j \leq n$ (line 46). A_j, upon receiving that message, decrypts it, assigns the resulting plaintext to his variable X_j, and then terminates (lines 50-51).

It is important to note that A_1 requests the current assignments from all agents whenever a full assignment is tested, regardless of the result of the inequality testing in line 12, in order to prevent from A_j, $j > 1$, learning about PAs that could be part of the optimal solution. Indeed, if A_1 had requested the current assignments only when $CurrentlyBest$ is **true** (since he ignores the messages with current assignments otherwise, see line 22), then every agent A_j could have created a list of all PAs $(a_1, \ldots, a_{j-1}, a_j)$ that were the CPAs when A_1 sent to him a **REQUEST_CURRENT_ASSIGNMENT_MSG** message. Then, at the end of P-SyncBB, if A_j learns that the assignment to his variable X_j in the optimal solution is say a_j^o, he would have been able to infer that the assignment vector to $X_1, \ldots, X_{j-1}$ in that optimal solution is one of the vectors $(a_1, \ldots, a_{j-1})$ that appears in his list together with a_j^o as the Jth component. By enforcing a less efficient approach and have A_1 send **REQUEST_CURRENT_ASSIGNMENT_MSG** messages even when he knows upfront that they are useless, we limit the ability of agents A_j, $j > 1$, to make such undesirable inferences.

Theorem 1 *P-SyncBB is sound and complete.*

Proof. The soundness of P-SyncBB depends on the correctness of selecting a new best solution. The relevant full assignment is verified by agent A_1 to have a cost lower than that of the existing best solution. To do that, A_1 invokes a secure computation of the new upper bound (**CHECK_SOLUTION_MSG**, line 11). Since the algorithm is completely sequential, all agents A_k have the correct assignments in their local versions of the CPA, so they participate in Protocol 3 with correct x_k values. Finally, the correctness of the $Cost$ computation is ensured by the correctness of Protocol 3 (see Section 4.1).

The completeness of P-SyncBB follows from the exhaustive search structure. Only partial assignments whose cost reach the upper bound are not extended and therefore it is guaranteed that the algorithm finds an optimal solution. Termination also follows from the exhaustive structure of the Branch-and-Bound algorithm in which no partial assignment can be explored twice. □

4. Secure Protocols and Privacy Analysis

In this section we present the secure protocols that are invoked by P-SyncBB, Algorithm 2. As discussed in Section 2.1, $C_{i,j}(a_i, a_j) \leq q_\infty$ for all $1 \leq i < j \leq n$ and $a_i \in D_i$, $a_j \in D_j$, where q_∞ is a publicly known upper bound on the costs of all constraints (hard and soft). Hence, $Q := \binom{n}{2} q_\infty$ is a publicly known upper bound on the cost of any partial or full assignment. Since $q_\infty = 1 + \binom{n}{2} q$, where q is the upper bound on the cost of all soft constraints, we have

$$Q = q\binom{n}{2}^2 + \binom{n}{2}. \tag{2}$$

In Section 4.1 we discuss the secure summation protocols that are invoked in (**CHECK_SOLUTION_MSG**, line 11) and (**assign_CPA**, line 35) of P-SyncBB, and we analyze their privacy. Our main cryptographic contribution is given in Section 4.2: we begin by describing our secure comparison protocol that is invoked in (**assign_CPA**, line 37) (Section 4.2.1); in Section 4.2.2 we analyze its privacy; and then in Section 4.2.3 we discuss efficient implementations of two steps in that protocol. Finally, we analyze the overall privacy preservation in P-SyncBB (Section 4.3), and shortly discuss runtime attacks (Section 4.4) and colluding agents (Section 4.5).

4.1 Secure Summation Protocols

The computational problem in (**CHECK_SOLUTION_MSG**, line 11) of P-SyncBB is the classical problem of secure summation of private integers. It can be solved by a slight modification of Benaloh's (1986) protocol, which is described in Protocol 3. Here, S denotes an arbitrary and publicly known integer which is much larger than Q (the upper bound on the inputs and their sum).

Protocol 3 – Secure summation of private inputs

Input: $Q, S \in \mathbf{Z}^+$ such that $S \gg Q$;
Agent A_i, $2 \leq i \leq k$, has an integer $x_i \in [0, Q]$, such that $x := \sum_{i=2}^{k} x_i \leq Q$.
Output: A_1 gets x.

1: Each A_i, $2 \leq i \leq k$, selects $x_{i,j} \in \mathbf{Z}_S := [0, S-1]$, $2 \leq j \leq k$, randomly and uniformly such that $\sum_{j=2}^{k} x_{i,j} = x_i \bmod S$.
2: A_i sends $x_{i,j}$ to A_j, for all $2 \leq i \neq j \leq k$.
3: A_j computes $s_j := \sum_{i=2}^{k} x_{i,j} \bmod S$, for all $2 \leq j \leq k$.
4: Agents $A_2, \ldots, A_k$ send $s_2, \ldots, s_k$ to A_1.
5: A_1 computes $x \leftarrow s_2 + \cdots + s_k \bmod S$.

Theorem 2 *Protocol 3 is perfectly secure: none of the agents $A_2, \ldots, A_k$ learns any information about the inputs of his peers or the overall sum x, while A_1 learns no information on the inputs of the other agents beyond what is implied by the overall sum x.*

Proof. The perfect security of the protocol is a direct consequence of the fact that each agent A_i, $2 \leq i \leq k$, breaks up his private input x_i to modular additive shares $x_{i,j} \in \mathbf{Z}_S$, $2 \leq j \leq k$, which are chosen independently and uniformly at random. Specifically, each of the agents A_i chooses $k-2$ independent and uniformly random values $x_{i,j} \in \mathbf{Z}_S$, $2 \leq j \leq k-1$, and then sets $x_{i,k} = x_i - \sum_{j=2}^{k-1} x_{i,j}$, where all arithmetic operations are modulo S. It follows that the last share, $x_{i,k}$, just like the other shares $x_{i,j}$, $2 \leq j \leq k-1$, is also uniformly distributed on $\mathbf{Z}_S$. Consequently, also each $s_j = \sum_{i=2}^{k} x_{i,j} \bmod S$, $2 \leq j \leq k$, is uniformly distributed on $\mathbf{Z}_S$, since the addends in the sum that defines s_j are independent of each other. Hence, none of the shares s_j in the sum x reveals any information about either the private inputs x_i or the sum x. Therefore, none of the agents $A_2, \ldots, A_k$ learns any information about the inputs of his peers, or about the overall sum x. As for A_1, he learns the value of x, but no information on the private inputs $x_2, \ldots, x_k$, since any possible tuple of values $(x_2, \ldots, x_k)$ is equally probable given the shares $s_2, \ldots, s_k$ due to the fact that the latter distribute uniformly on $\mathbf{Z}_S$ regardless of the private inputs. □

Next, we discuss the secure implementation of (**assign_CPA**, line 35) in P-SyncBB. Here, it is needed to compute the cost of the CPA that is incurred by the assignments $a_1, \ldots, a_{k-1}$ of all preceding agents. That cost equals $x := \sum_{i=2}^{k-1} x_i$, where $x_i = \sum_{j=1}^{i-1} C_{j,i}(a_j, a_i)$ is a value that is known to A_i. While the summation in (**CHECK_SOLUTION_MSG**, line 11) computes the cost of a *full* assignment, and the sum is revealed only to agent A_1 who does not know the actual assignments, the summation in (**assign_CPA**, line 35), which occurs more frequently than the previous summation, needs to be executed for *partial* assignments. Revealing the resulting sum to any of the agents, even to agent A_1 who is not aware of the actual assignments, might be hazardous since it may be used to infer information on the private inputs of other agents. Hence, instead of letting a single agent reveal the sum $x = \sum_{i=2}^{k-1} x_i$, Protocol 4 ends with agents A_1 and A_k sharing that sum.

Protocol 4 – Computing additive shares in the sum of private inputs

Input: $Q, S \in \mathbf{Z}^+$ such that $S \gg Q$;
Agent A_i, $2 \leq i \leq k-1$ (where $k \geq 4$), has an integer $x_i \in [0, Q]$, such that $x := \sum_{i=2}^{k-1} x_i \leq Q$.
Output: A_1 gets a random $s_2 \in \mathbf{Z}_S$ and A_k gets $s_k \in \mathbf{Z}_S$ so that $s_2 + s_k = x \bmod S$.

1: Agents $A_2, \ldots, A_{k-1}$ perform Steps 1-3 of Protocol 3 for $x_2, \ldots, x_{k-1}$.
2: Agents $A_3, \ldots, A_{k-1}$ send $s_3, \ldots, s_{k-1}$ to A_k.
3: A_k computes $s_k \leftarrow s_3 + \cdots + s_{k-1} \bmod S$.
4: A_2 sends s_2 to A_1.

Protocol 4 starts by implementing the first three steps of Protocol 3 (Step 1). Then, agents $A_3, \ldots, A_{k-1}$ send their shares to A_k who adds them up to get s_k (Steps 2-3), while agent A_2 sends his share to A_1 (Step 4). Consequently, the two agents A_1 and A_k hold two

values s_2 and s_k that are random modular shares in the sum x. Namely, each of those values distributes uniformly at random over $\mathbf{Z}_S$ (as a result of the uniform selection of shares $x_{i,j}$ in Step 1 of Protocol 3) and $s_2 + s_k = x \bmod S$.

A note on the assumption $k \geq 4$: Protocol 4 is executed by a prefix-subsequence of agents, $A_1, A_2, \ldots, A_{k-1}, A_k$. The $k-2$ agents $A_2, \ldots, A_{k-1}$ hold private inputs; the protocol computes shares in the sum of those private inputs, which are then given to A_1 and A_k. This problem definition is relevant only for $k \geq 3$. However, the case $k = 3$ requires a different treatment since then there is no need to invoke Protocol 3 for a summation of just one element (x_2). Hence, for the sake of simplicity, Protocol 4 is invoked only when $k \geq 4$.

Corollary 3 *Protocol 4 is perfectly secure in the sense that none of the agents $A_1, \ldots, A_k$ learns any information about the inputs of his peers or the overall sum x.*

Corollary 3 follows directly from Theorem 2 and its proof since Protocol 4, which is a small variant of Protocol 3, reveals to no single agent all of the shares in the sum (as done in Protocol 3 with A_1).

4.2 A Secure Comparison Protocol

The main computational problem occurs in (**assign_CPA**, line 37) of P-SyncBB. There, agent A_k needs to check whether $CPA.cost + x_k \geq B$ where: (i) $CPA.cost$ is the cost of the CPA which is incurred by the assignments $a_1, \ldots, a_{k-1}$ of all preceding agents, (ii) x_k is the cost that agent A_k's assignment adds (see Eq. (1)), and (iii) B is the current upper bound. We recall that, as a result of executing Protocol 4 in (**assign_CPA**, line 35), $CPA.cost$ is shared by A_1 and A_k that hold two modular additive shares in it, denoted s_2 and s_k respectively. The value of x_k is known only to A_k while B is known only to A_1.

