

Asymmetric Distributed Constraint Optimization

Alon Grubshtein¹, Tal Grinshpoun¹, Amnon Meisels¹, and Roie Zivan²

¹ Dept. of Computer Science
Ben Gurion University of the Negev
Beer-Sheva, 84105, Israel
{alongrub, grinshpo, am}@cs.bgu.ac.il

² Carnegie Mellon University
Robotics Institute
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
zivanr@cs.cmu.edu

Abstract. The standard model of distributed constraints optimization problems (DCOPs), assumes that the cost of a constraint is checked by one of the agents involved in the constraint. For DCOPs this is equivalent to the assumption that each constraint has a global cost which applies to each of the participating agents and in other words that all constraints are symmetric. Many multi agent system (MAS) problems involve *asymmetric constraints*. For example, the gain from a scheduled meeting of two agents is naturally different for each of the participants. In order to solve Asymmetric DCOPs (ADCOPs), one needs to design algorithms in which all agents participating in a constraint independently check the gain for each of them. This naturally brings up the question of privacy, enabling agents to keep their cost (or gain) of constraints private, at least partially.

The present paper presents search algorithms for ADCOPs which handle asymmetric constraints in a privacy preserving manner. New versions of Asynchronous Forward Bounding and of Synchronous Branch & Bound are proposed. In addition, two local search algorithms are presented in which agents negotiate moves prior to assigning values. All algorithms are empirically evaluated, and their performance in terms of run-time, network load and solution quality is presented.

Keywords: Asymmetric DCOPs, Distributed optimization

1 Introduction

Distributed constraint optimization problems (DCOPs) are composed of agents, each holding its local constraint network. These are connected by constraints among variables to other agents. Agents assign values to their variables, attempting to generate a globally optimal assignment of all agents [1–3]. To achieve this goal, agents check the value assignments of their variables for the best local cost and exchange messages with other agents, to check the optimality of their proposed assignments against constraints with variables owned by different agents [1, 2].

DCOP is an elegant model for many every day combinatorial problems that are distributed by nature. Take for example the meeting scheduling problem in which n

agents attempt to schedule k meetings. In each meeting, a sub-group of the n agents participate [4–6]. Arrival constraints define the time that must differentiate meetings with common participants. In addition, agents have their own preferences for different time-slots [6, 7].

Standard search algorithms for DCOPs, like Asynchronous Distributed constraints Optimization (ADOPT) [1], assume a static priority order among all agents. Higher priority agents perform assignments and send them via messages to lower priority agents. ADOPT, like all other DCOP search algorithms, assumes that the cost of every inter-agent constraint is evaluated by the lower priority agent that is involved in the constraint. In other words, the lower priority agent holds the entire constraint [1].

In many real world problems the above assumptions are too strong. In the meeting scheduling example, people associate different gains with a scheduled meeting among them. Many times people prefer to keep these costs or gains private and not reveal more than is necessary for scheduling the meeting. Furthermore, in such a distributed setting, evaluating the value of some constraint by a lower priority agent requires the propagation of an arbitrarily large preference function between the different agents. A more suitable model for a realistic DCOP has *Asymmetric Constraints*, where the cost or gain of each inter-agent constraint is composed of two parts, each held by one of the two constraining agents. When agents hold parts of the constraint privately, checking for the cost has to be performed by both of the constrained agents because the cost is the sum of the costs of all agents.

Consider the following example: two smart agents, which are used to handle daily travel plans are considered. Each agent corresponds to either Alice or Bob, and are denoted by A or B respectively. In this example, there are two possible routes to get from Alice and Bob’s neighborhood to their work area. The highway, P_α , is a fast lane. Traveling through this route costs money, and this toll is calculated per vehicle passing through the route. On the other hand, P_β is the slower, free of cost, route. Besides the monetary costs, the agents have other preferences which are represented by non monetary costs. These include the travel time, convenience, etc. The cost function is part of the agent’s private information, and its value naturally depends on actions taken by the other agent. For simplicity, we use an identical cost function for both A and B . This cost is a function of the load on each route and is given by

$$C_\alpha(x) = \frac{3}{x^2} \quad \text{and} \quad C_\beta(x) = \frac{x^2}{2}$$

as depicted in Figure 1a. The variable x denotes the number of agents that have selected the route, which in our example is either 1 or 2.

Naturally each agent seeks to minimize its own cost. However, the designer of the system, is interested in a different property of the solution. She would like to see to the satisfaction of all participants, and find a solution which minimizes the total cost.

