A suite of secure multi-party computation algorithms for solving distributed constraint satisfaction and optimization problems

Marius C. Silaghi
Florida Institute of Technology

Technical Report CS-2004-04

Abstract

Privacy requirements can be modeled within the distributed constraint satisfaction framework, where the constraints are secrets of the participants. In [20] we introduced a first technique allowing agents to solve distributed constraint problems (DisCSPs), without revealing anything and without trusting each other or some server. The first technique we propose now, MPC-DisCSP2, is a dm times improvement for m variables of domain size d. A second technique we propose, MPC-DisCSP3, has a similar complexity as MPC-DisCSP2, a little bit slower in the worst case, but guarantees that the returned solutions are picked according to a uniform distribution over the total set of solutions. A third technique, MPC-DisCSP0, is based solely on secure arithmetic circuit evaluation and the distribuition for picking its solutions can be manipulated in a more flexible way. Nevertheless this last technique has a O(d^m!d^m) complexity. All techniques of [20] can be extended to solve optimization for distributed weighted CSPs.

1. Introduction

Someone's private concerns can often be formulated in a general framework such as constraint satisfaction problems (CSPs), i.e. where everything is modeled by either variables, or constraints on the possible assignments to those variables. Then, they can be solved with any of the applicable CSP techniques. Often, one has to also find agreements with others for a solution from the set of possible valuations for variables modeling shared resources. The general framework modeling this kind of combinatorial problems is called Distributed Constraint Satisfaction.

In practice one also meets optimization problems. Distributed Weighted CSPs (DisWCSPs) is a general formalism that can model distributed problems with some optimization requirements. Now we introduce the Distributed Weighted Constraint Satisfaction Problem and present shortly the solution proposed in this work.

CSP A *constraint satisfaction problem* (CSP) is defined by three sets: (X, D, C). $X = \{x_1, ..., x_m\}$ is a set of variables and $D = \{D_1, ..., D_m\}$ is a set of domains such that x_i can take values only from $D_i = \{v_1^i, ..., v_{d_i}^i\}$. $C = \{\phi_1, ..., \phi_c\}$ is a set of constraints (predicates) such that ϕ_i is a constraint over an ordered subset $X_i = \{x_{i_1}, ..., x_{i_{k_i}}\}$ of the variables in X, $X_i \subseteq X$. An assignment is a pair $\langle x_i, v_k^i \rangle$ meaning that the variable x_i is assigned the value v_k^i. ϕ_i constrains the legality of each combination of assignments to the variables in X_i.

A tuple is an ordered set. The projection of a tuple ϵ of assignments over a tuple of variables X_i is denoted $\epsilon_{|_{X_i}}$. A solution of a CSP (X,D,C) is a tuple of assignments ϵ with one assignment for each variable in X such that each $\phi_i \in C$ is satisfied by $\epsilon_{|_{X_i}}$.

Constraint Satisfaction Problems (CSPs) do not model optimization requirements. An extension allowing for modeling some optimization concerns is given by Weighted CSPs.

Definition 1 ([13]) *A Weighted CSP (WCSP) is defined by a triplet of sets (X, D, C) and a bound B. X and D are defined as in CSPs. In contrast to CSPs, C=$\{\phi_1, ..., \phi_c\}$ is a set of functions, $\phi_i : D_{i_1} \times ... \times D_{i_{k_i}} \rightarrow [0..b_i]$ where b_i is a maximal value (aka weight) for ϕ_i.*

Its solution is $\underset{\epsilon \in D_1 \times ... \times D_m}{\operatorname{argmin}} \sum_{i=1}^{c} \phi_i(\epsilon_{|_{X_i}})$, if the corresponding sum is smaller than B.

A Distributed CSP (DisCSP) is defined by four sets (A, X, D, C). A=$\{A_1, ..., A_n\}$ is a set of agents. X, D, C and the solution are defined like in CSPs. Each constraint ϕ_i is known only by one agent, being the secret of that agent.

Example 1 *In a problem P, two persons Alice (A_1) and Bob (A_2) want to find a common place (x_1) and time (x_2) for meeting. x_1 is either Shanghai (S) or Halifax (H), i.e. $D_1 = \{S, H\}$. x_2 is either Monday (M) or Thursday (T), i.e. $D_2 = \{M, T\}$. Each of them has a secret constraint on the possible time and place of their meeting. Alice accepts only $\{(H, M), (H, T)\}$ which defines ϕ_1. Bob accepts either of $\{(S, M), (H, M), (H, T)\}$, defined by ϕ_2.*

The problem is to find values for x_1 and x_2 satisfying both ϕ_1 and ϕ_2 and without revealing anything else about ϕ_2 to Alice or about ϕ_1 to Bob.

Definition 2 (DisWCSP) *A Distributed Weighted CSP is defined by four sets (A, X, D, C) and a bound B. A, X, D are defined as for DisCSPs. In contrast to DisCSPs, the elements of C are functions $\phi_i : D_{i_1} \times ... \times D_{i_{k_i}} \rightarrow [0..b_i]$, like for WCSPs.*

Its solution is $\underset{\epsilon \in D_1 \times ... \times D_n}{\operatorname{argmin}} \sum_{i=1}^{c} \phi_i(\epsilon_{|_{X_i}})$, if the corresponding sum is smaller than B.

We assume that agents know the variables involved in the constraints of each other (variables can be falsely declared as involved when this is needed to hide the structure of the problem). Instead, agents want to avoid that others find their *secrets*, namely details about the exact combinations allowed by the constraints that they enforce.

Example 2 *Consider again the problem in Example 8. The users know attached costs to the arrangements that they accept. The bound on the total cost of an accepted solution, B, is \$3000. We can consider the costs in units of \$500, such that we take B=6. Alice accepts only $\{\langle(H,M),\$500\rangle\ \langle(H,T),\$500\rangle\}$ which defines ϕ_1, such that $\phi_1(H,M)$=$\phi_1(H,T)$=1 and $\phi_1(S,M) = \phi_1(S,T)$=6, i.e. B. Bob accepts either of $\{\langle(S,M),\$1000\rangle, \langle(H,M),\$500\rangle, \langle(H,T),\$1000\rangle\}$, defining ϕ_2 similarly. $\phi_2(S,M)$=$\phi_2(H,T)$=2, $\phi_2(H,M)$=1, and $\phi_2(S,T)$=6.*

The problem is to find values for x_1 and x_2 minimizing the sum of ϕ_1 and ϕ_2 such that it is lower than B, and without revealing anything else about ϕ_2 to Alice and anything else about ϕ_1 to Bob.