4.2.1 The Basic Protocol

Protocol 5 solves the above described computational problem. In Step 1, A_k adds x_k to his share s_k in $CPA.cost$. As a consequence, s_2 and s_k (which are held by A_1 and A_k respectively) are two modular shares in the augmented sum $x := \sum_{i=2}^{k} x_i$ that equals the cost of the CPA due to the assignments $a_1, \ldots, a_k$ to $X_1, \ldots, X_k$. If we view those shares as integers from $[0, S-1]$, then either $s_2 + s_k = x$ (Case 1) or $s_2 + s_k = S + x$ (Case 2). Next, A_k sends to A_1 the value $s_k + r$, where r is selected uniformly at random from $[0, S-Q-1]$ (Steps 2-3). A_1 then computes the difference $y = s_2 + s_k + r - B$ (Step 4). (The purpose of adding the random mask r is to prevent A_1 from inferring the difference $\delta := x - B$.)

Our goal now is to check whether δ is negative or not. In Case 1 $y = \delta + r$ while in Case 2 $y = \delta + r + S$. Since $x, B \in [0, Q]$ then $\delta \in [-Q, Q]$. Hence, in Case 1 $y - r \leq Q$ while in Case 2 we have $y - r \geq S - Q$ (where $S - Q \gg Q$). Therefore, in order to check in which of the two cases we are, A_k and A_1 perform a secure protocol to check whether $y \geq S - Q + r$ (Step 5): since y is known only to A_1 while $S - Q + r$ is known only to A_k, this is an instance of the well-known Yao's millionaires' problem (see Section 2.2), which can be solved by one of many available protocols for its solution (Yao, 1982; Fischlin, 2001; Ioannidis & Grama, 2003; Blake & Kolesnikov, 2004). If that inequality does not hold then we are in Case 1 and $y = \delta + r$. If, however, it does hold, then we are in Case 2 and $y = \delta + r + S$. In the latter case, A_k sets $r \leftarrow r + S$. Hence, at the completion of Step 5,

we have $y = \delta + r$. It is important to note that only A_k needs to learn the answer to the inequality verification; A_1 learns no information about whether $y \geq S - Q + r$ or not.

Next, the two agents check whether $y \geq r$ or not. This is again an instance of the millionaires' problem, since y is known only to A_1 and r is known only to A_k. Since $x - B = \delta = y - r$ then $y \geq r$ if and only if $x \geq B$. Hence, A_k may learn from the inequality verification whether $x \geq B$ or not (Step 6).

Protocol 5 – Comparing a shared sum against an unknown bound

Input: $Q, S \in \mathbf{Z}^+$ such that $S \gg Q$;
Agent A_1 has $s_2 \in \mathbf{Z}_S$ and $B \in [0, Q]$;
Agent A_k has $s_k \in \mathbf{Z}_S$ and $x_k \in [0, Q]$.
Output: Letting $x := (s_2 + s_k + x_k) \bmod S$, A_k learns whether $x \geq B$.

1: A_k sets $s_k \leftarrow s_k + x_k \bmod S$.
2: A_k generates uniformly at random an integer $r \in [0, S - Q - 1]$.
3: A_k sends $s_k + r$ to A_1.
4: A_1 computes $y = s_2 + s_k + r - B$.
5: A_k and A_1 check securely whether $y \geq S - Q + r$. If so, A_k updates $r \leftarrow r + S$.
6: A_k and A_1 check securely whether $y \geq r$. A_k infers that $x \geq B$ if and only if $y \geq r$.

4.2.2 Privacy Analysis of Protocol 5

Protocol 5 is "almost" perfectly secure, as we prove in Theorem 5. But first, we prove the following general purpose lemma.

Lemma 4 *Let:*

- *X be a random variable that takes values in the domain $[W_X] := \{0, 1, 2, \ldots, W_X\}$.*
- *Y be a random variable that distributes uniformly on $[W_Y] := \{0, 1, 2, \ldots, W_Y\}$, where $W_Y \geq W_X$.*
- *$Z = X + Y$.*

Then in probability $1 - \frac{W_X}{W_Y+1}$, the value of Z reveals no information on the value of X. Furthermore, in probability $\frac{W_X}{W_Y+1}$, the value of Z reveals either an upper or a lower bound on X, but nothing beyond that.

Proof. The random variable Z takes values in the interval $[0, W_X + W_Y]$. That interval can be partitioned into three disjoint intervals: $I_1 = [0, W_X - 1]$, $I_2 = [W_X, W_Y]$ and $I_3 = [W_Y + 1, W_X + W_Y]$. Denoting the value that Z attains by z, there are three cases to consider:

- $z \in I_1$: This can happen if $X = x$ and $Y = z - x$ for any of the values $0 \leq x \leq z$. The probability of such an event is therefore

$$\Pr(Z = z) = \sum_{x=0}^{z} \Pr(X = x) \cdot \Pr(Y = z - x) = \frac{1}{W_Y + 1} \cdot \sum_{x=0}^{z} \Pr(X = x)\,, \qquad (3)$$

where the latter equality follows from the uniform distribution of Y.

- $z \in I_3$: This can happen if $X = x$ and $Y = z - x$ for any of the values $z - W_Y \leq x \leq W_X$. The probability of such an event is

$$\Pr(Z = z) = \sum_{x=z-W_Y}^{W_X} \Pr(X = x) \cdot \Pr(Y = z - x) = \frac{1}{W_Y + 1} \cdot \sum_{x=z-W_Y}^{W_X} \Pr(X = x) \,. \quad (4)$$

- $z \in I_2$: This can happen if $X = x$ and $Y = z - x$ for any of the values $0 \leq x \leq W_X$. The probability of such an event is

$$\Pr(Z = z) = \sum_{x=0}^{W_X} \Pr(X = x) \cdot \Pr(Y = z - x) = \frac{1}{W_Y + 1} \cdot \sum_{x=0}^{W_X} \Pr(X = x) = \frac{1}{W_Y + 1} \,. \quad (5)$$

We will now prove that if $z \in I_2$ then z reveals no information on x. By Bayes Theorem,

$$\Pr(X = x | Z = z) = \frac{\Pr(Z = z | X = x) \cdot \Pr(X = x)}{\Pr(Z = z)} \,, \quad 0 \leq x \leq W_X \,. \quad (6)$$

Since $\Pr(Z = z | X = x) = \Pr(Y = z - x)$ and Y distributes uniformly on $[0, W_Y]$, we infer that

$$\Pr(Z = z | X = x) = \frac{1}{W_Y + 1} \quad (7)$$

in this case. (Note that in this case $z - x$ always falls in the range $[0, W_Y]$.) Hence, by Eqs. (6), (5) and (7),

$$\Pr(X = x | Z = z) = \frac{\frac{1}{W_Y+1} \cdot \Pr(X = x)}{\frac{1}{W_Y+1}} = \Pr(X = x) \,,$$

whence the a-posteriori information that z reveals on x is just like the a-priori knowledge we had on x. In view of Eq. (5), the probability of $z \in I_2$ is

$$\sum_{z=W_X}^{W_Y} \Pr(z) = \frac{W_Y - W_X + 1}{W_Y + 1} = 1 - \frac{W_X}{W_Y + 1} \,.$$

That concludes the proof of the first claim in the lemma.

If $z \in I_1$ then the value of z reveals a non-trivial upper bound on x (since then $x \leq z$ where $z \leq W_X - 1$), while if $z \in I_3$ it reveals a non-trivial lower bound on x (since then $x \geq z - W_Y$ where $z \geq W_Y + 1$). The probability of the union of those two cases is $\frac{W_X}{W_Y+1}$. In either of those cases, nothing is revealed beyond the upper or lower bound. Indeed, if $z \in I_1$ we have, by Eqs. (6) and (3),

$$\Pr(X = x | Z = z) = \frac{\Pr(Z = z | X = x) \cdot \Pr(X = x)}{\Pr(Z = z)} =$$
$$= \frac{\frac{1}{W_Y+1} \cdot \Pr(X = x)}{\frac{1}{W_Y+1} \cdot \sum_{\xi=0}^{z} \Pr(X = \xi)} = \frac{\Pr(X = x)}{\sum_{\xi=0}^{z} \Pr(X = \xi)} \,.$$

Therefore, the a-posteriori distribution $\Pr(X|Z=z)$ coincides with the a-priori one, after removing the values of X beyond the upper bound which z implies and re-scaling. Similarly if $z \in I_3$. That concludes the proof of the second claim in the lemma. □

Theorem 5 *At the end of Protocol 5 agent A_1 may learn a non-trivial lower or upper bound on x, but nothing beyond that, in probability $\frac{Q}{S-Q}$, or nothing at all, in probability $1-\frac{Q}{S-Q}$. As for agent A_k, he may learn either a lower bound on x, in probability x/S, or an upper bound, in probability $(Q-x)/S$, or nothing at all, in probability $1-\frac{Q}{S}$.*

Proof. A_1 learns the value $y+B=s_2+s_k+r$ (Step 4). If $y+B<S$ he infers that it is Case 1 and therefore $y+B=x+r$; otherwise he infers that it is Case 2, whence $y+B-S=x+r$. In any case, he learns the value $x+r:=z$. Since x takes values in the range $[Q]=\{0,1,\ldots,Q\}$ while r is chosen uniformly at random from $[S-Q-1]$, the first claim in the theorem follows directly from Lemma 4. Next, we turn to discuss A_k.

A_k learns in Step 5 of Protocol 5 whether Case 1 holds ($s_2+s_k=x$) or Case 2 does ($s_2+s_k=S+x$). In Case 1, A_k infers that $x \geq s_k$. That lower bound is non-trivial only when $s_k>0$, namely, when $s_k \in \{1,\ldots,x\}$. Since s_k distributes uniformly at random over $\mathbf{Z}_S$ (as we explain below), A_k may learn a non-trivial lower bound on x in probability $|\{1,\ldots,x\}|/|\mathbf{Z}_S|=x/S$. (Prior to Step 1 in Protocol 5 s_k distributes uniformly at random over $\mathbf{Z}_S$ since at that point it is the sum of random $\mathbf{Z}_S$-shares. Hence, even after adding to s_k the value of x_k it remains uniformly distributed over $\mathbf{Z}_S$.)

In Case 2, on the other hand, both shares s_2 and s_k must be strictly greater than x. Indeed, if one of the shares, say s_2, would have been less than or equal to x, then also the other share, $s_k=x-s_2$, would have been less than or equal to x, and that is Case 1. Hence, in Case 2 $x<s_k$. Only when $s_k \leq Q$, that upper bound is non-trivial. Therefore, A_k learns a non-trivial upper bound on x if and only if $x<s_k \leq Q$, namely, in probability $(Q-x)/S$.