An optimal solution can easily be found by any complete DCOP algorithm. However, to do so, one must first aggregate all costs as depicted in Figure 1b. This is required in the DCOP framework so each participant may evaluate any possible state. Such an aggregation has two major flaws:

1. It requires the transmission of an exponential amount of data (exponential in the arity of the constraint).

<i>Alice</i>	<i>Bob</i>	P_α	P_β
P_α		$\frac{3}{4}, \frac{3}{4}$	$3, \frac{1}{2}$
P_β		$\frac{1}{2}, 3$	$2, 2$

(a)

<i>Alice</i>	<i>Bob</i>	P_α	P_β
P_α		$\frac{3}{2}$	$3\frac{1}{2}$
P_β		$3\frac{1}{2}$	4

(b)

Fig. 1: The value of the single constraint in the Travel Planners’ problem for each participant (a), and the aggregated sum used by DCOP algorithms (b)

2. It reveals all private information a priori. If the state of the system is $\langle P_\alpha, P_\alpha \rangle$, Alice’s agent does not need to send its personal cost to Bob’s agent for the evaluation of $\langle P_\alpha, P_\beta \rangle$ since Alice can already tell that it results in a lower quality state.

As the volume of work on applications derived from Game Theory increases, it turns out that besides the possibly improved privacy and reduction in network load, such an asymmetric view of a problem also provides a richer framework for describing strategic Multi Agent Systems. Take for example normal form games (cf. [8]). In many of these games, the payoff of agents participating in the underlying MAS is dependent upon the assignments of other agents as demonstrated in our previous example. In such cases it is often difficult to describe the personal payoffs of participants by a standard DCOP. Adding unary constraints to represent different preferences of the underlying agents, may not be applicable in all cases. In fact, there are strategic situations which do not have a DCOP representation at all.³

Finding the joint optimal solution in such cases is often crucial for many multi-agent system (MAS) problems, but it is also important in Game Theory. There, the efficiency of equilibria is often measured as a ratio between the equilibrium value and the joint optimal value [8, 10].

The implications of asymmetric constraints for the design of DCOP algorithms are demonstrated in the following example: consider the well known Synchronous Branch and Bound algorithm (SyncBB) [11] and a given order of agents. The partial assignment is generated sequentially on a special message - the Current Partial Assignment (CPA) (cf. [12]). Each agent assigns its variables on the CPA when it receives it and assigns it so that the overall cost of the partial assignment is minimal. Running a branch and bound algorithm implies that whenever the cost of a partial assignment becomes larger than the upper bound the agent which holds the CPA backtracks to the former agent. The main point of this description is that for *Asymmetric* DCOPs the above standard process is not correct. To compute the full cost of each partial assignment each constraint must be evaluated by all the agents which participate in it. In other words, once an agent i assigns its variables and computes the cost of the CPA from its point of view it needs to send the CPA back to agents with higher priority (e.g., ordered *before it*, or agents

³ The proof is quite simple: according to [9] a DCOP admits at least one stable point corresponding to a pure strategy Nash Equilibrium (NE). Falsely assume that any strategic situation can be described by a DCOP and use a normal form game which lacks a pure strategy NE, such as “matching pennies” (cf. [8]), to serve as a counter example.

$j < i$) whose assignments are already on the CPA, so that they add the cost of their constraints to the overall cost of the partial assignment.

Similarly to the DCSP case [13] considering both sides of the constraints can be generally performed in two ways. The first strategy is to solve the problem in two phases. In the first phase, a full assignment is reached while considering only one side of the constraints – this phase can be performed using a symmetric DCOP algorithm. In the second phase, the full cost of the assignment is verified by checking the second side of all the involved constraints. After the second phase is complete, the first phase is resumed in a search for a better solution until the entire search space is covered. The second strategy is to systematically check both sides of the constraints before reaching a full assignment, and is thus considered a one-phase strategy. We refer to checking the second side of the constraints as *back-checking*. The back-checking can be performed either synchronously or asynchronously.

Local search techniques can also be modified to handle ADCOPs. These modifications include partial sharing of constraint information via some form of coordination. Although not required, existing algorithms are expected to generally produce better results when a coordination mechanism exists.

The present paper presents several asymmetric versions for DCOP algorithms, both complete and stochastic. Two asymmetric versions for the synchronous branch & bound algorithm (SyncBB), the first version follows the two-phase strategy, while the other versions pursue the one-phase strategy including an asynchronous back-checking algorithm. For the more advanced DCOP algorithms, an asymmetric version for the Asynchronous Forward Bounding algorithm (AFB) [2] is presented. We also present two new local search algorithms which attempt to coordinate moves, allowing changes to take place only when such changes incur a positive gain on the surrounding. This is in contrast to standard (symmetric) DSA [14], which only utilizes the agent’s local problem structure and assignment information of neighbors and can also be used to solve ADCOPs with no alteration.

