A Privacy-Preserving Algorithm for Distributed Constraint Optimization

Tal Grinshpoun*
Dept. of Industrial Engineering and Management
Ariel University
Ariel, Israel
talgr@ariel.ac.il

Tamir Tassa
Dept. of Mathematics and Computer Science
The Open University
Ra'anana, Israel
tamirta@openu.ac.il

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 algorithm for this purpose. The proposed algorithm requires a secure solution of several multiparty computation problems. Consequently, appropriate novel secure protocols are devised and analyzed.

Categories and Subject Descriptors

[**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence*

General Terms

Algorithms

Keywords

DCOP, Constraint Privacy, Search

1. INTRODUCTION

Distributed constraint optimization (DCOP) [13, 21] 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 [20], sensor nets [31], and vehicle routing [15], resulted in the development of various complete DCOP algorithms. Most of those algorithms, such as SyncBB [13], ADOPT [21], NCBB [2], AFB [6], and BnB-ADOPT [29], are based on *search*. Other paradigms for solving DCOPs include grouping of sub-problems (OptAPO [19]) and dynamic programming (DPOP [23]).

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 [5, 8, 10]. Only one privacy type is common to these various categorizations – constraint privacy, meaning that constraint information must be known only to the agents whose variables are involved in the constraints. Indeed, most research on DCOP privacy focused on constraint privacy.

Some work has focused on measuring the extent of constraint privacy loss. Most notably is Maheswaran et al. [17] 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 [9]. Doshi et al. proposed to inject privacy-loss as a criterion to the problem solving process [3]. 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. Examples include [22, 25, 30]. The work of Silaghi and Mitra [25] 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 [8, 24]. Recently, Léauté and Faltings [16] 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 [18] and *asymmetric* DCOPs [11]. Here, we restrict ourselves to the traditional symmetric DCOPs.

In the present paper we devise a novel DCOP algorithm that preserves constraint privacy. The new algorithm is based on SyncBB [13], 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 the size of messages grows exponentially in the arity of the constraints [23]; this fact renders DPOP inapplicable for solving dense problems [7]. 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 [12].

[*] Double affiliation with Dept. of Mathematics and Computer Science, The Open University, Ra'anana, Israel.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

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 on 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 [28] that does not depend on costly oblivious transfer [4] sub-protocols.

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 computation analysis of the security protocols, as well as experimental evaluation of the overall performance of the search algorithm. Conclusions are drawn in Section 6.

2. PRELIMINARIES

2.1 Distributed constraint optimization

A Distributed Constraint Optimization Problem (DCOP) [13] 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 $C : D_{i_1} \times \cdots \times D_{i_k} \to \mathbb{R}_+ := [0, \infty)$, for some $1 \leq i_1 < \cdots < i_k \leq m$.

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 objective is to find a full assignment of minimal cost.

For simplicity, and due to page limitation, 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. For the same reasons 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 $C_{i,j} : D_i \times D_j \to \mathbb{R}_+$. These assumptions are customary in DCOP literature (e.g., [21, 23]).

2.2 Security notions

A secure multiparty protocol is a protocol that enables several parties to jointly compute a function over their inputs, while at the same time keeping these inputs private. Such a protocol is considered *perfectly secure* if it does not reveal to any of the interacting parties any information on the private inputs of his peers beyond what is implied by his own input and the final output of the protocol. This notion of security coincides with the disclosure of *semi-private information*, a concept that was suggested in the context of distributed constraint satisfaction [5].

When looking for practical solutions, however, some relaxations of the notion of perfect privacy are usually inevitable, provided that the excess information is deemed benign (see examples of such protocols in [14, 26, 27, 32]). 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.

3. PRIVACY-PRESERVING DCOP SEARCH

Synchronous Branch-and-Bound (SyncBB) [13] was the first complete algorithm 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 [2] and AFB [6], 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 privacy. Namely, the value of $C_{i,j}(a_i, a_j)$, for any two assignments $a_i \in D_i$ and $a_j \in D_j$, should not be disclosed to any agent other than A_i and A_j. We proceed to present the Privacy-preserving Synchronous Branch-and-Bound algorithm (P-SyncBB).

3.1 The P-SyncBB algorithm

When considering a version of SyncBB that preserves constraint 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 the costs of partial assignments and the upper bound, while still enabling pruning of the search space. Such separation is achieved by preventing any agent at any stage from knowing both the upper bound and the *current partial assignment* (CPA).