To summarize, A_k learns some information on x if and only if $1 \leq s_k \leq Q$. He learns nothing on x when $s_k=0$ or when $s_k>Q$. Since s_k distributes uniformly in $\mathbf{Z}_S$, A_k learns no information at all in probability $1-\frac{Q}{S}$. □

Few comments are in order here:

1. Theorem 5 implies that the only potential leakages of information are to only two agents – A_1 and A_k.
2. Those potential leakages of information are only with respect to $x=\sum_{i=2}^{k} x_i$ (but not with respect to x_i), and only in the form of a lower or an upper bound.
3. Even though A_1 plays a pivotal role in P-SyncBB, as he keeps the upper bound B, he does not know any of the assignments to the variables that are controlled by the other agents. Namely, even though A_1 may learn (in negligible probability) lower bounds on the CPA's cost x, he does not know what are the assignments to $X_2,\ldots,X_k$ that determine that x.
4. Most importantly, the probability of those potential leakages ever to occur can be made negligible by using a sufficiently large S, as stated in Lemma 6 below. Increasing S does not pose a practical barrier since it has a modest toll in terms of both communication and computational costs, as we show in Section 5.1.

Lemma 6 *Let $\varepsilon > 0$ be an arbitrarily small number and let w be an upper bound on the number of invocations of Protocol 5 in the course of P-SyncBB. Then if $S > (nw + \varepsilon)Q/\varepsilon$, it is guaranteed that none of the agents will learn in the course of P-SyncBB even a single lower or upper bound on the cost of any partial assignment, in probability at least $1 - \varepsilon$.*

Proof. By Theorem 5, in any single invocation of Protocol 5, the probability of each agent not learning anything is at least $1 - \frac{Q}{S-Q}$. Hence, for any single agent, if he participates in w invocations of Protocol 5, the probability that he learns nothing at all in all of those invocations of the protocol is greater than $1 - \frac{wQ}{S-Q}$. Hence, the probability that none of the n agents learns anything throughout the execution of P-SyncBB is greater than $1 - \frac{nwQ}{S-Q}$. It is now easy to verify that if $S > (nw + \varepsilon)Q/\varepsilon$, then the latter probability is greater than $1 - \varepsilon$. □

For example, assume a setting with $n = 100$ agents, where each of the soft binary constraints has a cost that is bounded by $q = 100$. Then, by Eq. (2), $Q \approx 2.5 \cdot 10^9$. Assume further a generous bound $w = 10^8$ on the number of invocations of Protocol 5. Then in order to get a certainty of zero information leakage from this protocol in probability at least $1 - \varepsilon$ with $\varepsilon = 10^{-50}$, we should take $S \approx 2.5 \cdot 10^{69}$. As our analysis in Section 5.1 shows that the communication and computational costs depend only on $\ell_S := \lceil \log S \rceil$, such a value of S poses no practical burden (apart from the need to use multiple-precision integers in the implementation).

4.2.3 Solving the Millionaires' Problem

In Steps 5 and 6 of Protocol 5, agents A_1 and A_k need to resolve two instances of the millionaires' problem. In each of those steps they need to compare two private values without disclosing to each other the difference between the two compared values. They may resolve these millionaires' problems by applying Yao's garbled circuit protocol (Yao, 1982) or any one of the many protocols that appeared later (Fischlin, 2001; Ioannidis & Grama, 2003; Blake & Kolesnikov, 2004). However, each of those protocols invokes costly sub-protocols for oblivious transfer (Even et al., 1985), or public-key encryption and decryption. We therefore suggest here simpler solutions that rely on a third party, T. That third party can be A_2 (since Protocols 4 and 5 are relevant only for $k \geq 4$). The third party is non-trusted in the sense that he too must not learn the difference between the two compared values.

The solution that we describe in each step is different.

The millionaires' problem in Step 5. Here, A_1 holds the private value y while A_k holds the private value r. The goal is to distinguish between two possible cases. Viewing y and r as integers (and not as residues modulo S), then, as discussed earlier, in Case 1 $y \leq Q + r$, while in Case 2 $y \geq S - Q + r$. If A_1 and A_k had sent to T the two values y and r, then T would have been able to distinguish between the two cases (that's good) but then he would be able to subsequently infer the value of δ, as $\delta = y - r$ in Case 1 and $\delta = y - r - S$ in Case 2 (that's bad). In order to allow T to distinguish between the two cases, but disable him from inferring the value of δ, we suggest to implement the following sub-protocol, [MP-Step5], which we describe informally.

Sub-protocol [MP-Step5]. A_k will choose uniformly at random an integer $\gamma \in [Q+1, S-Q]$. Then, A_1 will send y to T while A_k will send $z := \gamma + r$ to T. T will check if $y \geq z$. If $y \geq z$ then $y \geq \gamma + r \geq Q + 1 + r$, whence Case 1 cannot hold and then this is Case 2. If, on the other hand, $y < z$, then $y < \gamma + r \leq S - Q + r$, whence Case 2 cannot hold and then this is Case 1. Therefore, the information that A_1 and A_k send to T (namely, y and z) can enable T to distinguish between the two cases.

It remains to analyze what the third party T may learn on δ from the information it gathers in sub-protocol [MP-Step5].

Theorem 7 *At the end of sub-protocol [MP-Step5], the third party T may learn a non-trivial lower or upper bound on δ, but nothing beyond that, in probability $\frac{2Q}{S-2Q}$, or nothing at all, in probability $1 - \frac{2Q}{S-2Q}$.*

Proof. The difference $y - z$ equals $\delta - \gamma$ in Case 1 and $\delta - \gamma + S$ in Case 2. Therefore, in either case, T may learn the value of $\delta - \gamma$, where δ is an unknown integer in $[-Q, Q]$ and γ is uniformly distributed on the large interval $[Q+1, S-Q]$. Introducing $\delta' := Q - \delta$ and $\gamma' := \gamma - Q - 1$, we see that δ' takes values in $[2Q] = \{0, 1, 2, \ldots, 2Q\}$ and γ' takes values in $[0, S-2Q-1]$. Furthermore, $\delta' + \gamma' = -(\delta - \gamma) - 1$. Hence, T learns the value of $\delta' + \gamma'$. The claims of the theorem now follow from Lemma 4 (with $W_X = 2Q$ and $W_Y = S - 2Q - 1$). □

The millionaires' problem in Step 6. In Step 6, $y = \delta + r$ where δ is an integer in the range $[-Q, Q]$. A_1 and A_k need to check whether $y \geq r$ in order to decide whether $\delta \geq 0$ or not. Here, we cannot apply the same simple strategy as we did in Step 5. What enabled the above described solution for Step 5 was the fact that the two cases that needed to be identified were separated by a large gap (the difference $y - r$ was either smaller than Q or larger than $S - Q$, a value which is much larger than Q). Here, there is no such gap, since if $\delta = -1$ the inequality verification should end with a negative result while if $\delta = 0$ it should end with a positive result. Therefore, a totally different strategy should be taken here.

We suggest to use a method that was presented by Tassa and Bonchi (2014). One of the basic secure multi-party problems that was considered there was the following. Assume that two parties, P_1 and P_2, are holding two private integers, a_1 and a_2. They wish to allow a third party, T, to learn the quotient a_1/a_2 without learning the values of a_1 and a_2. Protocol 3 there suggests that P_1 and P_2 choose a random real number ρ and then they send to T the values ρa_1 and ρa_2. T can then divide those two numbers in order to recover the correct fraction a_1/a_2. However, he cannot learn the values of a_1 and a_2, as they were masked by the random multiplier ρ that T does not know. The main effort in the analysis (Tassa & Bonchi, 2014, Section 4.2) is dedicated to the manner in which the random multiplier ρ should be selected in order to minimize the information that ρa_i leaks on a_i, $i = 1, 2$. Specifically, it is shown there that if one generates a random real number $M \sim Z$, where Z is the distribution on $[1, \infty)$ with probability density function $f_Z(\mu) = \mu^{-2}$, and then one proceeds to select r randomly and uniformly from the interval $(0, M)$, then the masked product ra_i which T receives reveals to him very little information on a_i. More specifically, even if T has an a-priori belief probability on the value of a_i, the received value of ra_i might enable T to draw a different a-posteriori belief probability on

the value of a_i, but that a-posteriori belief probability does not enable T to draw better estimates on the original value of a_i. This is shown by a detailed analysis (Tassa & Bonchi, 2014, Section 4.2) and then by experimental evaluation (Tassa & Bonchi, 2014, Section 7.2).

Our problem here is different, but it can be solved by applying the same idea. A_1 and A_k will choose a random multiplier ρ according to the distribution that was described above. In addition, A_k will choose uniformly at random a real number $\theta \in (0, 1)$. Then, A_1 will send to T the product ρy while A_k will send to T the value $\rho \cdot (r - \theta)$. Then, T will compare the two obtained values. It is easy to see that $\rho y \geq \rho \cdot (r - \theta)$ if and only if $\delta + \theta \geq 0$, namely, if and only if $\delta \geq 0$. Furthermore, even though T may compute the difference between the two values, which equals $\rho \cdot (\delta + \theta)$, then as shown by Tassa and Bonchi (2014), the usage of the random multiplier ρ prevents T from learning the value of δ.

We introduce here the random shift θ for the following reason. In the application that was discussed by Tassa and Bonchi (2014) the values $a_1 = 0$ or $a_2 = 0$ were not private (as the final quotient a_1/a_2 reveals those values); therefore, it was sufficient there to multiply a_i by the random ρ, even though T may infer that if $\rho a_i = 0$ then also $a_i = 0$. In our application, on the other hand, we consider the value $\delta = 0$ also private. Hence, in order to prevent it from being disclosed, we use the random shift θ.

4.3 Overall Privacy Analysis of Algorithm P-SyncBB

In the preceding sections we analyzed the privacy of each of the secure protocols that P-SyncBB invokes; these are the only places in P-SyncBB in which the agents exchange information that relates to their private inputs. Here we take a global look at P-SyncBB:

(1) In (**CHECK_SOLUTION_MSG**, line 11) P-SyncBB invokes Protocol 3. As shown in Theorem 2, Protocol 3 is perfectly secure except that it reveals to A_1 the solution's cost. However, as A_1 is not aware of the assignments to any of the variables apart from his, such leakage of information cannot be used to infer anything on private constraints of other agents.

(2) In (**assign_CPA**, line 35) P-SyncBB invokes Protocol 4. As shown in Corollary 3, that protocol reveals no private information to any of the interacting agents.

(3) In (**assign_CPA**, line 37) P-SyncBB invokes Protocol 5. As shown in Theorem 5, after executing that protocol agent A_1 may learn a non-trivial lower or upper bound on the cost of the CPA, but nothing beyond that, in probability $\frac{Q}{S-Q}$. Protocol 5 may also reveal to A_k non-trivial bounds on the cost of the CPA (Theorem 5). However, as shown in Lemma 6, by choosing a sufficiently large S, it is possible to ensure that the probability of any information leakage from Protocol 5 to any of the parties is made arbitrarily small.