The evaluation of the proposed algorithms for ADCOPs is performed on randomly generated Max-DCSPs. This is a commonly used benchmark for DCOPs [1, 3, 2]. Performance is evaluated by measuring the computational effort and the cost of communication. All evaluations follow common practice on DCOPs [15].

2 Asymmetric Distributed Constraint Optimization Problems

An *ADCOP* is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. \mathcal{A} is a finite set of agents A_1, A_2, \dots, A_n . \mathcal{X} is a finite set of variables X_1, X_2, \dots, X_m . Each variable is held by a single agent (an agent may hold more than one variable). \mathcal{D} is a set of domains D_1, D_2, \dots, D_m . Each domain D_i contains the finite set of values which can be assigned to variable X_i . \mathcal{R} is a set of relations (constraints). Each constraint $C \in \mathcal{R}$ defines a non-negative *cost* for **every participant** in every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times D_{i_2} \times \dots \times D_{i_k} \rightarrow \prod_{j=1}^k \mathbb{R}^+ \cup \{0\}$. Similar to DCOPs, a *binary constraint* refers to exactly two variables and is of the form $C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\} \times \mathbb{R}^+ \cup \{0\}$, and a *binary ADCOP* is an ADCOP in which all constraints are binary. An *assignment* (or a label) is a pair including a variable and a value from that variable’s domain. A

partial assignment (PA) is a set of assignments in which each variable appears at most once. The *cost of a partial assignment* PA is the sum of all applicable constraints to PA, each taken over all variables, with respect to the assignments in PA. A *complete assignment* is a partial assignment that includes value assignments to all variables. A *solution* is a full assignment of minimal cost.

Following common practice, this paper assumes that each agent owns a single variable, and hence uses the term “agent” and “variable” interchangeably. Constraints are assumed to be at most binary. Agents are aware only of their own topology (i.e., only of their neighbors in the constraints network and the constraints that they personally and privately hold).

3 Asymmetric DCOP algorithms

Asymmetric DCOPs require that the solving process will always take into account both sides of a constraint. We next present three asymmetric versions for the Synchronous Branch & Bound algorithm (SyncBB). The first version, SyncABB-2ph, follows the two-phase strategy. The other versions follow the one-phase strategy. We also present the Asynchronous Two-Way Bounding algorithm (ATWB), which is an asymmetric version for the AFB algorithm. ATWB also follows the one-phase strategy and naturally performs asynchronous back-checking.

3.1 Synchronous Asymmetric B&B – 2-phase

The SyncABB-2ph algorithm is a combination of the SyncBB algorithm with the two-phase strategy. In phase I the algorithm works exactly as SyncBB, where each agent counts the costs of its constraints with lower-indexed agents. Phase I is finished when a full assignment is reached. In symmetric SyncBB this means that a new best solution and a new bound has been found. However, for ADCOPs back-checking is needed in order to verify that the other side of the constraints does not breach the bound.

Algorithm 1 SyncABB-2ph – phase II

```

when received (CPA_BACK_MSG, CPA, cost)
1:  $f \leftarrow$  cost of constraints with upper-indexed agents ( $A_{i+1}..A_n$ )
2: if  $cost + f \geq B$  then
3:   send (CPA_MSG, CPA) to  $A_n$ 
4: else if  $A_i \neq A_1$  then
5:   send (CPA_BACK_MSG, CPA,  $cost + f$ ) to  $A_{i-1}$ 
6: else
7:    $B \leftarrow cost + f$ 
8:   broadcast (NEW_SOLUTION, CPA, B)
9:   send (CPA_MSG, CPA) to  $A_n$ 

```

In phase II the last agent (A_n) sends to its preceding agent (A_{n-1}) a message we label with CPA_BACK_MSG. This message includes the *Current Partial Assignment*

(*CPA*) and its one-side cost that was gathered in phase I. Each agent that receives the **CPA_BACK_MSG** message (Algorithm 1) performs back-checking by computing the cost f of its broken constraints with upper-indexed agents (line 1). In case the addition of f reaches the bound B (line 2), there is no point to continue the back-checking since a new value has to be assigned. Notice that since we are in phase II there is no knowing which agent's assignment actually breached the bound. Thus, to ensure the completeness of the algorithm, the *CPA* must be returned to the last agent (line 3). If the bound has not been reached, the back-checking continues to the preceding agent (lines 4-5) until it reaches A_1 . In case the total cost of the *CPA* is below the previous bound, agent A_1 updates the bound B (line 7) and informs all the agents of the new best solution and new bound (line 8). Finally, phase I is resumed by sending the *CPA* back to the last agent (line 9).

