

# Measuring Distributed Constraint Optimization algorithms <sup>\*</sup>

Amir Gershman<sup>(1)</sup>, Roie Zivan<sup>(2)</sup>, Tal Grinshpoun<sup>(1)</sup>, Alon Grubstein<sup>(1)</sup> and Amnon Meisels<sup>(1)</sup>

<sup>(1)</sup>Department of Computer Science

<sup>(2)</sup>Department of Industrial Engineering and Management,  
Ben-Gurion University of the Negev,  
Beer-Sheva, 84-105, Israel

**Abstract.** Complete algorithms for solving DisCOPs have been a major focus of research in the DCR community in the last few years. The properties of these algorithms belong to very different categories. Algorithms differ by their degree of asynchronicity, by the method of their combinatorial part, and by dividing the problem into sub parts. The wide variety of different families of algorithms makes it hard to find a uniform method for measuring and comparing their performance. The present paper proposes a uniform performance scale which is applicable for all DisCOP algorithms. The proposed performance measure enables an evaluation of the different DisCOP algorithms on a uniform scale, which was not published before. Preliminary results are presented and display the hierarchy of DisCOP search algorithms according to their performance on random DisCOPs.

## 1 Introduction

The Distributed Constraint Optimization Problem (*DisCOP*) is a general model for distributed problem solving that has a wide range of applications in Multi-Agent Systems and has generated significant interest in the last five years [7, 9, 10, 13]. DisCOPs are composed of agents, each holding one or more variables. Each variable has a domain of possible value assignments. Constraints among variables (possibly held by different agents) assign costs to combinations of value assignments. Agents assign values to their variables and communicate with each other, attempting to generate a solution that is globally optimal with respect to the sum of the costs of the constraints [9, 10].

There is a wide scope in the motivation for research on DisCOPs, since they can be used to model many every day combinatorial problems that are distributed by nature. Some examples are the *Nurse Shifts assignment problem* [3], the *Sensor Network Tracking problem* [13], and *Log Based Reconciliation* [1].

DisCOPs represent real life problems that cannot or should not be solved centrally for several reasons, among them the lack of autonomy, a single point of failure, and the privacy of agents.

A number of algorithms were proposed in the last few years for solving DisCOPs. The simplest algorithm of these is the *Synchronous Branch and Bound* (*Synch.B&B*)

---

<sup>\*</sup> The research was supported by the Lynn and William Frankel Center for Computer Science, and by the Paul Ivanier Center for Robotics.

algorithm which is a distributed version of the well-known centralized Branch and Bound algorithm. Another algorithm which uses a Branch and Bound scheme is *Asynchronous Forward Bounding (AFB)* [2], in which agents perform sequential assignments which are propagated for bounds checking and early detection of a need to backtrack. A number of algorithms use a pseudo-tree which is derived from the structure of the constraints network in order to improve the process of acquiring a solution for the search problem. *ADOPT* [9] is such an asynchronous search algorithm in which assignments are passed down the pseudo-tree. Agents compute upper and lower bounds for possible assignments and send costs which are eventually accumulated by the root agent, up to their parents in the pseudo-tree.

Another algorithm which exploits a pseudo tree is *DPOP* [10]. In DPOP, each agent receives from the agents which are its sons in the pseudo-tree all the combinations of partial solutions in their sub-tree and their corresponding costs. The agent calculates and generates all the possible partial solutions which include the partial solutions it received from its sons and its own assignments and sends the resulting combinations up the pseudo-tree. Once the root agent receives all the information from its sons, it produces the optimal solution and propagates it down the pseudo-tree to the rest of the agents. Yet another very different approach was implemented in the *OptAPO* algorithm in which agents which are in conflict choose a mediator to whom they transfer their data and which solves the partial problem. This algorithm benefits when the underlying constraints network includes independent sections which can be solved concurrently.

The wide variety of algorithms makes it hard to compare their performance. While some of the algorithms perform exhaustive sequential distributed search (Synchronous Branch and Bound) others perform asynchronous search (ADOPT) or avoid performing distributed search by centralizing information at some level (OptAPO, DPOP).