So to summarize, A_1 is the single agent that gets the cost of full assignments, but as he remains oblivious of the assignments that determine those costs, he is unable to extract information on unknown constraint costs. Agents A_k, $2 \leq k \leq n$, on the other hand, cannot extract information on the costs of CPAs, or on the current upper bound. Therefore, thanks to this carefully designed separation of information, P-SyncBB maintains constraint privacy.

We note that the achieved constraint privacy is not perfect because of inherent information leakage due to pruning. Specifically, since each of the agents A_k, $5 \leq k \leq n$, receives from its predecessor only CPAs that have a cost that is no larger than the current upper bound, A_k may infer that the cost of any CPA that he did not receive is higher than the

cost of the last CPA that he did receive. The sensitivity of such information leakage depends on the underlying application. In order to resolve this issue, it would be essential to refrain from passing the CPA to the next agent and, instead, utilize secure multi-party protocols whenever the cost of the augmented CPA needs to be computed. We leave such modifications of P-SyncBB to future work.

A direct consequence of preserving constraint privacy is that P-SyncBB preserves also topology privacy. Indeed, as the communication patterns between agents during P-SyncBB are determined solely by the static public ordering of the agents, and not by the topology, and as it is impossible to distinguish between zero cost binary constraints (as is the case with non-adjacent pairs of agents in the constraint graph) and positive cost binary constraints, topology privacy is respected.

Finally, P-SyncBB also preserves decision privacy. The main cryptographic tool that enables achieving that goal is the Paillier cryptosystem (see Section 2.4). The mechanism by which each agent A_k stores with A_1 current assignments of his variable, encrypted by $\mathcal{E}_k$, and then, at the end, A_1 gives back to A_k a "scrambled" version of the encrypted assignment of X_k in the found optimal solution, ensures that each agent gets, at the completion of P-SyncBB, his assignment in the optimal solution, but he remains oblivious of the assignments of his peer agents in that optimal solution.

Faltings et al. (2008) presented P-DPOP, a privacy-preserving version of DPOP (Petcu & Faltings, 2005). P-DPOP was designed for solving DCSPs (not DCOPs) while fully preserving agent privacy and partially preserving topology, constraint, and decision privacy. P-DPOP had a leak in topology privacy that was later fixed by Léauté and Faltings (2013). Apart from fixing that privacy leak, Léauté and Faltings also presented a sequence of three privacy-preserving versions of DPOP: P-DPOP$^{(+)}$, P$^{3/2}$-DPOP$^{(+)}$, and P^2-DPOP$^{(+)}$. All three versions preserve agent privacy and partial topology privacy. The least private and most efficient version, P-DPOP$^{(+)}$, preserves constraint and decision privacy only partially, as it may leak related information. P$^{3/2}$-DPOP$^{(+)}$ preserves decision privacy fully but it still respects constraint privacy only partially. The last version, P^2-DPOP$^{(+)}$ (most private, least efficient), preserves constraint and decision privacy fully.

It is hard to compare the constraint privacy preservation of P-DPOP$^{(+)}$, P$^{3/2}$-DPOP$^{(+)}$, and P^2-DPOP$^{(+)}$ to that of ours, since the former algorithms are designed for DCSPs. While they can be extended for solving DCOPs, it is beyond the scope of this study to analyze the constraint privacy leakages of such extensions.

As for topology privacy, all variants of Léauté and Faltings (2013) respect it only partially. The minor leaks of topology information lie in the fact that a variable might be able to discover a lower bound on a neighboring variable's degree in the constraint graph. In view of our discussion above, such leaks of information do not happen in P-SyncBB.

As for agent privacy, the variants of Léauté and Faltings (2013) manage to maintain it as their algorithm involves sending messages only between agents that are linked by a constraint, and thanks to using codenames. The nature of SyncBB (and thus the inherited nature of P-SyncBB) requires agents to send messages to agents that are not necessarily linked to them by a constraint. Hence, P-SyncBB seems to be unsuitable for cases in which agent privacy is of essence.

As emerges from our extensive experimental evaluation of Section 5.2, P-SyncBB appears to be a most attractive alternative to (the DCOP-extensions of) any of the variants of Léauté

and Faltings (2013), due to its superior performance in terms of runtime. Indeed, in almost all cases P-SyncBB is more efficient even than the least privacy-preserving (and hence the most efficient) variant, P-DPOP$^{(+)}$.

4.4 A Note on Runtime Attacks

Silaghi and Mitra (2004) pointed out that search-based algorithms, such as SyncBB, might leak information to an outside observer through their overall runtime. Specifically, any adversary, be it an outside observer or any one of the participating agents, may draw conclusions from the overall runtime of the distributed protocol on the density of constraints, since problems with dense constraints typically allow less pruning than problems with sparse constraints, whence their solution typically takes more time. While this might be considered as a downside of search-based algorithms (such as SyncBB and P-SyncBB), it must not be a show-stopper for this class of algorithms, as we proceed to argue.

First, the running time could be easily hidden from an outside observer by applying network security measures for preventing an outside network tap. As for any one of the participating agents, they could use the overall runtime to draw such conclusions, but apart from a very gross estimate of the constraint density, they could not learn anything about specific constraints nor about which pairs of agents are constrained. Moreover, such a general estimate of the constraint density is usually known upfront to the participating agents and can hardly be considered as a breach of privacy.

Furthermore, if there are settings where such leakage of information may be considered sensitive, we could enhance P-SyncBB by adding random noise to the runtime. Specifically, each agent A_k will generate a secret parameter $\theta_k \in [0,1]$ and then, after computing the flag *ShouldBacktrack* (**assign_CPA**, line 37), A_k will generate a bit b that equals **true** in probability θ_k and **false** in probability $1-\theta_k$. Then, the condition for backtracking (line 38) will be replaced with (*ShouldBacktrack* **and** b). By selecting $\theta_k = 1$ one recovers the original P-SyncBB. By taking lower values of θ_k, a fraction of $1-\theta_k$ of A_k's backtracking decisions will be ignored. Hence, such a mechanism further obfuscates the runtime to a level of practically no use, at the price of increased runtime.

It is important to stress that in privacy there are no silver bullet solutions. In order to fully resolve this potential problem of runtime leakage one could simply take $\theta_k = 0$ for all A_k, so that pruning is never done (in Section 5.2 we refer to that algorithm as P-Ex). Alas, such an algorithm performs exhaustive search, whence it is impractical. Alternatively, one may use one of the private versions of DPOP; here too, the downside is the exponential communication and computational costs.

Therefore, the bottom line conclusion is that any privacy-preserving solution has its advantages and drawbacks. None should be considered (or expected) to be a silver bullet solution. Any privacy-preserving solution, be it in the context of DCOP solving or in any other context, must identify its advantages and shortcomings, and then position itself on the privacy shelf as yet another choice available for the practitioner. The practitioner should then make his calculated choice based on the characteristics of his application, the available resources, the perceived threats, and the consequences of a successful attack.

4.5 A Note on Colluding Agents

As stated in Section 2.2, our algorithm P-SyncBB assumes that the players do not collude. Such an assumption is common in many studies that design secure multi-party protocols for practical problems. The smallest coalitions that pose privacy problems in P-SyncBB are coalitions between the pivotal agent A_1 and another agent A_k, for some $k \geq 4$. Such a coalition may recover the cost of CPAs of the form $(a_1, \ldots, a_k)$, what may jeopardize the privacy of the intermediate agents, $A_2, \ldots, A_{k-1}$. To solve this vulnerability, it is possible to skip (**assign_CPA**, line 35) which invokes Protocol 4, since that is the place where A_1 and A_k get to hold shares in the cost of the CPA $(a_1, \ldots, a_k)$. By doing so, we keep the cost of the CPA spread among all players $A_1, \ldots, A_k$. But then we would need to pay the price in (**assign_CPA**, line 37), since now the comparison of the CPA's cost against B would need to involve all agents $A_1, \ldots, A_k$. Specifically, we would need to design a new protocol in which:

- A_1's private input is B.
- A_i's private input is $x_i = \sum_{j=1}^{i-1} C_{j,i}(a_j, a_i)$, $2 \leq i \leq k$.
- The desired output to A_k is a bit which indicates whether $\sum_{i=2}^{k} x_i \geq B$ or not.

The new protocol should be designed in a manner that provides immunity to coalitions of size at most t, where t must be at least 2 in order to improve P-SyncBB's immunity to coalitions.

We leave this problem for future research. However, since that problem is a generalization of Yao's millionaires' problem (Yao, 1982), and already the solution of the latter problem depends on costly oblivious transfer sub-protocols (Even et al., 1985), it is clear that the advantage of increased privacy comes with a price of increased communication and computational costs.

5. Evaluation

Here, we evaluate the performance of P-SyncBB. In Section 5.1 we analyze the communication and computational costs of our secure protocols and then in Section 5.2 we report our experimental results with P-SyncBB.

5.1 Communication and Computational Costs

In this section we analyze the communication and computational costs of the secure protocols that we presented in Section 4. We begin with the secure inequality verifications in Steps 5 and 6 of Protocol 5, as described in Section 4.2.3.

In Step 5, in the first round A_1 and A_k send to T the values y and z that can be encoded in $\ell_S = \lceil \log S \rceil$ and $\ell_S + 1$ bits, respectively. In the second communication round T informs A_k of the inequality verification result (1 bit). Hence, in total, the inequality verification in Step 5 involves two communication rounds, and three messages with an overall size of $2\ell_S + 2$ bits.

In Step 6, in the first round A_1 and A_k send to T two real values, which are generated by multiplying ℓ_S-bit integers with a random real masking multiplier ρ. Let ℓ_f denote the size in bits for representing such large real numbers ($\ell_f \approx \ell_S$). In the second communication round T informs A_k of the inequality verification result (1 bit). Hence, in total, the inequality

verification in Step 6 involves two communication rounds, and three messages with an overall size of $2\ell_f + 1$ bits.

The runtime of these two steps is analyzed as follows. Step 5 involves one random generation (γ) followed by one addition ($z = \gamma + r$) and then one comparison of integers ($y \geq z$?). Step 6 involves three random number generations (M, ρ, θ), one addition and floating point multiplication ($\rho \cdot (r - \theta)$), and one comparison of floating point numbers ($\rho y \geq \rho \cdot (r - \theta)$?).