We assume that there is a static public ordering of the agents, $A_1, \ldots, A_n$. Agent A_1 is the perfect candidate to hold the upper bound, since according to the original SyncBB algorithm, A_1 never sees the assignments of the other agents. The other agents receive the CPA from their preceding agents throughout the search process, so at each stage they know the assignments of the agents preceding them in the order. Thus, in order to enable the above desired separation, agents $A_2, \ldots, A_n$ must not be aware of the upper bound.

In the original SyncBB algorithm, the CPA traverses the search space by each agent assigning a value to its variable and passing the augmented 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 found thus far. Whenever an agent sees that the cost of the CPA combined with the cost added by his variable reaches the upper bound, he prunes this branch of the search space by assigning the next value to his variable without sending the CPA to the succeeding agent.

Contrary to the case of SyncBB, in the P-SyncBB algorithm the cost of the CPA must not be known to any agent, since the cost of a CPA may leak private constraint information. Consequently, at each pruning decision point, the agent holding the CPA must compare between the cost of the CPA (including the cost added by his own assignment) and the upper bound in a secure manner, i.e., without learning neither the cost of the CPA nor the upper bound. Such nontrivial task is achieved by restraining the information flow in the algorithm and by introducing secure protocols for summation and comparison. The pseudo-code of P-SyncBB is given in Algorithm 1 and described in detail next, whereas a full description of the secure protocols is given in Section 4.

The run of P-SyncBB starts by agent A_1 initially setting the upper bound B to infinity (line 2). Agent A_1 also sets the flag *ComputedCPA* to **true** (line 3). This flag indicates whether an agent has already computed the cost of the CPA that was received from his preceding agent. Since A_1 is the first in order he never needs to compute this cost, so he initiates the flag accordingly. Next, A_1 runs the **assign_CPA** procedure. According to the initial value of *ComputedCPA* and the fact that at this preliminary stage there is no possibility of backtracking, A_1 simply chooses a value from his variable's domain (line 21), assigns it to his variable in the CPA (line 35), and sends a **CPA_MSG** with the updated CPA to agent A_2 (line 36). In lines 31-34, the current agent checks whether the cost of the CPA which was conveyed to him from his preceding agent is already too large and thus this branch of the search can be pruned. Those lines are irrelevant for A_1 so he skips them. For privacy reasons it may be decided that also agents A_2 and A_3 skip them too (see Section 4.2.2). These special cases were discarded from the pseudo-code for clarity reasons.

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

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

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 *ComputedCPA* flag is set to **false** (line 8), before the agent runs the **assign_CPA** procedure.

The **assign_CPA** procedure begins with the agent choosing the next value from the pre-ordered domain (line 21). In case no values remain, the agent backtracks (lines 22-23). Any agent which is not the last one A_n skips to line 28. We will relate to agent A_n (lines 24-26) later. In case the *ComputedCPA* flag is **false**, the agent 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)), and sets the flag to **true** (lines 28-30). The secure computation in line 29 is performed by invoking Protocol 3 (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. Hence, the actual cost is never disclosed to any single agent.

Algorithm 1 - P-SyncBB (executed by agent A_k)

```
procedure init
 1: if $A_k = A_1$ do
 2:     $B \leftarrow \infty$
 3:     $ComputedCPA \leftarrow$ true
 4:     $assign\_CPA()$

when received (CPA_MSG,$PA$) do
 5: $CPA \leftarrow PA$
 6: for each $v \in D_k$ do compute $x_k$
 7: order values of $D_k$ according to the computed $x_k$'s
 8: $ComputedCPA \leftarrow$ false
 9: assign_CPA()

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

when received (CHECK_SOLUTION_MSG) do
12: $Cost \leftarrow$ invoke Protocol 2 to securely compute the solution's cost
      (i.e., $\sum_{i=2}^{n} x_i$)
13: if $Cost < B$ do
14:     $B \leftarrow Cost$
15:     send(SOLUTION_ANS_MSG,true) to $A_n$
16: else
17:     send(SOLUTION_ANS_MSG,false) to $A_n$

when received (SOLUTION_ANS_MSG,$Answer$) do
18: if $Answer =$ true do
19:     $BestSolution \leftarrow CPA$
20: backtrack()

procedure assign_CPA
21: choose next value $v$ in order from $D_k$
22: if no value exists do
23:     backtrack()
24: else if $A_k = A_n$ do
25:     $X_k \leftarrow v$
26:     send(CHECK_SOLUTION_MSG) to $A_1$
27: else
28:     if $ComputedCPA =$ false do
29:         invoke Protocol 3 to securely compute $CPA.cost$ (i.e.,
            $\sum_{i=2}^{k-1} x_i$)
30:         $ComputedCPA \leftarrow$ true
31:     $ShouldBacktrack \leftarrow$ invoke Protocol 4 to securely check
        whether $CPA.cost + x_k \geq B$
32:     if $ShouldBacktrack =$ true do
33:         backtrack()
34:     else
35:         $X_k \leftarrow v$
36:         send(CPA_MSG,$CPA$) to $A_{k+1}$

procedure backtrack
37: if $A_k = A_1$ do
38:     broadcast(TERMINATE)
39: else
40:     remove $X_k$ from $CPA$
41:     send(BACKTRACK_MSG) to $A_{k-1}$
```