3.2 Synchronous Asymmetric B&B – 1-phase

The SyncABB-1ph algorithm is a combination of the SyncBB algorithm with the one-phase strategy. After each step of the algorithm, when an agent adds an assignment to the *CPA* and updates one direction of the bound, the *CPA* is sent back to the assigned agents to update its bound by the costs of all backwards directed constraints (back-checking). This is done by replacing the **CPA_MSG** message sent after each value assignment to the next agent (similar to SyncBB and SyncABB-2ph) with a **CPA_BACK_MSG** message to the preceding agent.

Algorithm 2 SyncABB-1ph – back-checking

```

when received (CPA_BACK_MSG, CPA, cost)
1:  $j \leftarrow CPA.lastId$ 
2:  $f \leftarrow$  cost of constraint with agent  $A_j$ 
3: if  $cost + f \geq B$  then
4:   send (CPA_MSG, CPA) to  $A_j$ 
5: else if  $A_i \neq A_1$  then
6:   send (CPA_BACK_MSG, CPA,  $cost + f$ ) to  $A_{i-1}$ 
7: else if  $A_j = A_n$  then
8:    $B \leftarrow cost + f$ 
9:   broadcast (NEW_SOLUTION, CPA,  $B$ )
10:  send (CPA_MSG, CPA) to  $A_n$ 
11: else
12:   $CPA.cost \leftarrow cost + f$ 
13:  send (CPA_MSG, CPA) to  $A_{j+1}$ 

```

The handling of **CPA_BACK_MSG** is slightly different in SyncABB-1ph (Algorithm 2). First, it is important to know the identity j of the initiator of the back-checking (in SyncABB-2ph it was always n). Consequently, agent A_n (in SyncABB-2ph) is now replaced with A_j (lines 2,4). Additionally, when the back-checking is complete (reaches A_1) and A_j is not the last agent (line 11), the algorithm simply sends the *CPA* to the next assigning agent A_{j+1} (line 13).

When the bound held by the agents is breached, a new value must be assigned, and the state of the CPA should be restored. The state of the CPA includes assignments of all higher priority agents and the cost (both backward and forward) associated with them. As the assignment does not change when back-checking, only the cost should be restored. This cost is the value of the latest CPA message received by the assigning agent from agent A_1 . As a result, the incremental cost of the constraint f (line 2), only includes the cost incurred by the assigning agent (agent A_j).

The main motivation for this one-phase version is that when the bound is breached, the CPA can be returned to the initiator of the back-checking (line 4), which in many cases will not be A_n . This can lead to effective pruning of the search space, which is not done very well in the two-phase strategy.

3.3 Semi-Synchronous Asymmetric B&B

The SemiSyncABB algorithm (Algorithm 3) is another one-phase version of SyncBB. It performs a synchronous branch & bound search for one side of the constraints and asynchronously back-checks the other side. The main difference from the fully-synchronous version (SyncABB-1ph - Algorithm 2) is that the search does not wait for the results of the back-checking. Instead, the semi-synchronous version sends after each value assignment both a **CPA_MSG** message to the next agent and a **CPA_BACK_MSG** message to the preceding agent (unless this is the last agent, for which only the **CPA_BACK_MSG** message is sent). Since the CPA is now moving forward concurrently with the back-checking process, there is no need to notify the next assigning agent in case the bound was not breached.

Algorithm 3 SemiSyncABB – back-checking

```

when received (CPA_BACK_MSG,  $CPA$ ,  $cost$ )
1:  $j \leftarrow CPA.lastId$ 
2:  $f \leftarrow$  cost of constraints with upper-indexed agents ( $A_{i+1}..A_j$ )
3: if  $cost + f \geq B$  then
4:   send (CPA_MSG,  $CPA$ ) to  $A_j$ 
5: else if  $A_i \neq A_1$  then
6:   send (CPA_BACK_MSG,  $CPA$ ,  $cost + f$ ) to  $A_{i-1}$ 
7: else if  $A_j = A_n$  then
8:    $B \leftarrow cost + f$ 
9:   broadcast (NEW_SOLUTION,  $CPA$ ,  $B$ )
10:  send (CPA_MSG,  $CPA$ ) to  $A_n$ 

```

Due to the added asynchronicity, time-stamps must be used in order to detect and discard obsolete CPAs (cf. [2]). Consequently, when the bound is breached and a new value is about to be assigned (Algorithm 3, lines 3-4), a notification of the new time-stamp is sent to all the relevant agents (all the agents after A_j) – added to the body of the if statement.