Definition 3 (Arithmetic Circuit [1]) *An arithmetic circuit is a function f, using solely the operations of some finite field $\mathbb{Z}_p$: $f : \mathbb{Z}_p^n \rightarrow \mathbb{Z}_p$*

Several multi-party techniques are known to compute general functions with secret inputs. These are mainly versions of oblivious evaluation of boolean circuits (boolean operations over $\{0,1\}$), or arithmetic circuit evaluation [1]. But a DisWCSP is not a function. For a given input problem, a DisWCSP can have several solutions or no solution at all.

In [20] we proposed an algorithm for solving DisCSPs, called SecureRandomSolution. We will refer it from now on as MPC-DisCSP1. It uses evaluations of some arithmetic circuits as one of its building blocks. The algorithm described here, MPC-DisCSP2, is based on faster arithmetic circuits than the ones of MPC-DisCSP1. It also allows the n participating agents to securely find a solution by interacting directly without any external arbiters and without divulging any secrets. It is a *threshold scheme*, namely guaranteeing that no subset of t malicious agents that follow the protocol, $t<\lceil n/2 \rceil$, can find anything about others' problems except for what is revealed by the solution.

As it was suggested for MPC-DisCSP1, MPC-DisCSP2 can be extended to perform optimization in Distributed Weighted CSPs. The extension consists in first redesigning one of the basic arithmetic circuits involved in MPC-DisCSP2 such that the algorithm is enabled to find a solution with a predefined weight. Then one can simply find the optimal solution of the DisWCSP by scanning for a solution with weight 0, then weight 1, etc. until the first solution is found. However, this reveals to everybody the quality of the found solution! We propose a new technique called MPC-DisWCSP2 which reveals the quality of the solution only to a set of agents chosen by the participants, or to nobody.

Intuition Consider a constraint in its multidimensional matrix representation, where an element of the matrix restricts the compatibility of some values for distinct variables. Each such element encoded as 0 or 1 is encrypted with a shared key (it can be decrypted only when the majority of the agents agree). One can perform additions and multiplications of such values, while they are encrypted.

The agents cooperate to generate a secret permutation of the encrypted constraints, that cannot be manipulated by any of them.

We give a fix (exponential) set of additions and multiplications that, applied on the constraints encrypted in the aforementioned way, returns the assignments in a solution picked among the possible solutions according to the random permutation.

The agents may show now their share of the key for the assignments in the solution. Each agent learns only the assignments of interest to him.

Problem subtleties The problem is how to formalize the meeting scheduling as an arithmetic circuit! An arithmetic circuit whose outcome is the set of all solutions was designed in [10]. If one tries to use that approach when only one solution is needed, the result returned by the function will reveal to everybody a lot more information than needed. It will tell, for example, that everybody is available and can reach the corresponding places on the days in the alternative solutions. It also suggest that at least one person is busy on each alternative that is not a solution. Some of this information can lead to undesired leaks of privacy. The approach of testing each alternative one by one until a solution is found has similar potential leaks of privacy.

In consequence, one needs to design arithmetic circuits returning only one solution. There is still the problem of which solution should be returned. It is possible to return the first solution in the lexicographical order on the search space [20]. However, knowing that the solution was computed in this way leaks the fact that the alternatives placed before it in the lexicographical order on the search space are rejected by some agent.

Therefore, what we need is a probabilistic arithmetic circuit that returns a solution picked randomly among the possible solutions to the problem. The approaches we proposed in [23, 22], MPC-DisCSP1 and MPC-DisCSP2, generate a secret permutation of domains (and eventually variables) on an encrypted description of the problem. The permuted problem is then input to an arithmetic circuit that computes the first solution in lexicographic order. The solution is then translated with the inverse permutations to the initial problem formulation, before being decrypted. The used permutation guarantees to give each solution a chance to be returned, so that no secret about meeting acceptance/rejection can be inferred from the returned result. If there is no solution, this will intrinsically reveal to everybody that each alternative is constrained by some agent, but this leak is inherent to the problem and not to the algorithm.

The remaining problem is that the permutation we propose in [23] does not guarantee that solutions are picked with a uniform distribution over the possible solutions. Therefore, when an agent uses his constraints in several computations using the same algorithm, some statistical information can be extracted about his secrets, besides his acceptance of the solution. For example, if the returned solutions often specify a meeting in Quebec on Tuesday and rarely some other alternatives, then it can be inferred that some agent can go to Quebec only Tuesday, with higher probability than what statistics ignorant of the used permutation algorithm could infer.

In this paper we analyze this leak and design a scheme, MPC-DisCSP3, where the solutions are picked with a uniform distribution over the possible solutions. Repeated use of the same constraint in different problems will still suggest that a certain meeting is the only one possible, if it is always returned. However, the likelihood of the inference is lower than in the previous techniques and this time it is inherent to the problem and not to the algorithm.

Moreover, it is easy to extend the technique such that alternatives already known to be accepted by an agent are verified first, if it is acceptable to save someone's privacy in the detriment of the others.

Next we present the background techniques and the details of MPC-DisCSP3. We also introduce the architecture of the web application that will service our technique, together with its implications.

2. Overview of MPC-DisCSP1

MPC-DisCSP1 uses general multi-party computation techniques. General multi-party computation techniques can solve securely only certain functions, one of the most general classes of solved problems being the arithmetic circuits over finite fields. A Distributed CSP is not a function. A DisCSP can have several solutions for an input problem, or can even have no solution. Two of the three reformulations of DisCSPs as a function (see [23]) are relevant here:

i A function DisCSP^1() returning the first solution in lexicographic order, respectively an invalid valuation τ when there is no solution.

ii A probabilistic function DisCSP() which picks randomly a solution if it exists, respectively returns τ when there is no solution.[1]

For privacy purposes only the 2^{nd} alternative is satisfactory. DisCSP() only reveals what we usually expect to get from a DisCSP, namely *some* solution. DisCSP^1() intrinsically reveals more [23]. MPC-DisCSP1 implements DisCSP() in three phases:

1. The input DisCSP problem is jointly shuffled by reordering values (and eventually variables) randomly by composing secret permutations from each participant agent.