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 reaches the upper bound. If so, then the pruning of this branch of the search space is achieved by backtracking. 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 4 (Section 4.2) in line 31. In case the output of Protocol 4 is **true**, the agent backtracks (lines 32-33). Otherwise, he assigns his variable in the CPA and sends a **CPA_MSG** to the subsequent agent (lines 35-36).

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 3). Whenever A_k *changes* his assignment to X_k, only Protocol 4 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.

Now we return to the case of the last agent A_n (lines 24-26). In this case, the CPA is actually a full assignment, so a **CHECK_SOLUTION_MSG** is sent to agent A_1 who can verify whether the CPA's cost is better than the cost of the best solution that was found thus far.

When agent A_1 receives the **CHECK_SOLUTION_MSG**, he invokes Protocol 2 (Section 4.1) with $k = n$ in order to compute the cost of the candidate solution (line 12). Due to the sequential operation of P-SyncBB, as each agent A_k, $k \geq 2$, has a local version of the CPA, he knows the current assignments $a_1, \ldots, a_k$ of $X_1, \ldots, X_k$. Hence, the cost of the CPA equals the sum of the private values x_k, $k \geq 2$, as defined in Eq. (1). (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 is lower than the existing upper bound, the new upper bound is updated (line 14). Finally, A_1 notifies A_n whether this is a new best solution or not by sending an appropriate **SOLUTION_ANS_MSG**.

When agent A_n, who knows the full assignment $(a_1, \ldots, a_n)$, receives the **SOLUTION_ANS_MSG**, he checks the input *Answer* from A_1 and updates the best solution in case it is **true** (lines 18-19). Regardless of the value of *Answer*, A_n then backtracks (line 20), since he does not need to check any of the subsequent values in his domain as they will issue worse solutions (due to the value pre-ordering).

Next, 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 37-38). Any other agent just removes his variable from the CPA (line 40) and sends a **BACKTRACK_MSG** to the preceding agent (line 41). An agent that receives a **BACKTRACK_MSG** removes his value from the CPA (line 10) and continues the search by assigning the next value in order (line 11).

At the end, the full assignment of the best solution is known only to A_n and its cost only to A_1. According to the application and its privacy requirements, agents A_1 and A_n can decide what to share with their peers. In any case, A_n must at least inform each of the other agents of their own value assignments in the best solution.

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

Proof. The soundness and completeness of P-SyncBB follow from those of SyncBB. The details of the standard proof are omitted due to page limitations. □

4. SECURE PROTOCOLS

In this section we present the secure protocols that are invoked by P-SyncBB, Algorithm 1. We assume herein that $C_{i,j}(a_i, a_j) \leq c$ for some publicly known c, for all $1 \leq i < j \leq n$ and $a_i \in D_i$, $a_j \in D_j$. Hence, $C := \binom{n}{2}c$ is a publicly known upper bound on the cost of any partial or full assignment. In Section 4.1 we discuss the secure summation protocols that are invoked in lines 12 and 29 of P-SyncBB. Our main cryptographic contribution is given in Section 4.2, which describes the secure comparison protocol that is invoked in line 31.

4.1 Secure summation protocols

The computational problem in line 12 of P-SyncBB is the classical problem of secure summation of private integers. It can be solved by a slight modification of Benaloh's protocol [1], which is described in Protocol 2. Here, S denotes an arbitrary integer much larger than C (the upper bound on the inputs and their sum).