We proceed to illustrate the great advantage which is offered by our secure inequality verifications, in comparison to invocations of protocols for solving Yao's millionaires' problem. Consider, for example, one of the simplest protocols for that problem that was presented by Blake and Kolesnikov (2004). That protocol, just like our sub-protocols for secure inequality verifications in Steps 5 and 6 of Protocol 5, has a minimal number of communication rounds – two. Its communication and computational costs depend on the length of the two integers to be compared. In both Steps 5 and 6 we need to compare integers of length ℓ_S bits. Hence, if we choose to implement the protocol of Blake and Kolesnikov in either of those steps, its communication cost will be $O({\ell_S}^2)$ (compared to $O(\ell_S)$ and $O(\ell_f)$ of our sub-protocols). As for the runtime, it is dominated by the need to execute ℓ_S 1-out-of-2 oblivious transfer sub-protocols. The most common way of implementing 1-out-of-2 oblivious transfer involves four encryptions and two decryptions in a commutative encryption scheme, like RSA (Rivest, Shamir, & Adleman, 1978). Namely, we are looking at a total of $6\ell_S$ modular exponentiations. By comparing the runtime needed to perform those computations to the negligible runtime of our proposed sub-protocols, the improvement is staggering.

Next, we analyze the costs of our Protocols 3, 4 and 5. Tables 1, 2 and 3 summarize the communication costs of those protocols. Each row in those tables corresponds to one round (or batch of rounds) in the related protocol, and the last row shows the total number of communication rounds, messages, and transmitted bits for that protocol.

Table 1 relates to Protocol 3, which is invoked by P-SyncBB for $k = n$ (in order to compute the overall cost of some given solution). In the first round (Step 2) each of $A_2, \ldots, A_n$ sends a message of ℓ_S bits to all others. In the second round (Step 4), those $n - 1$ agents send their shares to A_1.

Table 1: Communication costs of Protocol 3

Rounds	# messages	# bits
Step 2	$(n-1)(n-2)$	$(n-1)(n-2)\ell_S$
Step 4	$n-1$	$(n-1)\ell_S$
2	$(n-1)^2$	$(n-1)^2\ell_S$

In Table 2, the first round (Step 1 in Protocol 4) corresponds to the exchange of additive shares between the $k - 2$ agents $A_2, \ldots, A_{k-1}$ (as described in Step 2 of Protocol 3). The second and third rounds correspond to Steps 2 and 4 in Protocol 4.

In Table 3, the first round corresponds to Step 3 in Protocol 5; here, the integer could be as large as $2S - Q - 2$, whence it requires $\ell_S + 1$ bits. The next two rows in Table 3 correspond to the inequality verifications in Steps 5 and 6 of Protocol 5. As implied by our

Table 2: Communication costs of Protocol 4

Rounds	# messages	# bits
Step 1	$(k-2)(k-3)$	$(k-2)(k-3)\ell_S$
Step 2	$k-3$	$(k-3)\ell_S$
Step 4	1	ℓ_S
3	k^2-4k+4	$(k^2-4k+4)\ell_S$

earlier discussion, each such verification entails two communication rounds in which three messages are sent with a total size of $2\ell_S + 2$ bits (Step 5) or $2\ell_f + 1$ bits (Step 6).

Table 3: Communication costs of Protocol 5

Rounds	# messages	# bits
Step 3	1	$\ell_S + 1$
Step 5 (2)	3	$2\ell_S + 2$
Step 6 (2)	3	$2\ell_f + 1$
5	7	$3\ell_S + 2\ell_f + 4$

Next, we turn to discuss the runtimes of those protocols. It can be verified that the runtime of Protocol 3 is dominated by $3k-6$ non-concurrent modular additions. Similarly, the runtime of Protocol 4 is dominated by $3k-10$ non-concurrent modular additions. As for Protocol 5, it entails a small constant number of random number generations, additions, multiplications and comparisons.

5.2 Experiments

The performance of P-SyncBB is evaluated in three sets of experiments. The objective of the first set is to compare the runtime performance of P-SyncBB to competing privacy-preserving complete DCOP algorithms (Section 5.2.2). The purpose of the second set is to demonstrate the overhead of privacy preservation by comparing the runtime and network load of P-SyncBB to that of the regular (non-privacy-preserving) SyncBB (Section 5.2.3). These two sets of experiments include two classes of benchmarks – unstructured random DCOPs and graph coloring problems. The third set of experiments demonstrates the scalability of the various algorithms on four classes of benchmarks – unstructured random DCOPs, graph coloring problems, scale-free networks, and real-world meeting scheduling problems (Section 5.2.4).

The first benchmark consists of unstructured randomly generated DCOPs on which we perform two types of experiments. In the first type, relevant to the first two sets of experiments, we fix the number of agents n (a different value of n is chosen for each set of experiments), the domain sizes (we selected $|D_1| = \cdots = |D_n| = n$), and the maximal constraint cost ($q = 100$), and vary the constraint (edge) density $0.3 \leq p_1 \leq 0.9$. Note that using lower density values $p_1 < 0.3$ results in unconnected constraint graphs. In the second type, relevant to the third set of experiments, we demonstrate the scalability of the algorithms by fixing the constraint density p_1 and varying the number of agents n.

The second benchmark consists of distributed 3-color graph coloring problems. In traditional graph coloring problems, where it is a-priory known that all constraints are binary inequality constraints with unit costs, constraint privacy is irrelevant (though topology privacy is relevant). Thus, we consider a variation of distributed 3-color graph coloring problems in which each pair of equal values of constrained agents imposes a random and *private* cost up to a maximum of $q = 100$. In such problems, both constraint and topology privacy are of interest. The structure in these problems lies in the constraint matrix between every pair of nodes; such matrices are diagonal, since a cost is paid only in the case where the two assigned colors are the same (and then it depends on the color and on the pair of nodes). We perform two types of experiments on this benchmark. In the first type, relevant to the first two sets of experiments, we fix the number of agents n and vary the constraint density $0.3 \leq p_1 \leq 0.9$. In the second type, relevant to the third set of experiments, we demonstrate the scalability of the algorithms by fixing the constraint density ($p_1 = 0.4$) and varying the number of agents n (following the respective experiments of Léauté & Faltings, 2013).

The third benchmark consists of scale-free networks, which are structured networks that are generated by the Barabási-Albert model (Barabási & Albert, 1999; Jackson, 2008). An initial set of four agents is randomly selected and connected. At each iteration of the Barabási-Albert procedure an agent is added and connected to two other agents with a probability that is proportional to the number of links that the existing agents already have. Every new agent is connected to just two existing agents in order to maintain the basic structure of the scale-free network given the rather small network size ($n \leq 20$). Larger networks are not applicable for complete algorithms. The size of domains is set to $|D_1| = \cdots = |D_n| = 5$. The significant parameter in these scale-free networks is the number of agents n. Consequently, we use this benchmark for demonstrating the scalability of the algorithms.

The fourth benchmark consists of distributed meeting scheduling problems, which are highly structured real-world problems. We construct the problems similarly to the PEAV (Private Events As Variables) formulation of Maheswaran et al. (2004), which is aimed for scenarios where privacy is a concern. The PEAV formulation generates multiple-variable agents, where each agent holds a variable for each meeting he participates in. For each meeting, a hard equality constraint is imposed over the corresponding variables owned by the participants of that meeting; this constraint enforces that the participants agree on the time for the meeting. The preferences of agents are represented as the costs of intra-agent binary constraints (Maheswaran et al., 2004). We follow the setting of Léauté and Faltings (2013), in which the number of meetings m is varied, while the number of participants per meeting is fixed to 2. For each meeting, participants are randomly drawn from a common pool of 3 agents. The goal is to assign a time to each meeting among $d = 8$ available time slots. However, in our experiments we use different intra-agent constraints. Léauté and Faltings addressed satisfaction problems, rather than optimization problems, and therefore used simple *allDifferent* constraints as the intra-agent constraints, so as to enforce that all meetings of an agent are scheduled at different times. Contrary to that, we address DCOPs and are interested in problems with meaningful constraint information. Thus, we consider two types of preferences – time preference (a cost of $0, \ldots, 3$ for each time slot) and meeting importance (a cost of $5, \ldots, 9$ for each meeting). The cost of an

intra-agent binary constraint (representing two meetings of the same agent) incorporates both types of preferences by averaging the time preferences of the two meetings and adding the importance of the least important meeting in case of a collision between the meetings. Such a collision means that the least important meeting will not take place. The treatment of the actual existence of a meeting as a soft constraint, rather than a hard one, is not new in DCOP research (Zivan, Okamoto, & Peled, 2014). The meeting scheduling benchmark is appropriate for demonstrating the scalability of the algorithms, since the state space (the Cartesian product of the domains of the decision variables) keeps increasing with the number of meetings/decisions to be made.

All the evaluated algorithms were implemented in the AgentZero simulator (Lutati, Gontmakher, Lando, Netzer, Meisels, & Grubshtein, 2014). The experiments were run on a hardware comprised of an Intel i7-4600U processor and 16GB memory. The presented results in all the subsequent figures are shown in a logarithmic scale and are the average over 50 problem instances (for each setting/benchmark).

The runtime of DCOP algorithms is commonly measured by logical operations, such as non-concurrent constraint checks (Zivan & Meisels, 2006). Since P-SyncBB includes arithmetic operations that are not constraint checks, we follow the *simulated time* approach of Sultanik, Lass, and Regli (2008) in all our runtime experiments. We report results below a cutoff time of 15 minutes. Namely, if in a given problem setting the run over a sequence of 50 problem instances did not complete in 12.5 hours, we do not report any result for that setting (with the exception of the P-DPOP$^{(+)}$ results in Figure 2, which enables a comparison over the full range of constraint densities in that experiment). A result for a given problem setting is omitted also in cases where the experiment ran out of the allocated 8GB heap memory.

5.2.1 Evaluated Algorithms

We have implemented the P-SyncBB algorithm according to Algorithm 2. We set the value of S (the size of the additive group $\mathbf{Z}_S := [0, S-1]$ in which we perform our secure protocols, see Section 4.1) to $S = 2^{256}$. Such a setting mandates the usage of multiple-precision integers.

In order to faithfully comprehend the effectiveness of the P-SyncBB algorithm, we compare its runtime to that of the recently proposed P-DPOP$^{(+)}$ algorithm, which is the most efficient (and at the same time the least privacy-preserving) private version of DPOP that was presented by Léauté and Faltings (2013).

DPOP (Petcu & Faltings, 2005) is an algorithm based on the dynamic programming paradigm. It is actually a distributed version of the general bucket elimination scheme proposed by Dechter (2003). DPOP has three phases. In the first phase it constructs a *pseudo-tree* (Freuder & Quinn, 1985), which is a tree-like structure that allows links (back-edges) between remote ancestors/descendants. The second phase consists of **UTIL** messages that propagate utilities up the pseudo-tree. The **UTIL** messages start from the leaves of the pseudo-tree and are passed only through the tree edges (not back-edges). Cycles in the constraint graph, represented by back-edges in the pseudo-tree, result in multidimensional **UTIL** messages. In fact, the size of messages grows exponentially in the induced width of the pseudo-tree, where the base of this exponentiation is the domain size.

The third phase is initiated when the utility propagation reaches the root agent. The root agent chooses his best value according to the received utilities, and sends **VALUE** messages to his children, informing them about his decision. **VALUE** messages are propagated down the tree until all agents choose their optimal values.