2. A version of DisCSP^1() where operations performed by agents are independent of the input secrets, is computed by simulating arithmetic circuit evaluation with the technique in [1].

3. The solution returned by the DisCSP^1() at step 2 is translated into the initial problem definition using a transformation that is inverse of the shuffling at step 1.

It is also possible and very simple to find all solutions [10]. However, when only a solution is needed this leaks a lot of information. At step 2, MPC-DisCSP1 requires a version of the DisCSP^1() function whose cost is independent of the input since otherwise the users can learn things like: *The returned solution is the only one, being found after unsuccessfully checking all other valuations, all other valuations being infeasible.* However, the DisCSP^1() used by MPC-DisCSP1 is very complex and we propose a much simpler and faster solution.

Computation of DisCSP() Revealing the first solution ϵ_0 in a lexicographic order reveals two distinct things: ϵ_0 is a solution (or at least that the elements communicated to each participant are part of a solution ϵ_0), and there exists no solution lexicographically ordered before ϵ_0. MPC-DisCSP1 returns a solution picked randomly from the existing solutions by rephrasing the DisCSP with a hidden permutation after its secrets were shared.

Theorem 1 *For any CSP whose search space has size Θ, and for any j, $0 \leq j < \Theta$ there exists a shuffling of the values in its domains such that a solution with any initial lexicographic position i in this search space is mapped into the position j of the obtained problem.*

Proof. This can be proven by constructing the shuffling. First, we find the positions p_k^j and p_k^i of each value in the domain of each variable x_k for the tuples with lexicographic positions j and i. This is done by iteratively computing for k from n to 1, $p_k^j := j\%d_k$, $j = \lfloor j/d_k \rfloor$. Next, the permutation, π_k, for the domain of each variable x_k is chosen such that $\pi_k[p_k^j] = p_k^i$. The shuffling defined by permutations π_k satisfies the requirements and the theorem is proven. □

Corollary 1.1 *For any CSP and a given solution, there exists a shuffling of the values in its domains mapping that solution into the lexicographically first tuple of the obtained problem.*

As follows from the previous corollary, one cannot extract any additional non-statistical information from the identity of the solution of the problem shuffled with unknown permutations of the domains.

[1] Another encounter of randomization with CSPs appears in [5]

function value-to-unary-constraint2(v, M)

1. Jointly, all agents build a vector u, $u = \langle u_0, u_1, ..., u_M \rangle$
   with $4M-2$ multiplications of secrets, computing:
   1. a vector: $\{x_i\}_{0 \leq i \leq M}$, x_0=1, x_{i+1}=$x_i * (v-i)$
   2. a vector: $\{y_i\}_{0 \leq i \leq M}$, y_M=1, y_{i-1}=$y_i * (i-v)$
   then, $u_k = \frac{1}{k!(M-k)!}(v-k+1)x_k y_k$, where $0! \stackrel{\text{def}}{=} 1$.

2. Return u.

Algorithm 1: Transforming secret value $v \in \{0, 1, 2, ..., M\}$ to a shared secret unary constraint.

3. Overview of Secure Arithmetic Circuit Evaluation

Secure evaluation of functions with secret inputs (where sometimes secret functions can also be treated as secret inputs and vice-versa) have been often addressed in literature [1, 28]. Several recent versions are based on (oblivious) boolean circuit evaluation [12, 9]. Others, like MPC-DisCSP1, are based on arithmetic circuit evaluation.

MPC-DisCSP1 uses $(+, \times)$-homomorphic encryption functions E_{K_E}, i.e. respecting:

$$\forall m_1, m_2, E_{K_E}(m_1)E_{K_E}(m_2) = E_{K_E}(m_1 + m_2).$$

Some encryption functions take a randomizing parameter r. However, we write $E_i(m)$ instead of $E_i(m, r)$, to simplify the notation. A good example of a $(+, \times)$-homomorphic scheme with randomizing parameter is the Paillier encryption [16].

To destroy the visibility of the relations between the initial problem formulation and the formulation actually used in computations one can exploit random joint permutations that are not known to any participant. Here we reformulate the initial problem by reordering its parameters. Such permutations appeared in Chaum's mix-nets [3]. The shuffling is obtained by a chain of permutations (each being the secret of a participant) on the encrypted secrets.

[1]'s secure arithmetic circuit evaluation exploits Shamir's secret sharing [19]. This sharing is based on the fact that a polynomial $f(x)$ of degree $t-1$ with unknown parameters can be reconstructed given the evaluation of f in at least t distinct values of x. This can be done using Lagrange interpolation. Instead, absolutely no information is given about the value of $f(0)$ by revealing the valuation of f in any at most $t-1$ non-zero values of x. Therefore, in order to share a secret number s to n participants $A_1, ..., A_n$, one first selects $t-1$ random numbers $a_1, ..., a_{t-1}$ that will define the polynomial $f(x) = s + \sum_{i=1}^{t-1}(a_i x^i)$. A distinct non-zero number k_i is assigned to each participant A_i. The value of the pair $(k_i, f(k_i))$ is sent over a secure channel (e.g. encrypted) to each participant A_i. This is called a (t, n)-threshold scheme.

Once secret numbers are split and shared with a (t, n)-scheme, computations of an arbitrary agreed function of a certain class can be performed over the shared secrets, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, t) [1, 26]. For [19]'s technique, one knows to perform addition and multiplications when $t \leq (n-1)/2$.

4. New Arithmetic Circuits for DisCSP^1()

The main building block of DisCSP^1() consist of evaluating some arithmetic circuits. It is for this step that we are proposing a simpler and faster version. An implementation of DisCSP^1() can be

easily obtained by checking all tuples until one satisfies all the constraints. Such a solution has a number of operations dependent on the secret constraints of the problem. This is why it cannot be used in DisCSP().