Protocol 2 - Secure summation of private inputs

```
Input: $C, S \in \mathbb{Z}^+$ such that $S \gg C$;
Agent $A_i$, $2 \leq i \leq k$, has an integer $x_i \in [0, C]$, such that $x := \sum_{i=2}^{k} x_i \leq C$.
Output: $A_1$ gets $x$.
1: Each $A_i$, $2 \leq i \leq k$, selects random values $x_{i,j} \in \mathbb{Z}_S := [0, S-1]$,
   $2 \leq j \leq k$, 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 2 is perfectly secure: none of the agents $A_2, \ldots, A_k$ learns any information about the inputs of his peers, nor about 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 breaks up his input to modular additive shares which are chosen independently and uniformly at random. We omit further details of this standard proof. □

Next, we discuss the secure implementation of line 29 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 line 12 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 line 29, 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 3 ends with agents A_1 and A_k sharing that sum.

Protocol 3 - Computing additive shares in the sum of private inputs

```
1: Agents $A_2, \ldots, A_{k-1}$ perform Steps 1-3 of Protocol 2 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 3 starts by implementing the first three steps of Protocol 2 (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 $\mathbb{Z}_S$ (as a result of the uniform selection of shares $x_{i,j}$ in Step 1 of Protocol 2) and $s_2 + s_k = x \bmod S$. Protocol 3, which is a small variant of Protocol 2, is also perfectly secure, as it does not reveal to the interacting agents any information about the sum x or about the inputs of the other agents.

4.2 A secure comparison protocol

The main computational problem occurs in line 31 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 3 in line 29, $CPA.cost$ is split between 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 4 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 $[0, S-C-1]$ (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 $x - B$.)