3.4 Asymmetric Two-Way Bounding

To achieve a larger degree of asynchronicity, one can build upon an existing and efficient asynchronous DCOP algorithm. The AFB algorithm [2] updates forward bounds asynchronously, on symmetric DCOPs. For ADCOPs one can combine forward bounding with backward bounding (i.e. perform the back-checking asynchronously). We refer to this version as the Asynchronous Two-Way Bounding algorithm (ATWB).

The ATWB algorithm follows the pseudo-code of AFB [2] with a few changes. **BOUND_CPA** messages are now sent both forward and backwards whenever a value is assigned (procedure **assign_CPA**). Additionally, the last agent n cannot declare a new solution until it receives all the estimates from the backward bounding. Thus, the handling of ESTIMATE messages must be revised (Algorithm 4).

Algorithm 4 ATWB – receive estimate

```
when received (ESTIMATE, estimate)
1: save estimate
2: if ( $CPA.cost + \text{all saved estimates} \geq B$ ) then
3:   assign_CPA()
4: else if  $CPA$  is a full assignment and all estimates arrived then
5:    $B \leftarrow CPA.cost + \text{all saved estimates}$ 
6:   broadcast (NEW_SOLUTION,  $CPA$ ,  $B$ )
7:   assign_CPA()
```

When an estimate is received the algorithm checks whether the new estimate causes a breach of the bound (line 2). If this is the case, then a new value is assigned by the current agent (line 3). In case this is the last agent and all the backward estimates have arrived (line 4), the agent can declare a new solution (lines 5-6) and assign a new value (line 7).

It must be noted that when an agent i receives a **BOUND_CPA** message from an agent $j > i$ ordered after it (backward bounding), an accurate answer can be given since all the relevant assignments for the back-checking are already on the CPA. This is in contrast to the forward bounding estimates which provide a lower bound. The additional precomputed $h(v)$ function that gives a lower bound on the cost of constraints with succeeding agents in the order [2] cannot be used for backward bounding. However, a precomputed $h_2(v, j)$ function (per value v , per agent j which holds the current CPA) that gives a lower bound on the cost of constraints with agents that are after j in the order (all $k > j$) can be added to the estimation calculation. The $h_2(v, j)$ function can also be used in forward bounding as a lower bound for the back-checking of value v with all the agents between i and j ($j < k < i$). The h_2 function is additive, since it refers to the back-checking of yet unassigned variables.

4 Asymmetric distributed local search

Similar to DCOPs, running complete ADCOP algorithms on large problems is impractical, and the desire for an optimal solution is often replaced with the desire for a solu-

tion which can be found in a reasonable (bounded) time [14]. This section presents two new local search algorithms for finding a good solution for ADCOPs. Unlike ASyncBB, ASemiSyncBB and ATWB, in these algorithms agents may only act according to the local information at their disposal. This includes knowledge of their local problem, the assignment of their immediate neighbors and any information exchanged with these neighbors [14].

The main difference between the various local search algorithms is the effort put into coordinating moves. The first algorithm examined is Distributed Stochastic Search (DSA) [14]. Although originally introduced as an algorithm for solving DisCSPs and DCOPs, DSA can also be applied to ADCOPs. Its simple protocol avoids any coordination between agents. The second approach examined is Proposal Based Search (PBS). In PBS agents attempt to “convince” neighbors that the joint assignment of the proposer and receiver is beneficial. A move is made if it does not degrade the cost of the neighborhood of the agent. The last approach we examine, Asymmetric Coordinated Local Search (ACLS), can be viewed as a semi-coordinated DSA for ADCOPs.

4.1 Distributed Stochastic Search

DSA is a well studied algorithm which was successfully applied to solve both DCSPs and DCOPs [14, 7]. Agents participating in DSA use a random initial value, which is broadcast to all neighbors. The algorithm proceeds in synchronized rounds. In each round an agent selects a value which improves its current state (if such a value exists). A stochastic decision rule is used when considering a change of the current assignment to the newly found, state improving, value. If an agent assigns a new value, a notification of this new value is sent to all neighboring agents.