Consider the CSP P=(X, D, C). One can interpret the constraints of C as functions with results in the set $\{0, 1\}$ (0 is infeasible and 1 is feasible). The solutions of P are the tuples of assignments ϵ (of type $\langle(x_1, v^1_{\epsilon^1}), ..., (x_m, v^m_{\epsilon^m})\rangle$) with $\prod_{\phi_k \in C} \phi_k(\epsilon_{|X_k}$)=1. The size of the search space (total number of tuples) is $\Theta = \prod_{k=1}^{m} d_k$.

Let us detail now MPC-DisCSP2. If $p(\epsilon) = \prod_{\phi_k \in C} \phi_k(\epsilon_{|X_k}$), and ϵ_k denotes the k^{th} tuple in the lexicographic order, then define:

$$\begin{aligned} h_1(P) &= 1 \\ h_i(P) &= h_{i-1}(P) * (1 - p(\epsilon_{i-1})) \end{aligned}$$

The index of the lexicographically first solution can be computed by accumulating the terms of the h series, weighted as follows:

$$id(P) = \sum_{i=1}^{\Theta} i * p(\epsilon_i) * h_i(P) \quad (1)$$

A result of 0 means that there is no solution. The cost of this computation is $(c+1)d^m$ multiplications of secrets, md times less than the technique in MPC-DisCSP1, which is $O((cm + m^2)d^{m+1})$, where d= $\max_i(d_i)$.

One can then compute the values of the different variables in the found solution. Now we first transform the index id of the solution computed with Equation 1 into a shared vector S, of size Θ where only the id^{th} element is 1 and all other elements are 0. This is achieved using Equation 2. The technique for transforming the solution to a vector, shown in Algorithm 1, has $3M$ multiplications, M less than **value-to-unary-constraint1** [20].

The value of the u^{th} variable in the t^{th} tuple of the search space is $\eta_u(t)$, computed with Equation 3. An arithmetic circuit, $f_i(P)$, (see Equation 4), can now be used to compute the value of each variable x_i in the solution.

$$S\text{=value-to-unary-constraint2}(id, 1{+}\Theta) \quad (2)$$

$$\eta_u(t) = \lfloor (t-1) / \prod_{k=1}^{u-1} d_k \rfloor \mod d_u \quad (3)$$

$$f_i(P) = \sum_{t=1}^{\Theta} (\eta_i(t) + 1) * S[t] \quad (4)$$

It can be noticed that the space required for computing S is $O(d^m)$. This can be reduced by not reusing intermediary results in Algorithm 1 and computing S on demand during the evaluation of f functions, but with efficiency losses that are unacceptable, $O(d^{2m})$. We call this circuit, DisCSP2[1]().

Example 3 *Let us see a full example of how this arithmetic circuit is applied to Example 8.*
$p(S, M)$=0, $p(H, M)$=1, $p(S, T)$=0, $p(H, T)$=1.
$h_1(P)$=1, $h_2(P)$=1, $h_3(P)$=0, $h_4(P)$=0.

The index of the solution is computed with Equation 1, yielding id(P)=2. This is used according to Equation 2 to generate the vector S=$\{0,0,1,0,0\}$.

The vector S is used to compute the values of the variables in the solution, using Equations 3 and 4:
$\eta_1(1)$=0, $\eta_1(2)$=1, $\eta_1(3)$=0, $\eta_1(4)$=1. $\eta_2(1)$=0, $\eta_2(2)$=0, $\eta_2(3)$=1, $\eta_2(4)$=1. $f_1(P)$=2, $f_2(P)$=1.

This signifies that the solution chosen by this arithmetic circuit is x_1=Halifax and x_2=Monday.

5. Computation of DisCSP()

Revealing the first solution, ϵ_0, in a lexicographic order reveals two distinct things: ϵ_0 is a solution (or at least that the elements communicated to each participant are part of a solution ϵ_0), and there exists no solution lexicographically ordered before ϵ_0. To avoid the second leak, MPC-DisCSP1 returns a solution picked randomly from the existing solutions by rephrasing the input DisCSP with a hidden permutation after its secrets were shared.

MPC-DisCSP2's mix-net for reordering shared secret DisCSPs MPC-DisCSP2 shares and shuffles the DisCSP's domains (and eventually variables) in a similar way as MPC-DisCSP1. Informally (see formalization in [21]), each agent chooses a random secret permutation π_k for each domain D_k:

$$\pi_i : \quad [1..d_i] \rightarrow [1..d_i], \qquad\qquad i \in [1..m]$$

The secret shares, of the $\{0,1\}$ value associated by the extensional representation of each constraint ϕ to a tuple, are serialized according to the current lexicographic order on domains.

The serialized constraints are passed along a mix-net [1], as introduced in Section 3, and shuffled according to the permutations $\pi_1, ..., \pi_m$ of each agent.

To avoid that agents get a chance to learn the final permutation by matching final shares with the ones they generated, a randomization step is applied at each shuffling. Each agent applies a randomization step on the set of shares for each secret constraint value, by adding shares of 0, as in [1]. Because of the encryption, this randomization step is based on $(+, \times)$-homomorphic encryption [21].

Decoding the solution After DisCSP[1] is run on the shared problem shuffled as shown by the previous technique, the shares of the results of functions f (processed with Equation 5) have to be revealed without revealing the permutation. Randomization steps are performed as for encoding. Therefore, each agent A_k generates md random sets of shares of zero, $z_k^j[t]$ being A_j's share of the t^{th} zero.

The vectors $\{\langle\{E_j(f_i'^j[t])\}_{t\in[1..d_i]}, j\rangle\}_{i\in[1..m]}$, for each j, are sent backward through the mix-net, where $f_i'^j[t]$ is A_j's share for $f_i'[t]$. When A_k receives $\{\langle\{E_j(f_i'^j[t])\}_{t\in[1..d_i]}, j\rangle\}_{i\in[1..m]}$, it generates and sends to A_{k-1} the vector $\{\langle\pi_i^{-1}(\{E_j(f_i'^j[t])E_j(z_k^j[(i-1)d+t])\}_{t\in[1..d]}),$
$j\rangle\}_{i\in[1..m]}$. A_1 broadcasts them.

$$f_i' = \text{value-to-unary-constraint2}(f_i - 1, d_i) \tag{5}$$

Complexity The total number of messages that have to be sent with MPC-DisCSP2 is $3n$ (for shuffling), $n^2(c+1)d^m$ for Equation 1, $n^2 3d^m$ for Equation 2, $n^2 m3d$ for m Equations 5, and $n^2 + 2n$ for decoding the solution. The total number of messages is $n^2(d^m(c+4) + 3md + 1) + 5n$.