Our goal now is to check whether $\delta := x - B$ is negative or not. In Case 1 $y = \delta + r$ while in Case 2 $y = \delta + r + S$. Since $x, B \in [0, C]$ then $\delta \in [-C, C]$. Hence, in Case 1 $y - r \leq C$ while in Case 2 we have $y - r \geq S - C$ (where $S - C \gg C$). 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 $y \geq S - C + r$ (Step 5). Since y is known only to A_1 while $S - C + r$ is known only to A_k, this is an instance of the millionaires' problem, which can be solved securely by Yao's garbled circuit protocol [28]. 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 $y \geq S - C + r$ 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 4 - Comparing a shared sum against an unknown bound

```
1: $A_k$ sets $s_k \leftarrow s_k + x_k \bmod S$.
2: $A_k$ generates uniformly at random an integer $r \in [0, S-C-1]$.
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 $y \geq S - C + r$. 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

Protocol 4 is "almost" perfectly secure in the following sense.

THEOREM 3. *At the end of Protocol 4 agent A_1 may learn either a lower bound on x, in probability at most $C/(S-C)$, or an upper bound, in probability at most $C/(S-C)$, or nothing at all, in probability at least $(S-3C)/(S-C)$. As for agent A_k, he may learn either a lower bound on x, in probability x/S, or an upper bound, in probability $(C-x)/S$, or nothing at all, in probability $(S-C)/S$.*

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 $x + r := z$. As $x = z - r$ and $0 \leq r \leq S - C - 1$, it follows that $z - (S - C - 1) \leq x \leq z$. Since it is known upfront that $0 \leq x \leq C$, the upper bound z reveals new information on x only when $z < C$. For every $i \in \{0, 1, \ldots, C-1\}$, the probability $\Pr(z = i) = \sum_{j=0}^{i} \Pr(x = j) \cdot \Pr(r = i - j)$. Since r distributes uniformly on $[0, S-C-1]$, it follows that $\Pr(z = i) = \frac{1}{S-C} \sum_{j=0}^{i} \Pr(x = j) \leq \frac{1}{S-C}$. Hence, $\Pr(z < C) = \sum_{i=0}^{C-1} \Pr(z = i) \leq \frac{C}{S-C}$. Therefore, A_1 may learn a non-trivial upper bound on x in probability no greater than $C/(S-C)$. Similarly, we can show that he may learn a non-trivial lower bound in probability no greater than $C/(S-C)$. The last two probability inequalities imply that in probability at least $(S-3C)/(S-C)$, the value of z does not allow A_1 to exclude any possible value of x in the range $[0, C]$. In addition, the value of z does not induce an a-posteriori belief probability distribution on x that differs from the belief probability distribution that A_1 had prior to seeing z, owing to the uniform random manner in which the masking value r is chosen. Next, we turn to discuss A_k.

A_k learns in Step 5 of Protocol 4 whether Case 1 holds ($s_2 + s_k = x$) or Case 2 does ($s_2 + s_k = S + x$). In Case 1, $0 \leq s_2, s_k \leq x$. We note that s_k distributes uniformly at random over $\mathbb{Z}_S$: Indeed, that is the case prior to Step 1, since then s_k is the sum of random $\mathbb{Z}_S$-shares (see Step 3 in Protocol 3); hence, even after adding to it x_k it remains uniformly distributed over $\mathbb{Z}_S$. Consequently, since the case $s_2 + s_k < S$ occurs if and only if $s_k \in \{0, 1, \ldots, x\}$, its probability is $(x+1)/S$. In that case, A_k may infer that $x \geq s_k$. That lower bound is non-trivial only when $s_k > 0$. Hence, A_k learns a (non-trivial) lower bound on x in probability x/S. In Case 2, on the other hand, both s_2 and s_k are strictly greater than x (since if, say, $s_1 \leq x$, then $s_k = x - s_1 \leq x$). Hence, in that case $x \leq s_k - 1$. Only when $s_k \leq C$, that upper bound is non-trivial. Therefore, A_k learns a (non-trivial) upper bound on x if and only if $x < s_k \leq C$, namely, in probability $(C-x)/S$. However, when $s_k > C$, A_k can learn nothing on x. To summarize, A_k learns nothing on x when $s_k = 0$ or when $s_k > C$. Since s_k distributes uniformly in $\mathbb{Z}_S$, A_k learns no information at all in probability $(S-C)/S$. □

Theorem 3 implies that the only potential leakages of information are to only two agents. 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. Moreover, the probability of those potential leakages to occur can be made negligible since C is a given integer and S can be chosen arbitrarily large.

It is worth noting that even though A_1 plays a pivotal role in this protocol, since 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.

As for A_k, beyond the negligible probability that he learns a lower or an upper bound on x, he also learns whether $x < B$ or $x \geq B$, without actually knowing the value of B. x is the cost of the current CPA and it equals $x = \sum_{1 \leq i < j \leq k-1} C_{i,j}(a_i, a_j) + \sum_{i=1}^{k-1} C_{i,k}(a_i, a_k) := C_1 + C_2$. Since A_k knows $a_1, \ldots, a_k$ (the current assignments to X_1, $\ldots, X_k$), he knows C_2. He wishes to extract additional information about C_1. If A_k learns that $x < B$ he infers that $C_1 < B - C_2$, while if he learns that $x \geq B$ he infers that $C_1 \geq B - C_2$. Since the value of B is not known to A_k, the above inferences are meaningless. If at the end of P-SyncBB the final value of B is still not published, then those inferences that were collected by A_k throughout the execution of P-SyncBB remain harmless. However, assume that A_1 shares with his peers the final value of B that was found, say B_0. A_k knows that all intermediate values of B were greater than or equal to B_0. Therefore, while inequalities of type $C_1 < B - C_2$ cannot yield any further information, inequalities of type $C_1 \geq B - C_2$ will imply that $C_1 \geq B_0 - C_2$.

Due to the above potential leakage of information on costs of CPAs (only in cases where the cost B of the optimal solution has to be published), we may decide that in P-SyncBB the agents perform pruning only for $k \geq k_0$ for some setting of k_0. Namely, agents $A_1, \ldots, A_{k_0-1}$ skip lines 31-34 and therefore do not invoke Protocol 4. As a consequence, only agents A_k with $k \geq k_0$ may infer lower bounds on sums of the form $\sum_{1 \leq i < j \leq k-1} C_{i,j}(a_i, a_j)$. Higher values of k_0 increase privacy (since then the above sum involves more constraints) but obviously reduces efficiency.

4.2.3 Solving the millionaires' problem

Let A and B be two agents; A holds a private integer a and B holds a private integer b. They wish to test whether $b \geq a$ without learning any information on the difference $b - a$. Both agents know that a, b can take values in some interval $[L, U]$, and, furthermore, that $b \geq a$ if and only if $b - a \in [0, K]$ for some known and small constant K. This setting is the case in Steps 5 and 6 of Protocol 4, where the two agents are A_k and A_1, respectively. The reader may verify that in both cases we can take $[L, U] = [-C, 2S - 1]$; moreover, in Step 5 we can take $K = 2C$ while in Step 6 we can take $K = C$.

Agents A and B may resolve this millionaire's problem by applying Yao's garbled circuit protocol [28]. However, as that protocol invokes costly sub-protocols for oblivious transfer [4], we suggest here a simpler solution that relies on a non-trusted third party, T. In Protocol 4 that third party can be A_2. (Recall that, as discussed in Section 4.2.2, Protocol 4 is invoked only for $k \geq k_0$.) That third party too must not learn the difference $b - a$. Protocol 5 offers a possible solution. In that protocol, p is a random prime that satisfies $p > U - L + 1$, and h is a secure hash function.

Protocol 5 - Secure inequality verification using a non-trusted third party

Input: A has an integer a and B has an integer b so that $a, b \in [L, U]$ and $b \geq a$ if and only if $b - a \leq K$, where L, U, K are known integer bounds.

Output: The third party T learns whether $b \geq a$, but he should not learn the value of $b - a$.