The privacy-preserving version of DPOP, P-DPOP$^{(+)}$, uses obfuscation techniques in order to protect the private information that could be leaked by the **UTIL** messages. These techniques include codenames and addition of random numbers. The use of codenames prevents P-DPOP$^{(+)}$ from joining dimensions of **UTIL** messages that originate from the same agent, which in turn leads to increased induced widths of the pseudo-trees, compared to those of the original DPOP.

Our comparison also includes a naïve privacy-preserving exhaustive-search algorithm that we call P-Ex, which is similar to P-SyncBB except that it does not do pruning. This is achieved by changing the command in line 30 of P-SyncBB to "**else**" and removing lines 33-42; other than that P-Ex coincides with P-SyncBB, including the usage of Protocol 3 and homomorphic encryption for privacy preservation.

In some of the experiments we also compare the performance of P-SyncBB and P-DPOP$^{(+)}$ to their respective original, non-privacy-preserving, versions, SyncBB and DPOP.

5.2.2 Comparison Between Privacy Preserving DCOP Algorithms

In this first set of experiments we chose the maximal values of n for which P-DPOP$^{(+)}$ could complete all its runs without running out of heap memory (heap size was set to 8GB). For the random DCOPs benchmark this value was $n = 6$, while for the graph coloring benchmark it was $n = 9$. (We note that P-DPOP$^{(+)}$ crashed due to lack of memory on some graph coloring instances with $n = 9$ and $p_1 = 0.9$, so these instances were excluded from the experiment.)

Figures 1 and 2 present the runtime performance on the random DCOPs and graph coloring benchmarks, respectively. As can be seen, the runtime performance of P-Ex is not affected by the problem density. This is because P-Ex performs exhaustive search with no pruning at all. In both benchmarks, the performance of P-SyncBB is superior to that of P-Ex even in the densest problems. This indicates that the relatively low computational costs of the proposed solutions to the millionaires' problems (Section 4.2.3) make pruning worthwhile.

In both benchmarks the runtime performance of P-DPOP$^{(+)}$ deteriorates as the problems become denser. This is a known phenomenon of the original non-privacy-preserving DPOP algorithm, since denser problems lead to higher induced widths of the pseudo-trees (Petcu & Faltings, 2005). As reported by Léauté and Faltings (2013), this problem is amplified in the case of P-DPOP$^{(+)}$, since the need to protect topology privacy results in increased induced widths of the pseudo-trees, compared to those of the original DPOP. Indeed, the experiments show that even the naïve algorithm P-Ex outperforms P-DPOP$^{(+)}$ on dense problems.

5.2.3 The Overhead of Privacy Preservation

In the second set of experiments we examine the overhead of privacy preservation by comparing the performance of P-SyncBB to that of the original non-privacy-preserving SyncBB

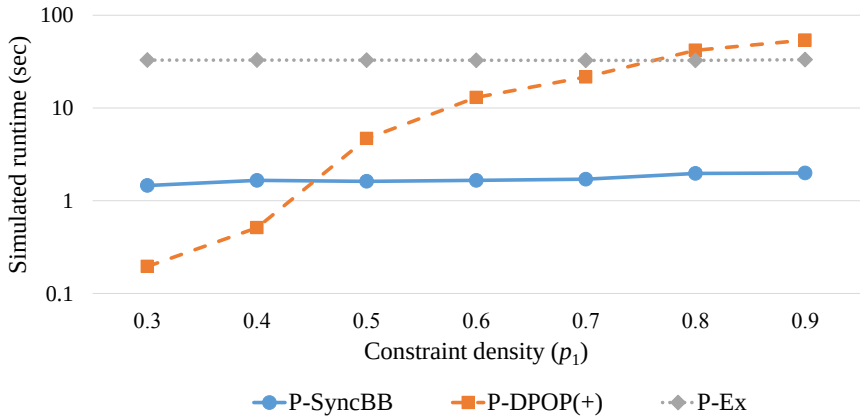

Figure 1: Runtime performance on random DCOPs ($n = 6$, $|D_1| = \cdots = |D_6| = 6$).

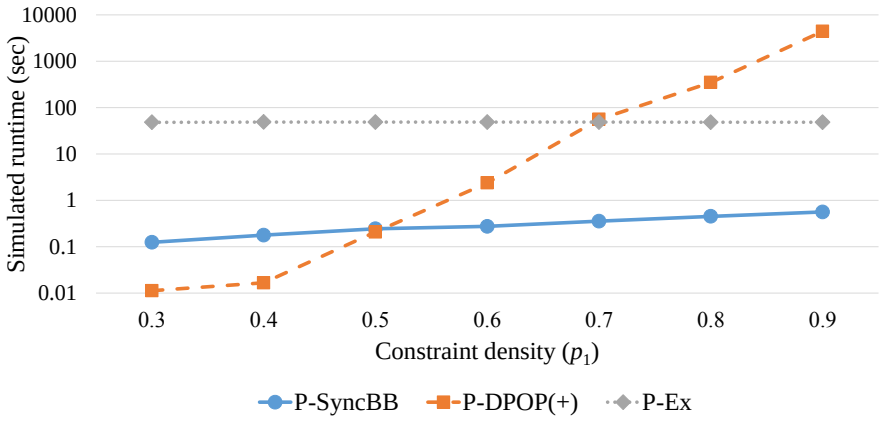

Figure 2: Runtime performance on 3-color graph coloring problems ($n = 9$).

algorithm on larger instances of the random DCOPs ($n = 9$) and graph coloring ($n = 16$) benchmarks. P-Ex and P-DPOP$^{(+)}$ were not able to solve these larger instances in reasonable time (cutoff time of 15 minutes per problem instance) and without running out of heap memory (8GB), and are thus omitted from this set of experiments.

Figures 3 and 4 present the runtime performance on the random DCOPs and graph coloring benchmarks, respectively. On both benchmarks the computation overhead of privacy preservation is a little over one order of magnitude.

Figures 5 and 6 present the network load on the random DCOPs and graph coloring benchmarks, respectively. We follow the common practice of measuring the network load by the total number of messages (Mailler & Lesser, 2004; Gershman et al., 2009; Gutierrez, Meseguer, & Yeoh, 2011). Such comparisons are not possible with algorithms of the DPOP family, since the size of messages in these algorithms is exponential in the induced width of the pseudo-tree (Petcu & Faltings, 2005, Thm. 1).

The communication overhead of privacy preservation is about 30 times on the random DCOPs benchmark and about 60-70 times on the graph coloring benchmark. While the

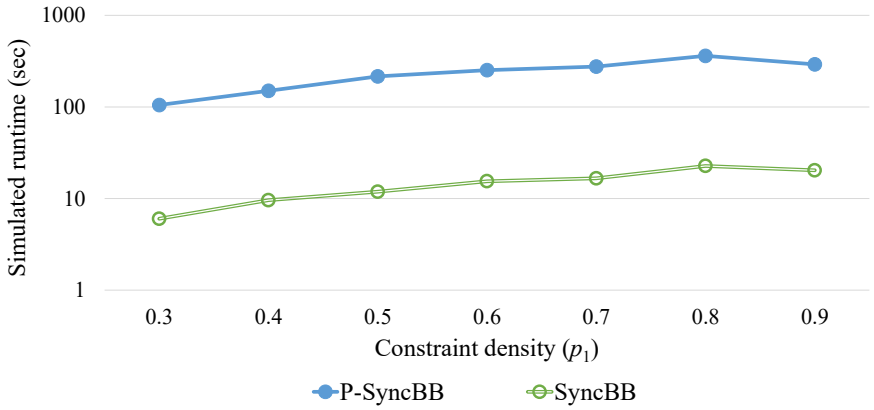

Figure 3: Runtime performance on random DCOPs ($n = 9$, $|D_1| = \cdots = |D_9| = 9$).

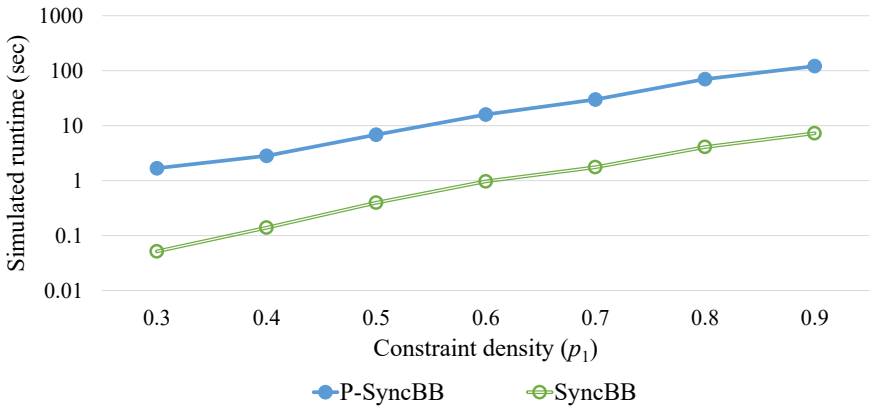

Figure 4: Runtime performance on 3-color graph coloring problems ($n = 16$).

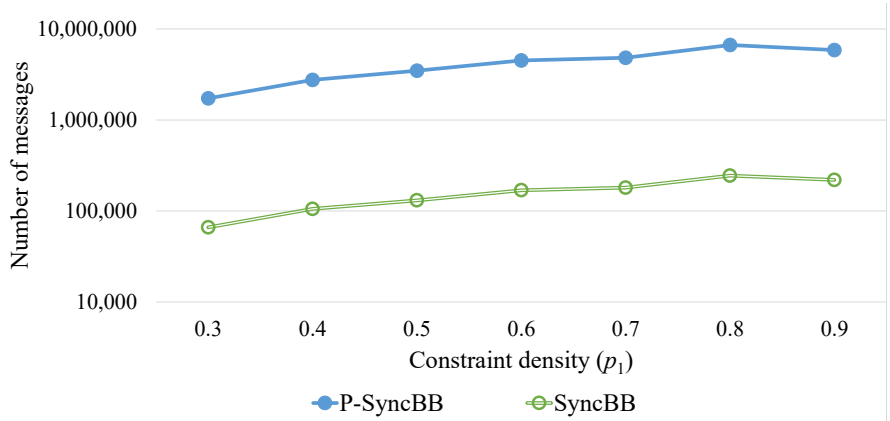

Figure 5: Network load on random DCOPs ($n = 9$, $|D_1| = \cdots = |D_9| = 9$).