Many of these messages can be sent in parallel. Namely, the number of needed rounds for arithmetic circuit evaluation is given by the depth of the circuit [1]. For our circuits, the depth is $\log_2(c)$ for p, $\log_2(\Theta)$ for id(P), $\log_2(\Theta)$ for value-to-unary-constraint2, and 1 for f. The total number of rounds is therefore O($m\log(d) + \log(c)$). Nevertheless, this parallelism requires $O(cd^m)$ concurrent messages.

Remark 1 (MPC-DisCSP3) *A relevant version is obtained if the shuffling is done on the vector of values $p(\epsilon)$ rather than on constraints. The decoding will then be performed on the vector S with the inverse permutations. All other operations are performed similarly. This version is called MPC-DisCSP3.*

6. Uniform Distribution for Solution Selection

Like all existing secure techniques for solving a DisCSP, the MPC-DisCSP2 algorithm gives a chance to each solution to be returned. The solution can be seen as a random variable over the set of tuples with $p(\epsilon) = 1$. However, we have proven that none of the existing techniques returns solutions according to a uniform distribution.

Theorem 2 *Shuffling variables and domains for a CSP does not guarantee that the first solution in the obtained lexicographic order is selected according to a uniform distribution over the set of all solutions.*

Proof. Take as example the CSP $(X, D{=}\{D_1, D_2, D_3\}, C{=}\{\phi_1\})$, $X{=}\{x_1, x_2, x_3\}$, $D_1{=}\{\{a_1, a_2, a_3\}$, $D_2{=}\{b_1, b_2, b_3\}$, $D_3{=}\{c_1, c_2, c_3\}\}$, $\phi_1 = \{(a_2, b_2, c_2), (a_1, b_2, c_2), (a_1, b_1, c_1)\}$.

Applying random permutation of domains and variables drawn from a uniform distribution over the set of possible distributions:

- the solution (a_2, b_2, c_2) appears 1/3% of the times.
- the solution (a_1, b_2, c_2) appears 1/4% of the times.
- the solution (a_1, b_1, c_1) appears 5/12% of the times.

It can be noticed that the frequency with which the solution is drawn is inverse proportional to the frequency of its values among the other solutions. □

Therefore, if an agent participates with the same constraints in several computations, statistical information can be extracted concerning the occurrence of those values in other solutions of the agent. Namely, an assignment that occurs very often indicates that most solutions use only that assignment for the corresponding variable.

Let us now present a technique called MPC-DisCSP3 that is more complex than MPC-DisCSP2 and that returns solutions according to a uniform distribution.

To ensure that solutions are returned according to a uniform distribution, we notice the following Theorem:

Theorem 3 *Consider the application of the following process to a CSP:*

- *Create a (big) vector S' containing the values $p(\epsilon)$ for all search space tuples ϵ, in lexicographic order.*
- *Shuffle the vector S' according to a permutation π picked with a uniform distribution over the possible permutations, obtaining a vector S.*
- *Pick the first value of S' with $p(\epsilon) = 1$. Choose ϵ as the solution to be returned.*

The tuple returned by the three steps above is chosen according to a uniform distribution over all solutions (tuples having $p(\epsilon) = 1$).

Proof. For any sufficiently large number of applications of described procedure, the possible permutations π applied to S' are drawn a relatively equal number of times, since they have a uniform distribution. Therefore, all obtained permutations of the values of S' will result a relatively equal number of times. By symmetry, each ϵ with $p(\epsilon) = 1$ will be placed an equal number of times before all the other solutions. Therefore, the method defines its outcome as a random variable with uniform distribution over the set of all solutions. □

7. Arithmetic Circuit for computing DisCSP

It is possible to design an arithmetic circuit that has as input a set of coin tosses and selects randomly a solution with a uniform distribution over the set of all solutions. The approach requires $O(\Theta!\Theta)$ messages. Due to its cost, it has just a theoretical relevance.

The arithmetic circuit is constructed as follows:

Choose a prime number p larger than Θ. All arithmetic circuits are evaluated in $\mathbb{Z}_p$, if not otherwise specified.

For each of the $k \in [1..\Theta!]$ possible permutations π_k of all tuples in the search space, evaluate securely an arithmetic circuit that finds the first solution. For example, one can use the circuit of MPC-DisCSP1 or the one of MPC-DisCSP2. The result for each variable i is stored in a vector $F_i[k]$.

Then each agent generates and shares a random number in $\mathbb{Z}_{\Theta!}$. The random numbers of all agents are added securely, mod $\Theta!$. The obtained secret r is transformed to a vector with R=value-to-unary-constraint2(r+1,d^m). Each element $R[k]$ of the vector R is multiplied to each value $F_i[k]$ for eack i.

Then, the values for each variable of all the $\Theta!$ solutions are summed with each other: $f_i = \sum_k F_i[k]$. The results of f_i are revealed to the agents owning x_i.

Remark 2 *If uniform distribution is not required, one can create the previous circuit only for all permutations of domains and eventually of variables. The complexity decreases accordingly.*

8. MPC-DisCSP3

Now let us present MPC-DisWCSP3, a multiparty computation simulating securely the method of the Theorem 3. First we redefine the DisCSP framework.

Definition 4 *A Distributed CSP (DisCSP) is defined by four sets (A, X, D, C). A=$\{A_1, ..., A_n\}$ is a set of agents. X, D, C and the solution are defined like in CSPs. Each constraint ϕ_i is known only by one agent, being the secret of that agent. There may exist a public constraint in C, ϕ_0, known to everybody. Each variable x_i is owned by a set of agents A^i, that are entitled to learn its assignment in the solution.*

The public constraint is new for this DisCSP formalism, but is anchored in known arguments [17].