Algorithm 5 Distributed Stochastic Search

```

1:  $val \leftarrow$  random init value
2: while (stop condition not met) do
3:   collect neighbors' new value, if any, and update current  $state$ 
4:   find assignment  $val^*$  which improves current  $state$ , if any
5:   assign  $val \leftarrow val^*$  with probability  $p$ 
6:   if (new val was assigned) then
7:     inform all neighbors of value change

```

As can be seen in Algorithm 5, agents running DSA have minimal knowledge requirements. Selecting a candidate assignment is based only on the neighbors' assignments (line 3) and on local problem structure (line 4). Since agents in DSA consider only alternative values which (weakly) improve their local state, in standard (symmetric) DCOPs, DSA has a high probability to converge after a large number of iterations [9]. The convergence state is a local minimum of the objective function. That is, any change of an assignment by one agent, can only produce a lower quality solution. However, this is not true for ADCOPs. Convergence states are not guaranteed to be local

minima. Consider the following example, in which two agents A and B are connected by the following constraint (and attempt to maximize the overall gain):

- $\langle x_A, x_B \rangle, U_A = 1, U_B = 1.$
- $\langle x_A, y_B \rangle, U_A = 5, U_B = 0.$
- $\langle y_A, x_B \rangle, U_A = 0, U_B = 5.$
- $\langle y_A, y_B \rangle, U_A = 4, U_B = 4.$

It is easy to verify that DSA may only converge when the assignment is $\langle x_A, x_B \rangle$, yet any single move by either A or B will result in a higher quality solution. Furthermore, a DSA solving an ADCOP may even fail to converge at all (take for example a variant of the “matching pennies” [8] example in which agents will continue changing their assignments to counter the opponent and thus improve their own local state. Although any state is also the optimal state, the convergence will not be in terms of assignments).

4.2 Proposal Based Search

PBS attempts to drive neighboring agents to a locally optimal assignment. Like DSA, PBS also proceeds in synchronized rounds. However, each round in PBS is composed of the four steps sketched in Algorithm 6.

Algorithm 6 Proposal Based Search

```

when in PROPOSAL phase:
1: if (neighbors changed value) then
2:    $improvements \leftarrow$  all assignment profiles with value  $\leq state$ 
3:    $proposal \leftarrow$  randomSelect( $improvements$ )
4:    $prop\_value \leftarrow$  valueOf( $proposal$ )
5:   for all  $n \in neighbors$  do
6:     send ( $proposal_i, proposal_n$ )
when in IMPACT phase:
7: collect all neighbor’s proposals, if any
8: for all  $n \in neighbors$  do
9:   send worst impact value
when in VALUE phase:
10: for all  $m \in$  neighbor’s impact message do
11:    $prop\_value \leftarrow prop\_value + m.data$ 
12: for all  $n \in neighbors$  do
13:   send ( $prop\_value - state$ )
when in UPDATE phase:
14:  $val \leftarrow proposal_n : \max\{prop\_value_i\}$ 
15: for all  $n \in neighbors$  do
16:   send ( $val$ )

```

PBS also uses a random initial assignment, and continues running until some condition is met. An agent running PBS begins by solving an inner COP which yields the set of all assignment profiles involving both the agent and its neighbors which improve the

state of the solving agent (line 2). This set is only recalculated when a neighbor changes its value. Once an improving profile set is generated the agent randomly picks an assignment profile and sends to its peers the relevant part of it, (e.g., the agent's value and the peer's proposed value, lines 3-6). Each peer receiving such a proposal calculates an estimate of the worst case state it will be in after neighbors proposals were either accepted or rejected. The value of this state (i.e., the number of broken constraints in the case of Max-DCSPs) is sent back to the proposing agent (lines 7-9). Once the impact of the proposal over all neighbors is known, the expected gain from the proposal is sent to all neighbors (lines 10-13), and a proposal yielding the best improvement is accepted by agents (lines 14-16).

The idea behind PBS is to avoid superficial local gains of a single agent which degrade the global state value, while making ever improving proposals.

4.3 Asymmetric Coordinated Local Search

In a third algorithm - Asymmetric Coordinated Local Search (ACLS) agents attempt to combine information from their surroundings, but do so cautiously. Similarly to DSA and PBS, ACLS proceeds in synchronized rounds, each divided into three steps.

Algorithm 7 Asymmetric Coordinated Local Search

```

when in PROPOSAL phase:
1: if (neighbors changed value) then
2:   improvements  $\leftarrow$  all values which improve local state
3:   prop_assignment  $\leftarrow$  randomSelect(improvements)
4:   prop_value  $\leftarrow$  valueOf(proposal)
5:   for all  $n \in$  neighbors do
6:     send (prop_assignment)
when in IMPACT phase:
7:   for all prop_assignment messages do
8:     send local constraint  $\langle$ assignmenti, prop_assignment $\rangle$ 
when in UPDATE phase:
9:   for all  $m \in$  neighbor's impact message do
10:    prop_value  $\leftarrow$  prop_value +  $m.data \times$  WARINESS
11:   if prop_value < state then
12:     assign val  $\leftarrow$  proposed_assignment with probability  $p$ 
13:     for all  $n \in$  neighbors do
14:       send (val)