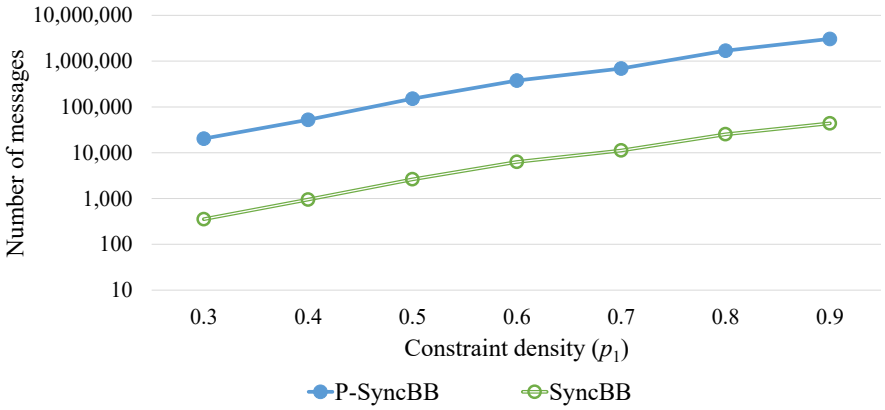

Figure 6: Network load on 3-color graph coloring problems ($n = 16$).

computation and communication overhead of P-SyncBB is significant, its effect remains almost the same as the problems become denser.

5.2.4 Scalability

In the third set of experiments we examine the scalability of P-SyncBB, and compare it to that of SyncBB, P-DPOP$^{(+)}$, DPOP, and P-Ex. This experiment also shows how P-SyncBB and P-DPOP$^{(+)}$ perform compared to their non-privacy-preserving counterparts.

The DPOP family of algorithms is known to be especially effective on very sparse problems, which have very low induced width. However, their performance deteriorates as the problems become denser. Thus, in order to conduct a thorough and fair examination, we use different benchmarks with various density levels.

Figures 7 and 8 present the scalability of runtime performance on unstructured random DCOPs with different constraint densities. The respective experiments are on sparse problems ($p_1 = 0.3$, Figure 7) and dense problem ($p_1 = 0.7$, Figure 8).

We may see that both SyncBB and P-SyncBB scale well. Contrary to that, P-DPOP$^{(+)}$ does not scale well, even in the sparse problems of Figure 7, where DPOP outperforms SyncBB. Moreover, even P-Ex scales better than P-DPOP$^{(+)}$. This phenomenon is amplified in the dense problems of Figure 8, due to the fact that P-Ex is completely unaffected by the density parameter. Another interesting observation is that P-SyncBB displays similar performance to that of P-Ex when $n = 4$ (Figure 8). This happens because in P-SyncBB pruning is not performed by the first 3 agents (see Algorithm 2, lines 30-32), and consequently for problems with $n \leq 4$ agents P-SyncBB is exactly the same as P-Ex.

Next, we examine the scalability of runtime performance on structured problems. Figure 9 presents the results on graph coloring problems. Figure 10 presents the results on scale-free networks.

The performance of P-DPOP$^{(+)}$ on graph coloring problems is consistent with the original experiments that were conducted for this algorithm (Léauté & Faltings, 2013, Figures 7 and 8). As in all other experiments, P-SyncBB scales similarly to SyncBB. As the number of agents increases, the runtime performance of P-SyncBB even becomes comparable

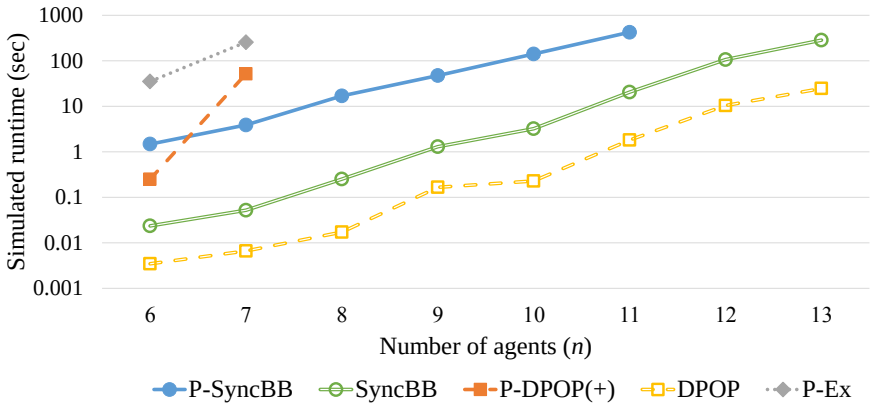

Figure 7: Runtime performance on sparse random DCOPs ($p_1 = 0.3$, $|D_1| = \cdots = |D_n| = 6$).

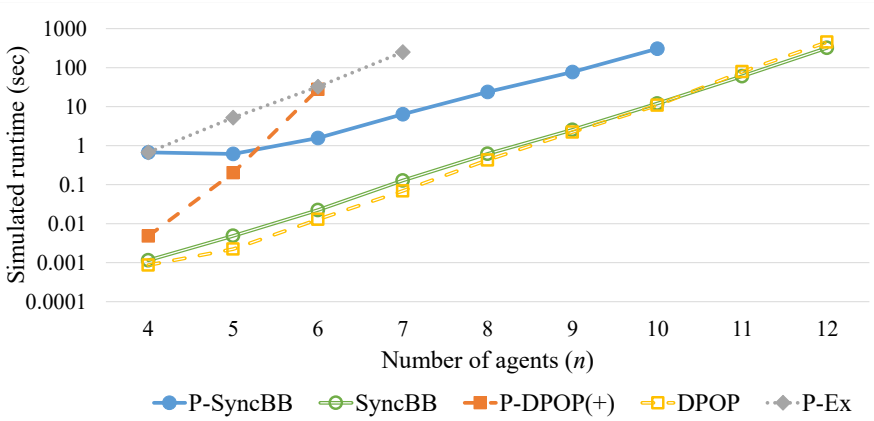

Figure 8: Runtime performance on dense random DCOPs ($p_1 = 0.7$, $|D_1| = \cdots = |D_n| = 6$).

to that of the non-privacy-preserving DPOP algorithm. It is also evident that even P-Ex scales better than P-DPOP$^{(+)}$.

The experiment on scale-free networks demonstrates the runtime performance in a domain that clearly favors the DPOP family of algorithms. The Barabási-Albert procedure leads to a network with few agents that are very connected. We apply a heuristic that sets one of these agents as the root of the pseudo-tree, which leads to very efficient pseudo-trees. We also chose to use rather small domains ($|D_1| = \cdots = |D_n| = 5$), a selection that also helps DPOP. Indeed, as evident in Figure 10, DPOP exhibits superior performance to SyncBB. Nonetheless, P-SyncBB outperforms P-DPOP$^{(+)}$ even in this domain. Yet, P-DPOP$^{(+)}$ outperforms P-Ex, because the latter cannot take advantage of the special topology of scale-free networks.

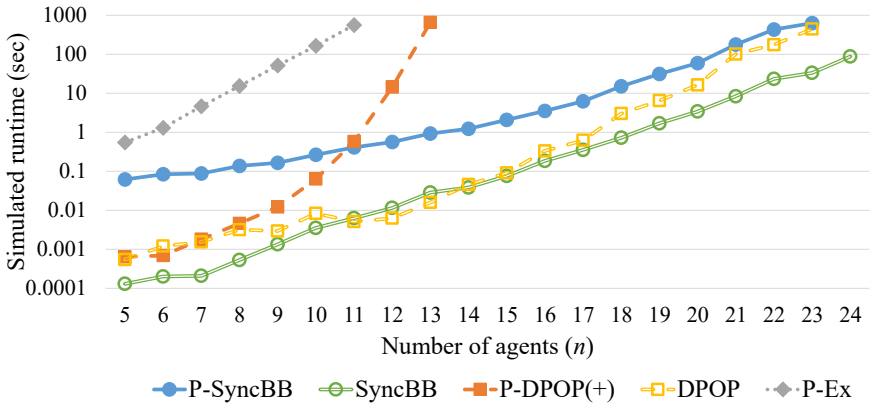

Figure 9: Runtime performance on 3-color graph coloring problems ($p_1 = 0.4$).

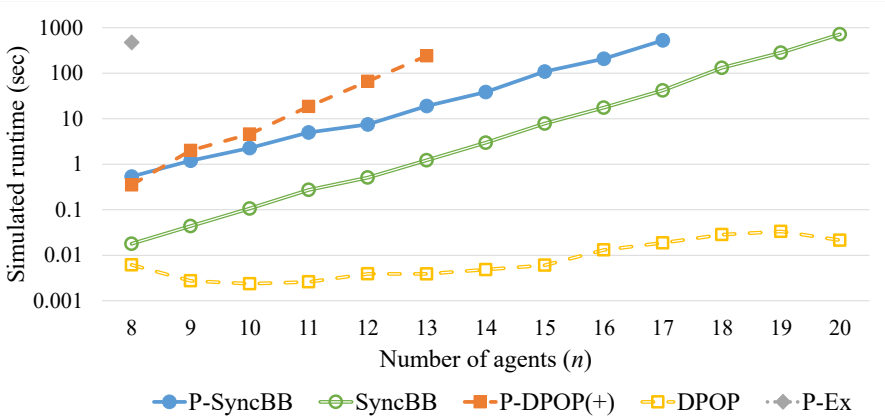

Figure 10: Runtime performance on scale-free networks ($|D_1| = \cdots = |D_n| = 5$).

Finally, we examine the scalability of runtime performance on real-world distributed meeting scheduling problems. These problems are not standard in the sense that they consist of agents with multiple variables. For simplicity and clarity reasons, the presentation of most DCOP algorithms assumed a single variable per agent, leaving the issue of multiple-variable agents to the implementer. In order to avoid falling into such implementation issues and conduct a fair comparison between the different algorithms, we follow the generic *decomposition* method of Yokoo and Hirayama (2000) that turns each variable into a *virtual agent*. Figure 11 reports the resulting runtime performance.

We see that in this setting both SyncBB and P-SyncBB scale much better than DPOP and P-DPOP$^{(+)}$. Moreover, P-SyncBB even outperforms the non-privacy-preserving DPOP algorithm when $m \geq 5$. An interesting phenomenon is observed when the number of meetings is small ($m \leq 2$), which leads to a small number of variables ($n \leq 4$) in the corresponding DCOP problem. P-SyncBB does not conduct any pruning in these instances, and therefore performs exactly as P-Ex. Nevertheless, as the number of meetings increases, P-SyncBB starts to effectively prune the search space, which in turn leads to much better

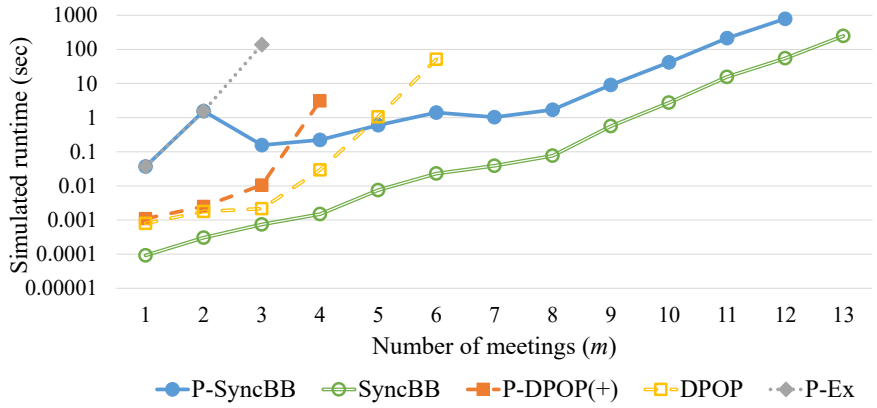

Figure 11: Runtime performance on distributed meeting scheduling problems.

scalability. Moreover, the performance of P-SyncBB actually improves when the number of meetings increases from $m = 2$ to $m = 3$ because then some of the search space is pruned, which in turn prevents many costly encryptions (see Algorithm 2, line 18).