Example 4 *In another view of the problem P, two persons Alice (A_1) and Bob (A_2) want to find a common place (x_1) and time (x_2) for meeting. x_1 is either Paris (P) or Quebec (Q), i.e. $D_1 = \{P, Q\}$. x_2 is either Tuesday (T) or Wednesday (W), i.e. $D_2 = \{T, W\}$. Each of them has a secret constraint on the possible time and place of their meeting. Alice accepts only $\{(P,T), (P,W), (Q,W)\}$ which defines ϕ_1. Bob accepts either of $\{(P,T), (Q,T), (Q,W)\}$, defined by ϕ_2. There is also a publicly known constraint, ϕ_0, which due to an announced strike forbids a meeting in Paris on Wednesday, $\phi_0 = \{(P,T), (Q,T), (Q,W)\}$.*

The problem is to publish values for x_1 and x_2 satisfying all constraints and without revealing anything else about ϕ_2 to Alice or about ϕ_1 to Bob.

MPC-DisCSP3 is shown in Algorithm 2.

Unlike MPC-DisCSP2, this technique does not start by shuffling the shares of the encoded constraints. Rather, it starts by simply sharing the encoded constraints with the Shamir secret sharing scheme. Then, the vector S' of size $\Gamma = \prod_{k=1}^{m} d_k$ is computed by evaluating the arithmetic circuits

Proc MPC-DisCSP3

1. Each agent shares the encoded secret constraints for each tuple.
2. Compute $p(\epsilon)$ for each ϵ that is compatible with ϕ_0, and place the results in a shared secret vector S', ordered lexicographically.
3. Each agent applies on its shares a permutation π that moves the tuples rejected by ϕ_0 to the end.
4. Each agent A_i encrypts shares in its vector S' with his own public key, and submits the resulting vector, without end tuples incompatible to ϕ_0, to the mix-net.
5. The mix-net shuffles the vectors of shares, S', randomizing the shares at each permutation by adding shares of 0 exploiting homomorphic encryption.
6. The shuffled S' is broadcasted by the mix-net to agents.
   - Compute the index of the first solution in S, $id(P)$, with Equation 1.
   - Compute now a vector S with the Equation 2
7. The mix-net decodes the vectors of shares, S, randomizing the shares at each inverse permutation by adding shares of 0 exploiting homomorphic encryption.
8. The shuffled S is broadcasted by the mix-net to agents.
9. Apply the inverse of the permutation π.
10. Compute the assignments of the solution with Equation 4.
11. Reveal each assignment to the interested agents.

Algorithm 2: MPC-DisCSP3

$p(\epsilon) = \prod_{\phi_k \in C} \phi_k(\epsilon_{|_{X_k}})$ for each ϵ compatible with ϕ_0. Each agent applies on its shares a permutation π:

$$\pi : [1..\Gamma] \rightarrow [1..\Gamma].$$

that moves the tuples rejected by ϕ_0 to the end. The number of tuples that are not rejected by ϕ_0 is denoted by Θ.

Only now is the problem shuffled and the shares randomized with a mix-net. After computing $id(P)$ with Equation 1, the vector S is computed with Equation 2. The vector S is now decoded by the mix-net with the inverse permutations, and randomizing the shares again (with the same share randomization technique like in MPC-DisCSP1/MPC-DisCSP2).

π^{-1} is applied to S. Any index after the end of S, is considered by π^{-1} to contain the value 0.

$$\eta_u(t) = \lfloor (t-1) / \prod_{k=1}^{u-1} d_k \rfloor \mod d_u \quad (6)$$

$$f_i(P) = \sum_{t=1}^{\Theta} (\eta_i(t)+1) * S'[t-1] \quad (7)$$

In the end, the values in the solution are computed with the arithmetic circuits in Equations 7. Each assignment in the solution, defined by the results to the functions f_i, is then revealed to the corresponding agents.

Example 5 *Let us see an example of how MPC-DisCSP3 is applied to the Example 8.*
$p(P,T)$=1, $p(Q,T)$=0, $p(P,W)$ *not computed,* $p(Q,W)$=1.
S'=(1,0,_,1)
After applying $\pi = (0,1,4,3)$.
S'=(1,0,1,_)
Shuffle (1,0,1), assume it remains unchanged
$h_1(P)$=1, $h_2(P)$=0, $h_3(P)$=0.
The index of the solution is computed with Equation 1, yielding id(P)=1. This is used according to Equation 2 to generate the vector S=$\{$*0,1,0,0*$\}$.
Unshuffle S[1-3]=(1,0,0)
Apply $\pi^{-1} = (0,1,4,3)$
S=(1,0,0,_)
The vector S *is used to compute the values of the variables in the solution, using Equations 3 and 4:*
$\eta_1(1)$=0, $\eta_1(2)$=1, $\eta_1(3)$=0, $\eta_1(4)$=1. $\eta_2(1)$=0, $\eta_2(2)$=0, $\eta_2(3)$=1, $\eta_2(4)$=1. $f_1(P)$=1, $f_2(P)$=1.
This signifies that the solution chosen by this arithmetic circuit is x_1=*Paris and* x_2=*Tuesday.*

Analysis When compared to classical agent approaches to solving distributed meeting scheduling and CSPs [15, 8, 2, 20, 7, 6, 4, 18, 27, 23, 14, 11, 30, 25], the advantages and drawbacks of MPC-DisCSP3 are the ones defined by t-privacy [1] (i.e. no collusion of less than t participants can learn anything, but the final solution with its quality, and what can be inferred from it), and highlighted in [24]. Namely, MPC-DisCSP3 should never be used when an agent does not trust a majority of the agents, or when it suspects that two thirds of the agents may want to misuse the protocol. It is potentially slower, but more secure in the other cases.

Compared to the techniques based on trusted servers used in [29], the advantages and drawbacks of MPC-DisCSP3 depend on the amount of trust in those servers.

The main advantage of MPC-DisCSP3 over its previous alternatives MPC-DisCSP1 and MPC-DisCSP2, is that it offers the solutions picked according to a uniform distribution over the total set of solutions, as guaranteed by the Theorem 3.