```
1: $A$ and $B$ select a random generator $g$ of $\mathbb{Z}_p^*$.
2: $A$ sends to $T$ the values $p$ and $g^a$ mod $p$.
3: $B$ selects a random integer $0 \leq s \leq p - 1$ and computes the set $W = \{g^{s+i} : 0 \leq i \leq K\}$.
4: $B$ sends to $T$ the value $g^{s+b}$ mod $p$, and the randomly permuted set $h(W) := \{h(w) : w \in W\}$.
5: $T$ computes $u := g^{s+b-a} \bmod p$ and checks if $h(u) \in h(W)$.
6: If $h(u) \in h(W)$, $T$ outputs to $A$ the answer true (i.e., $b \geq a$ holds); else, he outputs to $A$ the answer false.
```

The idea is to represent the integers a and b by elements in a multiplicative group. To that end, the agents select upfront a multiplicative group $\mathbb{Z}_p^*$ where p is a prime greater than $U - L + 1$. Then, A and B select a random and secret generator g of that group (Step 1). As g is a generator of $\mathbb{Z}_p^*$, it is guaranteed that the mapping $i \mapsto g^i \mod p$ is a one-to-one function from the integer interval $[L, U]$ to $\mathbb{Z}_p^*$, since any $p - 1$ consecutive powers of g must be distinct, and $p - 1 \geq U - L + 1$. Then, A and B send to T the values which are described in Steps 2-4. By our assumptions on a and b, it follows that $b \geq a$ if and only if $s + b - a \in [s, s + K]$. Hence, $b \geq a$ if and only if u, as computed by T in Step 5, is in the set W that B computes in Step 3. However, as T receives only $h(W)$ (and not W), the check that he performs in Step 5 is whether $h(u)$ is found in $h(W)$. If it is, then T infers that $b \geq a$; otherwise, the final conclusion is that $b < a$. (Note that T informs only A and not B about his finding, since that is the requirement in Protocol 4.)

THEOREM 4. *Let ε be a bound on the probability of two randomly selected elements from $\mathbb{Z}_p^*$ to form a collusion for h. Then Protocol 5 is a* **false***-biased Monte Carlo algorithm with error probability which is no larger than $(K + 1)\varepsilon$.*

In modern hash functions (e.g., SHA-3 or RIPEMD-320), the collusion probability is negligible. Moreover, it is possible to make sure that those potential errors of negligible probability might result only in performing unnecessary work, as they might lead to a decision not to prune a search path, but not in missing the optimal solution.

THEOREM 5. *Consider Protocol 4 where: (1) in Step 6 the two agents verify the inequality $r \geq y + 1$, which holds if and only if $x < B$; and (2) in both Steps 5 and 6 the inequality verification is carried out by invoking Protocol 5. Then such a protocol is guaranteed to find the optimal solution.*

PROOF. Since $r \geq y + 1$ holds if and only if $y \geq r$ does not hold, then $r \geq y + 1$ holds if and only if $x < B$. In addition, it is easy to check that the two sides of the inequality, $b := r$ and $a := y + 1$, are confined to the interval $[L, U] = [-C + 1, 2S]$, and if $b \geq a$ then $b - a \in [0, K := C - 1]$.

Assume that Protocol 5 issues a wrong **true** answer to the inequality verification in Step 5 of Protocol 4. That can occur only in Case 1 ($y = \delta + r$) and the result would be that r will be replaced by $r + S$, unnecessarily. But then, in Step 6, r would be clearly greater than y. That inequality will be verified and, as a result, the agents will infer that $x < B$. Consequently, the CPA will be considered as one that could be extended to an optimal solution and, therefore, it will not be discarded. The potential damage of such an erroneous answer of Protocol 5 is only excessive work.

Next, if Protocol 5 issues a wrong **true** answer to the inequality verification in Step 6 of Protocol 4, then the players will wrongly infer that $x < B$. Here too, the potential damage is only that of performing unnecessary work (by not

Table 1: Communication costs of Protocol 2

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

Table 2: Communication costs of Protocol 3

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	$(k^2-4k+4)\ell_S$

discarding a CPA that cannot be extended to an optimal solution). The conclusion is that the combined protocol will never miss the true optimal solution. □

Next, we attend to the security of the protocol. First, we note that the protocol would have been secure against a computationally bounded agent T even if no hashing had been used, since T does not know the generator g nor the random mask s. However, in view of Theorem 6, we chose to strengthen the protocol by using a secure hash function.