6. Conclusion

We presented here P-SyncBB, a privacy-preserving version of the SyncBB algorithm for solving DCOPs while respecting constraint, topology and decision privacy. A fundamental ingredient in the design of P-SyncBB is the separation of information that prevents any agent at any stage from knowing both the upper bound and the current partial assignment (CPA). This separation is complemented by the invoking of secure multi-party protocols for computing the costs of CPAs and comparing them to the current upper bound. To this end, we devised special protocols that solve instances of the millionaires' problem securely without resorting to costly oblivious transfer sub-protocols. We then used the devised sub-protocols in Protocol 5, that compares the cost of a CPA, which is shared between two agents, to the upper bound which is held by only one of them.

Our extensive experimental evaluation showed that while the runtime of P-SyncBB is significantly higher than that of SyncBB, it exhibits superior performance to P-DPOP$^{(+)}$ of Léauté and Faltings (2013), which is the state-of-the-art privacy-preserving complete DCOP algorithm.

The key operations that require security in any search-based DCOP algorithm are the summation and comparison operations. Therefore, we believe that the secure multi-party protocols of Section 4, together with the aforementioned separation of information, may serve as the basis for developing additional search-based privacy-preserving DCOP algorithms. However, contrary to SyncBB, algorithms such as AFB (Gershman et al., 2009) or NCBB (Chechetka & Sycara, 2006), include asynchronicity and parallelism. Consequently, applying the ideas of P-SyncBB in order to devise privacy-preserving versions of these algorithms may not be straightforward, since the reliance of P-SyncBB on agent A_1 and on the third party T may create bottlenecks with those agents in an asynchronous/parallel envi-

ronment. Thus, we consider the development of additional search-based privacy-preserving DCOP algorithms an interesting and challenging prospect for future work.

Another interesting direction for future research is to extend our work to the case of asymmetric constraints (Grinshpoun et al., 2013). Finally, in view of the discussion in Section 4.5, it would be desirable to enhance P-SyncBB so that it becomes immune against coalitions, while minimizing the price that such enhancement entails in terms of communication and computational costs.

Acknowledgments

The authors would like to thank Vadim Levit for his help with the implementation of P-DPOP$^{(+)}$, and the anonymous reviewers for their insightful comments.

References

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509–512.

Benaloh, J. (1986). Secret sharing homomorphisms: Keeping shares of a secret secret. In *Crypto*, pp. 251–260.

Blake, I. F., & Kolesnikov, V. (2004). Strong conditional oblivious transfer and computing on intervals. In *ASIACRYPT*, pp. 515–529.

Brito, I., Meisels, A., Meseguer, P., & Zivan, R. (2009). Distributed constraint satisfaction with partially known constraints. *Constraints*, *14*(2), 199–234.

Chechetka, A., & Sycara, K. (2006). No-commitment branch and bound search for distributed constraint optimization. In *AAMAS*, pp. 1427 – 1429.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufman.

Doshi, P., Matsui, T., Silaghi, M. C., Yokoo, M., & Zanker, M. (2008). Distributed private constraint optimization. In *WI-IAT*, pp. 277–281.

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

Faltings, B., Léauté, T., & Petcu, A. (2008). Privacy guarantees through distributed constraint satisfaction. In *WI-IAT*, pp. 350–358.

Faltings, B., & Macho-Gonzalez, S. (2005). Open constraint programming. *Artificial Intelligence*, *161:1-2*, 181–208.

Fischlin, M. (2001). A cost-effective pay-per-multiplication comparison method for millionaires. In *Topics in Cryptology - CT-RSA*, pp. 457–472.

Freuder, E. C., & Quinn, M. J. (1985). Taking advantage of stable sets of variables in constraint satisfaction problems.. In *IJCAI*, pp. 1076–1078.

Gershman, A., Zivan, R., Grinshpoun, T., Grubshtein, A., & Meisels, A. (2008). Measuring distributed constraint optimization algorithms. In *DCR Workshops*, pp. 17–24.

Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, *34*, 61–88.

Greenstadt, R., Pearce, J., & Tambe, M. (2006). Analysis of privacy loss in distributed constraint optimization. In *AAAI*, pp. 647–653.

Greenstadt, R., Grosz, B., & Smith, M. D. (2007). SSDPOP: improving the privacy of DCOP with secret sharing. In *AAMAS*, pp. 171:1–171:3.

Grinshpoun, T., & Meisels, A. (2008). Completeness and performance of the APO algorithm. *Journal of Artificial Intelligence Research*, *33*, 223–258.

Grinshpoun, T. (2012). When you say (DCOP) privacy, what do you mean? - categorization of DCOP privacy and insights on internal constraint privacy. In *ICAART*, pp. 380–386.

Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., & Meisels, A. (2013). Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, *47*, 613–647.

Grinshpoun, T., & Tassa, T. (2014). A privacy-preserving algorithm for distributed constraint optimization. In *AAMAS*, pp. 909–916.

Gutierrez, P., Meseguer, P., & Yeoh, W. (2011). Generalizing ADOPT and BnB-ADOPT. In *IJCAI*, pp. 554–559.

Hirayama, K., & Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In *CP*, pp. 222–236.

Ioannidis, I., & Grama, A. (2003). An efficient protocol for yao's millionaires' problem. In *HICSS*, p. 205.

Jackson, M. O. (2008). *Social and Economic Networks*. Princeton University Press.

Jeckmans, A., Tang, Q., & Hartel, P. H. (2012). Privacy-preserving collaborative filtering based on horizontally partitioned dataset. In *CTS*, pp. 439–446.

Jiang, W., & Clifton, C. (2006). A secure distributed framework for achieving k-anonymity. *The VLDB Journal*, *15*, 316–333.

Kantarcioglu, M., & Clifton., C. (2004). Privacy-preserving distributed mining of association rules on horizontally partitioned data. *Transactions on Knowledge and Data Engineering*, *16*, 1026–1037.

Léauté, T., & Faltings, B. (2011). Distributed constraint optimization under stochastic uncertainty. In *AAAI*, pp. 68–73.

Léauté, T., & Faltings, B. (2013). Protecting privacy through distributed computation in multi-agent decision making. *Journal of Artificial Intelligence Research*, *47*, 649–695.

Lindell, Y., & Pinkas, B. (2000). Privacy preserving data mining. In *Crypto*, pp. 36–54.

Lutati, B., Gontmakher, I., Lando, M., Netzer, A., Meisels, A., & Grubshtein, A. (2014). Agentzero: A framework for simulating and evaluating multi-agent algorithms. In *Agent-Oriented Software Engineering*, pp. 309–327. Springer.

Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., & Varakantham, P. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS*, pp. 310–317.

Maheswaran, R., Pearce, J., Bowring, E., Varakantham, P., & Tambe, M. (2006). Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems*, *13*, 27–60.

Mailler, R., & Lesser, V. R. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pp. 438–445.

Modi, J., & Veloso, M. (2004). Multiagent meeting scheduling with rescheduling. In *DCR Workshops*.

Modi, P. J., Shen, W., Tambe, M., & Yokoo, M. (2005). ADOPT: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, *161*, 149–180.

Nissim, K., & Zivan, R. (2005). Secure DisCSP protocols - from centralized towards distributed solutions. In *DCR Workshops*.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, pp. 223–238.

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *IJCAI*, pp. 266–271.

Rabin, M. O. (1981). How to exchange secrets by oblivious transfer. Tech. rep. TR-81, Aiken Computation Laboratory, Harvard University.

Rivest, R. L., Shamir, A., & Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*, 120–126.

Schuster, A., Wolff, R., & Gilburd, B. (2004). Privacy-preserving association rule mining in large-scale distributed systems. In *CCGrid*, pp. 411–418.

Silaghi, M. C., Faltings, B., & Petcu, A. (2006). Secure combinatorial optimization simulating DFS tree-based variable elimination. In *ISAIM*.

Silaghi, M. C., & Mitra, D. (2004). Distributed constraint satisfaction and optimization with privacy enforcement. In *IAT*, pp. 531–535.

Sultanik, E., Lass, R. N., & Regli, W. C. (2008). DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *AAMAS (demos)*, pp. 1667–1668.

Tassa, T. (2014). Secure mining of association rules in horizontally distributed databases. *Transactions on Knowledge and Data Engineering*, *26*, 970–983.

Tassa, T., & Gudes, E. (2012). Secure distributed computation of anonymized views of shared databases. *Transactions on Database Systems*, *37, Article 11*.

Tassa, T., & Bonchi, F. (2014). Privacy preserving estimation of social influence. In *EDBT*, pp. 559–570.

Tassa, T., & Cohen, D. J. (2013). Anonymization of centralized and distributed social networks by sequential clustering. *Transactions on Knowledge and Data Engineering*, *25*, 311–324.

Tassa, T., Zivan, R., & Grinshpoun, T. (2015). Max-sum goes private. In *IJCAI*, pp. 425–431.

Tassa, T., Zivan, R., & Grinshpoun, T. (2016). Preserving privacy in region optimal DCOP algorithms. In *IJCAI*, pp. 496–502.

Vaidya, J., & Clifton, C. (2002). Privacy preserving association rule mining in vertically partitioned data. In *KDD*, pp. 639–644.

Yakut, I., & Polat, H. (2012). Arbitrarily distributed data-based recommendations with privacy. *Data & Knowledge Engineering*, *72*, 239–256.

Yao, A. (1982). Protocols for secure computation. In *FOCS*, pp. 160–164.

Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.

Yokoo, M., Suzuki, K., & Hirayama, K. (2005). Secure distributed constraints satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, *161*, 229–246.

Yokoo, M., & Hirayama, K. (2000). Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, *3*(2), 185–207.

Zhan, J., Matwin, S., & Chang, L. (2005). Privacy preserving collaborative association rule mining. In *Data and Applications Security*, pp. 153–165.

Zhang, W., Xing, Z., Wang, G., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: Properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, *161*, 55–88.

Zhong, S., Yang, Z., & Wright, R. (2005). Privacy-enhancing k-anonymization of customer data. In *PODS*, pp. 139–147.

Zivan, R., & Meisels, A. (2006). Message delay and DisCSP search algorithms. *Annals of Mathematics and Artificial Intelligence*, *46*, 415–439.

Zivan, R., Okamoto, S., & Peled, H. (2014). Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, *212*, 1–26.