From the space requirements point of view, it has the same exponential complexity as MPC-DisCSP2, namely O(cd^m), since it uses the same data structures (having to store and manipulate the whole vector S). The only additional structures, namely the permutations π and π_i have the same size as S, and do not change the complexity. From this point of view MPC-DisCSP3 is clearly inferior to MPC-DisCSP1 which has polynomial space requirements.

In terms of time complexity, its worst performance (namely when Γ=Θ) is worse than the one of MPC-DisCSP2. This is due to the fact that the messages for shuffling are much larger and the shuffling involves more computations, given by the size of S'. Compared to MPC-DisCSP1, which is $O(dm)$ times slower than MPC-DisCSP2, MPC-DisCSP3 will be faster. This is because MPC-DisCSP2 require passing more than just the vector S.

Remark 3 *When* $\Gamma >> \Theta$*, MPC-DisCSP3 can be much faster then most existing algorithms which do not exploit public constraints. In our knowledge, typically public constraints are not exploited.*

9. MPC-DisWCSP2

The weak extension to distributed weighted CSPs Let $q(\epsilon)=\sum_{\phi\in C}\phi(\epsilon)$. A solution of a CSP (X,D,C) is a valuation ϵ with $q(\epsilon)=|C|$. For addressing Distributed WCSPs, the function p has to be further adapted as follows. Now the maximum value of $q(\epsilon)$ is no longer $|C|$ but $b=\sum_{i=1}^{|C|}b_i$. We still want to isolate solutions ϵ whose $q(\epsilon)$ is some value, x_0 (actually we now need the minimal such x_0 allowing for a solution). We need to redefine $p(\epsilon)$:

$$p(\epsilon)=\frac{\prod_{i=0}^{x_0-1}(q(\epsilon)-i)\prod_{i=x_0+1}^{b}(i-q(\epsilon))}{x_0!(b-x_0)!}. \tag{8}$$

A solution with the lowest weight of a DisWCSP can be found by iterating MPC-DisCSP2 with the Definition 8 for p, for x_0 increasing from 0 to B. Note that the last technique reveals to everybody the quality of the solution, i.e. the sum of constraint weights in the solution.

Theorem 4 *With a $(\lceil n/2\rceil, n)$ threshold scheme, no coalition of less than $\lceil n/2\rceil$ agents can learn any secret of another agent*

Proof. Each step of the previous technique is based on the evaluation of a set of arithmetic circuits with threshold schemes where $t=\lceil n/2\rceil$. It has been proven in [1] that secure arithmetic circuit evaluation is $(t-1)$-private (i.e. No collusion of less than t participants can learn anything, but the final solution with its quality, and what can be inferred from it).

The mix-net is hiding the permutation of domains (the order under which the solution was found). We also proved in Theorem 1 of [23] that random permutations of domains gives each solution a chance to be returned. Therefore there is no case where a collusion of less than t agents can prove acceptance or rejection of another tuple by some agent, by studying the solution.

The information revealed to everybody is the quality of the solution and the allocation of their resources in that solution. The quality of the solution is revealed by the number of computation rounds. □

MPC-DisWCSP2: Solving a DisWCSP while hiding the weight of the solution. DisCSP() hides the number of rounds needed to find the first solution to a DisCSP. Now we must hide the number of rounds needed to find the first solution to a DisWCSP. A new set of functions w_i is defined to hold the value of x_i in the best solution found so far.

$$w_i^j \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } j=-1 \\ f_i(P) & \text{if } w_1^{j-1}=0 \\ w_i^{j-1} & \text{if } w_1^{j-1}\neq 0 \end{cases}$$

This can be computed with the following arithmetic circuits (for $i\in[1..d]$ and $j\in[1..(B-1)]$):

$$\begin{aligned} w_i^{-1} &= 0 \\ w_i^j &= w_i^{j-1}\left(1-\frac{\prod_{k=1}^{d}(k-w_1^{j-1})}{d!}\right)+f_i(P)\frac{\prod_{k\in[1..d]}(k-w_1^{j-1})}{d!} \end{aligned}$$

MPC-DisWCSP2 also consists in three phases:

1. First the DisWCSP is shared and then shuffled through the mix-net in the same way as it was done with the DisCSP (except that the values assigned by the constraint ϕ_k to tuples are in $[0..b_k]$ rather than $\{0,1\}$).

2. The vector $\{w_i^B\}_{i\in 1..m}$ is computed by iteratively building the vectors $\{w_i^j\}_{i\in 1..m}$ for j increasing from 0 to $B-1$. The computation in each iteration j is performed according to the arithmetic circuit DisCSP2[1] but with the new definition of p and with $x_0{=}j$. It is followed by a secure evaluation of the arithmetic circuits $\{w_i^j\}_{i\in 1..m}$.

3. The solution is decoded and distributed as in MPC-DisCSP2, except that the solution vectors are the ones containing the results of the functions w_i^{B-1} rather than f_i.

The complexity of MPC-DisWCSP2 is B times higher than the complexity of MPC-DisCSP2. For the most parallel version of MPC-DisCSP2, the number of rounds increases with $B \log d$.

In MPC-DisWCSP2 nobody can learn the total weight of the solution. In some problems one may nevertheless want to let some particular agents learn the total weight of the solution, while the rest of the agents should not learn it. This can be achieved by computing at the end of the second phase the first element of the solution vector according to:

$$w_0 \quad = \quad \sum_{k\in[1..(B-1)]} k(1 - \frac{\prod_{k_1\in[1..d]}(k_1 - w_1^k)}{d!})\frac{\prod_{k_2\in[1..d]}(k_2 - w_1^{k-1})}{d!}$$

The single non-zero term in the summation defining w_0 is for the round k where w_1^k is for the first time non-zero. w_0 specifies the weight of the solution and after decoding, is revealed only to the agents that should learn it.