THEOREM 6. *If $h = id$ then a computationally* <u>*unbounded*</u> *T will be able to find the value of $b-a$.*

As for s, it was introduced in order to prevent T from identifying the case $b-a=0$. Indeed, if we had selected $s=0$, then whenever $b-a=0$ we would have got $u=1$; such a case could have been identified by T since then $h(u)=h(1)$, regardless of the generator g that was selected.

THEOREM 7. *Protocol 5 is secure against computationally bounded agents.*

The proofs of Theorems 4, 6, and 7 are omitted due to page limitation.

5. EVALUATION

5.1 Communication and computation costs

In this section we describe the communication and computation costs of the secure protocols that we presented in Section 4. Tables 1, 2 and 3 summarize the communication costs of Protocol 2 (when executed with $k=n$), Protocol 3 (as a function of k), and Protocol 4. 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. Here, $\ell_S := \lceil \log S \rceil$ is the number of bits for representing integers in $[0, S-1]$, C is the upper bound on the cost of full assignments, and $\ell_p := \lceil \log p \rceil$ is the number of bits to represent numbers modulo p (recall that Protocol 4 executes Protocol 5 in which the arithmetic is over $\mathbb{Z}_p^*$). Due to lack of space we omit detailed explanations of these communication costs.

As for runtime, Protocols 2, 3, and 4 perform only random number generation and additions, which are low cost operations. The main computational toll is in Steps 5 and 6 of Protocol 4 in which Protocol 5 is invoked. The latter protocol executes the costly operations of modular exponentiations, multiplications, inversions, and hash evaluations. Denoting the runtimes of executing a single operation of those types by C_e, C_m, C_i, and C_h respectively, the runtime of Protocol 4 can be verified to be dominated by $8C_e + (3C+2)C_m + 2C_i + (3C+4)C_h$.

Table 3: Communication costs of Protocol 4

Rounds	# messages	# bits
Step 3	1	$\ell_S + 1$
Step 5 (3)	5	$5\ell_p + (2C+1)\ell_h + 1$
Step 6 (3)	5	$5\ell_p + (C+1)\ell_h + 1$
7	11	$10\ell_p + (3C+2)\ell_h + \ell_S + 3$

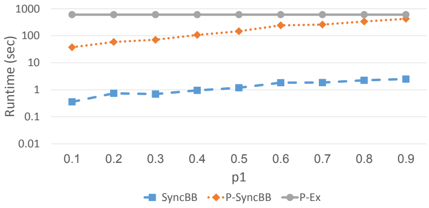

Figure 1: Runtime

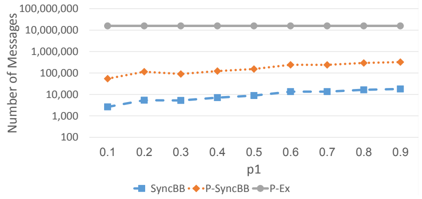

Figure 2: Network load

5.2 Experiments

We evaluated the performance of P-SyncBB by comparing its runtime and network load to two other algorithms: the regular (non-privacy-preserving) SyncBB, and a naïve privacy-preserving exhaustive-search algorithm that we call P-Ex. P-Ex is similar to P-SyncBB except that it does not do pruning. This is achieved by skipping lines 28-34; other than that P-Ex coincides with P-SyncBB, including the usage of Protocol 2 for privacy preservation. The algorithms were implemented in the AgentZero[1] simulator.

In the experiments we used randomly generated DCOPs with $n=8$ agents, $|D_1| = \cdots = |D_n| = 8$, maximal constraint cost $c=10$, and varying constraint density $0.1 \le p_1 \le 0.9$. The runtime of DCOP algorithms is commonly measured by logical operations, such as non-concurrent constraint checks [33]. As P-SyncBB includes arithmetic operations that are not constraint checks, we report non-concurrent runtime. As the algorithms are basically sequential, the non-concurrency of the measure only takes effect during the run of the secure protocols.

Figures 1 and 2 present the mean results (in logarithmic scale) of runtime and network load. While the overhead of the secure protocols is significant, its effect remains almost the same when the problems become denser. The main problem of P-SyncBB is the high computation overhead of Protocol 5 that is invoked for pruning. Nevertheless, the graphs show that even performing highly expensive pruning is still worthwhile, when considering the alternative, P-Ex.

6. CONCLUSION

We presented here P-SyncBB, a privacy-preserving version of the SyncBB algorithm for solving DCOPs while respecting constraint privacy. P-SyncBB preserves the private constraint information by computing the costs of CPAs and comparing them to the current upper bound, using secure multiparty protocols. To this end, we devised Protocol 5, that solves the millionaires' problem securely without resorting to costly oblivious transfer sub-protocols, and then