```

A sketch of ACLS is given in Algorithm 7. After a random initial assignment, the agents continue running until some condition is met. At each step, agents first attempt to find the set of assignments which can improve the local state (line 2). Note that this differs from PBS - it locates personal values as opposed to complete assignment profiles. As a result this requires only a linear scan of all values in the agent's domain. Once all improving assignments are found, a proposal is randomly selected based on

the distribution of gains from each proposal (for example, if, when maximizing, assigning x gains 5, and y gains 10, then the chance that x is selected is $\frac{1}{3}$, and for y , $\frac{2}{3}$). This proposal is sent to all neighbors (lines 3-6). An agent receiving a proposal immediately responds with the value of constraint between its current assignment and the proposed assignment (line 8). Finally, when all impact messages arrive, the agent may assess the potential gain or loss from the assignment (line 10). To avoid the over “rigidness” of PBS agents, ACLS agents use a *WARINESS* value, representing the amount of coordination with their neighborhood. When this value is zero, ACLS produces results similar to those of DSA (albeit with a high overhead of network load and privacy degradation). An ACLS agent concludes a round by committing to a change with probability p , and notifying all neighbors.

ACLS attempts to coordinate moves to achieve higher quality solutions. It is willing to sacrifice some privacy, and triple the number of messages to do so.

5 Experimental evaluation

The ADCOPs used in the following experiments are random Asymmetric Max-DCSPs. Max-DCSP is a subclass of DCOP in which all constraint costs (weights) are equal to one [1]. This feature simplifies the task of generating random problems, since by using Max-DCSPs one does not have to decide on the costs of the constraints. Max-CSPs are commonly used in experimental evaluations of constraint optimization problems (COPs) [16]. Other experimental evaluations of DCOPs include graph coloring problems [1, 14], which are a subclass of Max-DCSP.

In our formulation we consider the constraint tightness p_2 as the average fraction of disallowed value pairs, as viewed by each agent involved in a given constraint. This implies that some pairs allowed by one of the involved agents are not allowed by the other, and vice versa. As a result, the fraction of cost-inflicting pairs is greater than the actual p_2 value. We refer to this fraction as p_{2-eff} . The expected value of p_{2-eff} is $p_{2-eff} = 1 - (1 - p_2)^2$.

To evaluate the performance of the ADCOP algorithms presented in section 3 we use randomly generated Asymmetric Max-DCSPs with 10 agents ($n = 10$) and 10 values ($k = 10$), constraint density $p_1 = 0.4$, and varying constraint tightness $0.1 \leq p_2 \leq 0.9$.

p_2	0.1	0.3	0.5	0.7	0.9
p_{2-eff}	0.19	0.51	0.75	0.91	0.99
SyncABB-1ph	292	34645	1.4E+7	2E+8	DNF
SemiSync-ABB	2241	1.6E+6	1.1E+8	1.6E+9	DNF
ATWB	11101	8.8E+6	5E+7	4.8E+8	2.2E+9

Table 1: Mean NCCCs in ADCOPs ($p_1 = 0.4$).

Table 1 presents our initial results for the mean number of non-concurrent constraint checks (NCCCs) [15]. The results of SyncABB-2ph are not included since the algorithm failed to finish its run even for the looser problems. This can be explained by the limited pruning of the search space caused by the two-phase strategy. The results show that SyncABB-1ph outperforms SemiSync-ABB. This suggests that asynchronicity actually impairs the performance in ADCOPs, which is usually not the case in symmetric DCOPs. It turns out to be better to make sure that the bound is not breached before resuming the search. In the loose problems ($p_2 \leq 0.3$), ATWB is clearly outperformed by SyncABB-1ph and SemiSync-ABB, due to the unnecessary overhead inflicted by forward bounding in satisfiable problems. However, as the problems become tighter, the effectiveness of forward bounding increases. In very tight problem ($p_2 = 0.9$) ATWB was the only algorithm that managed to finish its run within a reasonable time frame (20 hours).

Our experiments show that the relations between the algorithms with respect to the total number of sent messages is very similar to the NCCCs results in Table 1.

Local search experiments were run on larger problems. Each problem consisted of 30 agents, with a domain size of 10 and various value of p_1 and p_2 . Table 2 summarizes results of running a set of randomly generated problems with $p_1 = 0.5$, averaged over 25 instances. This summary compares the overall quality of a solution, measured in terms of broken constraints. ACLS with a WARINESS factor of 0.3 and 0.5 are examined against standard DSA.

p_2	0.1	0.2	0.3	0.4	0.5
$p_2 - eff$	0.19	0.36	0.51	0.64	0.75
ACLS(0.3)	3.12	24.88	56.92	95.36	142.32
ACLS(0.5)	4.04	31.04	66.68	105.52	159.2
DSA	2.79	48.84	83.12	127.92	166.4

Table 2: Mean quality of solutions ($p_1 = 0.5$).