Example 6 *Let us see a full example of how this arithmetic circuit is applied to Example 1, (assuming the secret shuffling does not change any order).*
w_1^{-1}=0, w_2^{-1}=0, $\eta_1(1)$=0, $\eta_1(2)$=1, $\eta_1(3)$=0, $\eta_1(4)$=1. $\eta_2(1)$=0, $\eta_2(2)$=0, $\eta_2(3)$=1, $\eta_2(4)$=1.
$x_0 = 0$
p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=0.
$h_1(P)$=1, $h_2(P)$=1, $h_3(P)$=1, $h_4(P)$=1.
S={0,0,0,0,0}, id(P)=0.
$f_1(P)$=0, $f_2(P)$=0. $w_1^0 = 0$, $w_2^0 = 0$.
$x_0 = 1$
p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=0. ...
$f_1(P)$=0, $f_2(P)$=0. $w_1^1 = 0$, $w_2^1 = 0$.
$x_0 = 2$
p(S,M)=0, p(H,M)=1, p(S,T)=0, p(H,T)=0.
$h_1(P)$=1, $h_2(P)$=1, $h_3(P)$=0, $h_4(P)$=0.
S={0,0,1,0,0}, id(P)=2.
$f_1(P)$=2, $f_2(P)$=1. $w_1^2 = 2$, $w_2^2 = 1$.
$x_0 = 3$
p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=1.
S={0,0,0,0,1}, id(P)=4. ...
$f_1(P)$=2, $f_2(P)$=2. $w_1^3 = 2$, $w_2^3 = 1$.
$x_0 = 4$
p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=0. ...
$w_1^4 = 2$, $w_2^4 = 1$.
...
The iteration ends computing for x_0 equal to the maximum admissible cost for P (which according to Definition 2 is B-1). $w_1^5 = 2$, $w_3^5 = 1$.

$w_0 = 2$
This signifies that the solution chosen by this arithmetic circuit is x_1=Halifax and x_2=Monday.

10. Conclusions

Distributed CSPs are a very active research area and secrecy within DisCSPs has been stressed often as an important issue [15, 8, 2, 20, 7, 6, 4, 18, 29, 23, 14, 11, 30, 25]. The technique described here uses a $(\lceil n/2 \rceil, n)$ threshold scheme. It is also possible to develop techniques that may provide robustness to agents not following the protocol, as in other multiparty computations (notably see ($\lceil n/3 \rceil$,n) threshold schemes) [1]. However, is some agent does not trust a majority of the participants than the techniques proposed here are not appropriate, and one should use some other existing approach.

We presented a technique where agents that need to cooperate and whose problems can be modeled as CSPs can find a random solution without leaks of additional information about their constraints. The technique is exponential in space such that only problems with bounded search space size can be addressed (e.g. for computers with up to 256MB, the storage is sufficient for solving meeting scheduling problems with up to 4k alternative places and dates, 100 participants, in approximatively 300 ($=2m \log_2(d) + \log_2(c) + 2n$) rounds of parallel message exchages of less than 1G (cd^m) concurrent messages per round).

We also show how the technique can be extended to perform optimization in Distributed Weighted CSPs. In particular we find a way to avoid the privacy leaks concerning the solution quality, as exhibited by the obvious approach.

References

[1] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *STOC*, pages 1–10, 1988.

[2] C. Bessière, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In *Proc. IJCAI DCR Workshop*, pages 9–16, 2001.

[3] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[4] S. E. Conry, K. Kuwabara, and V. R. Lesser. Multistage negotiation for distributed constraint satisfaction. *IEEE Trans. on systems, man, and cybernetics*, 21(6):1462–1477, 1991.

[5] R. Dechter, K. Kask, E. Bin, and R. Emek. Generating random solutions for constraint satisfaction problems. In *Eighteenth national conference on Artificial Intelligence*, pages 15 – 21, Edmonton, Alberta, Canada, 2002.

[6] J. Denzinger. Tutorial on distributed knowledge based search. IJCAI-01, August 2001.

[7] B. Faltings. Incentive compatible open constraint optimization. In *Electronic Commerce*, 2003.

[8] E. Freuder, M. Minca, and R. Wallace. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *Proc. IJCAI DCR*, pages 63–72, 2001.

[9] O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge, 2004.

[10] T. Herlea, J. Claessens, G. Neven, F. Piessens, B. Preneel, and B. Decker. On securely scheduling a meeting. In *Proc. of IFIP SEC*, pages 183–198, 2001.

[11] X. Jin and J. Liu. Efficiency of emergent constraint satisfaction in small-world and random agent networks. In *IAT*, 2003.

[12] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. of ACM Symposium on Theory of Computing*, pages 20–31, 1988.

[13] J. Larrosa. Node and arc consistency in weighted csp. In *AAAI-2002*, Edmonton, 2002.

[14] J. Liu, H. Jing, and Y. Tang. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136:101–144, 2002.

[15] P. Meseguer and M. Jiménez. Distributed forward checking. In *DCS*, 2000.

[16] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, volume 1592, pages 223–238, 1999.
[17] T. Sandholm and S. Suri. Side constraints and non-price attributes in markets. In *Proc. of IJCAI DCR Workshop*, pages 55–61, Seattle, August 2001.
[18] S. Sen and E. Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 7:265–289, 1998.
[19] A. Shamir. How to share a secret. *Comm. of the ACM*, 22:612–613, 1979.
[20] M. Silaghi. Arithmetic circuit for the first solution of distributed CSPs with cryptographic multi-party computations. In *IAT*, Halifax, 2003.
[21] M. Silaghi. Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. In *IJCAI-DCR*, 2003.
[22] M. Silaghi. A suite of secure multi-party computation algorithms for solving distributed csps. Technical Report CS-2004-05, FIT, April 2004.
[23] M. Silaghi and V. Rajeshirke. A cryptographic multiparty-computation for distributed constraint satisfaction problems ensuring secrecy of constraints by shuffling domains. In *AAMAS*, 2004.
[24] M. C. Silaghi and B. Faltings. A comparison of DisCSP algorithms with respect to privacy. In *AAMAS-DCR*, pages 147–155, 2002.
[25] R. Wallace and M. Silaghi. Using privacy loss to guide decisions in distributed CSP search. In *FLAIRS'04*, 2004.
[26] A. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.
[27] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE TKDE*, 10(5):673–685, 1998.
[28] M. Yokoo and K. Suzuki. Generalized Vickrey Auctions without Third-Party Servers. In *FC04*, 2004.
[29] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.
[30] H. Zhou and B. Choueiry. Characterizing the bahavior of a multi-agent search by using it to solve a tight, real-world resource allocation problem. In *CP-03W: Immediate Applications of CP*, pages 79–99, 2003.