[1] http://code.google.com/p/azapi-test

used that protocol in Protocol 4, 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.

One research direction that we plan to pursue is improving the computational cost of Protocol 5. Other research directions include the extension of this work to other search-based algorithms for DCOP solving, such as AFB [6] or NCBB [2], and to other privacy types. Finally, we plan to compare our approach to the one that was recently presented in [16].

7. ACKNOWLEDGMENTS

The authors would like to thank Benny Lutati and Vadim Levit for their help with the AgentZero simulator.

8. REFERENCES

[1] J. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Crypto*, pages 251–260, 1986.

[2] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *AAMAS*, pages 1427–1429, 2006.

[3] P. Doshi, T. Matsui, M. C. Silaghi, M. Yokoo, and M. Zanker. Distributed private constraint optimization. In *WI-IAT*, pages 277–281, 2008.

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

[5] B. Faltings, T. Léauté, and A. Petcu. Privacy guarantees through distributed constraint satisfaction. In *WI-IAT*, pages 350–358, 2008.

[6] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding. *JAIR*, 34:25–46, 2009.

[7] A. Gershman, R. Zivan, T. Grinshpoun, A. Grubshtein, and A. Meisels. Measuring distributed constraint optimization algorithms. In *DCR Workshops*, pages 17–24, 2008.

[8] R. Greenstadt, B. Grosz, and M. D. Smith. SSDPOP: improving the privacy of DCOP with secret sharing. In *AAMAS*, pages 171:1–171:3, 2007.

[9] R. Greenstadt, J. Pearce, and M. Tambe. Analysis of privacy loss in distributed constraint optimization. In *AAAI*, pages 647–653, 2006.

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

[11] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer, and A. Meisels. Asymmetric distributed constraint optimization problems. *JAIR*, 47:613–647, 2013.

[12] T. Grinshpoun and A. Meisels. Completeness and performance of the APO algorithm. *JAIR*, 33:223–258, 2008.

[13] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *CP*, pages 222–236, 1997.

[14] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *TKDE*, 16:1026–1037, 2004.

[15] T. Léauté and B. Faltings. Distributed constraint optimization under stochastic uncertainty. In *AAAI*, pages 68–73, 2011.

[16] T. Léauté and B. Faltings. Protecting privacy through distributed computation in multi-agent decision making. *JAIR*, 47:649–695, 2013.

[17] R. T. Maheswaran, J. P. Pearce, E. Bowring, P. Varakantham, and M. Tambe. Privacy loss in distributed constraint reasoning: a quantitative framework for analysis and its applications. *JAAMAS*, 13:27–60, 2006.

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

[19] R. Mailler and V. R. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–445, 2004.

[20] J. Modi and M. Veloso. Multiagent meeting scheduling with rescheduling. In *DCR Workshops*, 2004.

[21] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[22] K. Nissim and R. Zivan. Secure DisCSP protocols – from centralized towards distributed solutions. In *DCR Workshops*, 2005.

[23] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[24] M. C. Silaghi, B. Faltings, and A. Petcu. Secure combinatorial optimization simulating DFS tree-based variable elimination. In *ISAIM*, 2006.

[25] M. C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *WI-IAT*, pages 531–535, 2004.

[26] T. Tassa and E. Gudes. Secure distributed computation of anonymized views of shared databases. *TODS*, 37, Article 11, 2012.

[27] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, pages 639–644, 2002.

[28] A. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

[29] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *JAIR*, 38:85–133, 2010.

[30] M. Yokoo, K.Suzuki, and K. Hirayama. Secure distributed constraints satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, 161:229–246, 2005.

[31] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:55–87, 2005.

[32] S. Zhong, Z. Yang, and R. Wright. Privacy-enhancing *k*-anonymization of customer data. In *PODS*, pages 139–147, 2005.

[33] R. Zivan and A. Meisels. Message delay and DisCSP search algorithms. *AMAI*, 46(4):415–439, 2006.