PBS was not added to this table, due to consistently, and significantly lower quality results. This is attributed to the over conservative estimate of the impact caused by a proposal which resulted in what we termed “rigid” agents. Such agents hardly allowed any changes in their local environment, and as a result did very little exploration of the search space. A similar phenomenon was also registered with ACLS when the problem tightness increased. In such cases, the impact of any assignment change proposed by an agent, was likely to decrease neighbors state, and the move would be prohibited.

In contrast, the results in Table 2 clearly indicate that when solving MAX-ADCSPs with sufficiently low p_2 value (yet high $p_2 - eff$) ACLS provides better solutions than DSA. ACLS with a WARINESS factor of 0.3 is better than 0.5 (or that of 1). This means that some coordination can improve the results, yet relying too much on the local neighborhood may result in poor performance.

6 Conclusions

DCOPs model real world combinatorial problems which are distributed by nature. The standard model assumes that constraints have a global cost (or utility) which is known to all agents involved in the constraint. In many realistic problems each of the parties involved in a constraint has its own valuation of the assignment. On the other hand, it is commonly assumed that a reasonable global view of these valuations would be to optimize their total sum.

The present paper proposes an Asymmetric Distributed Constraints Optimization (ADCOP) model, along with solving algorithms. In the case of complete algorithms, the more successful algorithms are the ones which consider the constraints in both directions during search. In the case of local search algorithms, the greedy nature of local search algorithms results many times in agents with conflicting goals which cannot simply find a locally improved assignment. Two local search algorithms, PBS and ACLS, which coordinate moves were proposed and investigated. For lower values of p_2 ACLS produced the best quality results, in some cases better by a factor of 2. Surprisingly, on tight problems, a stochastic standard algorithm (DSA) [14] outperformed the proposed asymmetric algorithms. It seems that in the case of local search, a symmetric stochastic approach is enough to avoid dead ends.

References

1. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraints optimization with quality guarantees. *Artificial Intelligence* **161:1-2** (January 2005) 149–180
2. Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward bounding. *J. of Artificial Intelligence Research* **34** (2009) 25–46
3. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proc. IJCAI-05, Edinburgh, Scotland, UK (2005) 266–271
4. Modi, J., Veloso, M.: Multiagent meeting scheduling with rescheduling. In: Proc. of the Fifth Workshop on Distributed Constraint Reasoning (DCR), CP 2004, Toronto (2004)
5. Modi, P., Veloso, M., Smith, S., Oh, J.: Cmradar: A personal assistant agent for calendar management. In: 6th Intern. Workshop on Agent-Oriented Information Systems (AOIS). (2004) 134–148
6. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In: Proc. 3rd Intern. Joint Conf. on Autonomous Agents & Multi-Agent Systems (AAMAS-04), NY, New York (2004) 310–317
7. Gershman, A., Grubshtein, A., Meisels, A., Rokach, L., Zivan, R.: Scheduling meetings by agents. In: Proc. 7th Intern. Conf. on Pract. & Theo. Automated Timetabling (PATAT 2008), Montreal (August 2008)
8. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: *Algorithmic Game Theory*. Cambridge University Press (2007)
9. Chapman, A.C., Rogers, A., Jennings, N.R.: A parameterisation of algorithms for distributed constraint optimisation via potential games. In: Proc. AAMAS Workshop on Distributed Constraint Reasoning (DCR-08), Estoril, Portugal (2008) 99–113
10. Roughgarden, T.: *Selfish Routing and the Price of Anarchy*. The MIT Press (2005)

11. Hirayama, K., Yokoo, M.: Distributed partial constraint satisfaction problem. In: Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP-97). (1997) 222–236
12. Meisels, A.: Distributed Search by Constrained Agents: Algorithms, Performance, Communication. Springer Verlag (2007)
13. Brito, I., Meisels, A., Meseguer, P., Zivan, R.: Distributed constraint satisfaction with partially known constraints. *Constraints* **14** (2009) (To appear).
14. Zhang, W., Xing, Z., Wang, G., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraints optimization problems in sensor networks. *Artificial Intelligence* **161:1-2** (January 2005) 55–88
15. Zivan, R., Meisels, A.: Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence (AMAI)* **46** (2006) 415–439
16. Larrosa, J., Schiex, T.: Solving weighted csp by maintaining arc consistency. *Artificial Intelligence* **159** (2004) 1–26