Most former studies on DisCOP complete search algorithms measured the performance of the algorithm by *steps* of computation of the algorithm (or cycles) [7, 9, 10, 2]. While for some of the algorithms as *Adopt* and *AFB* this is a reasonable choice since in each step agents perform similar amount of computation, in other algorithms like *DPOP* and *OptAPO* it is not clear what a step is since its size is dependent on the amount of information that was gathered.

The goal of the present paper is to compare the performance of such widely different algorithms with drastically different behavior. The present paper presents a uniform measure which enables a comparison of all DisCOP algorithm according to the same scale. In order to establish a uniform scale, the atomic operation performed by each algorithm is presented and its analogy to the atomic operations of the other algorithms is analyzed.

The proposed performance measure enables a comparison between algorithms which were not compared before (as DPOP and AFB). Preliminary experimental results show the hierarchy of the performance of these algorithms on random DisCOPs.

## 2 Distributed Constraint Optimization

A *DisCOP* 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 possible value combination of a set of variables, and is of the form  $C : D_{i_1} \times D_{i_2} \times \dots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ . A *binary constraint* refers to exactly two variables and is of the form  $C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$ . A *binary DisCOP* is a DisCOP 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.  $vars(PA)$  is the set of all variables that appear in PA,  $vars(PA) = \{X_i \mid \exists a \in D_i \wedge (X_i, a) \in PA\}$ . A constraint  $C \in \mathcal{R}$  of the form  $C : D_{i_1} \times D_{i_2} \times \dots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$  is *applicable* to PA if  $X_{i_1}, X_{i_2}, \dots, X_{i_k} \in vars(PA)$ . The *cost of a partial assignment* PA is the sum of all applicable constraints to PA over the assignments in PA. A *full assignment* is a partial assignment that includes all the variables ( $vars(PA) = \mathcal{X}$ ). 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 and the delay in delivering a message is finite [9, 12]. 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).

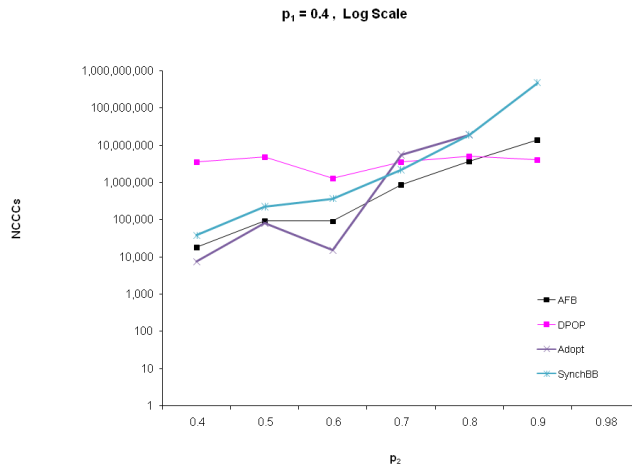
## 3 Performance measure for DisCOP algorithms

The common approach in evaluating the performance of distributed algorithms is to compare two independent measures of performance - time, in the form of steps of computation [6, 8, 14], and communication load, in the form of the total number of messages sent [6]. Non-concurrent computation effort, in systems with no message delays, are counted by a method similar to that of Lamport's logical clocks [4, 8, 14]. Every agent holds a counter of constraint checks performed. Every message carries the value of the sending agent's counter. When an agent receives a message it updates its counter to the largest value between its own counter and the counter value carried by the message. By reporting the cost of the search as the largest counter held by some agent at the end of the search, a measure of concurrent search effort is achieved that is close to Lamport's logical time [4]. This measure can be used to count the number of non-concurrent constraint checks performed (*NCCCs*) and to incorporate the local computational effort of agents in each step [8, 14]. It can also include some other type of atomic operation, such as a non-concurrent step of computation (cf. [2]).

In order to find a performance measure and scale that represents concurrent run-time and is applicable for all DisCOP algorithms, it is easy to follow the approach of our former study on performance measures for *Distributed constraints satisfaction problems* (*DisCSPs*) [15]. The simple idea is to measure the longest sequence of non-concurrent

atomic operations (e.g., constraint checks). In order to generate a concurrent and asynchronous performance measure for DisCOPs, one needs to understand the atomic operation that is relevant for each of the DisCOP algorithms. These operations can be very different for different families of algorithms.

1. *SynchBB*. In synchronous branch and bound, at each step of the algorithm an agent tries to assign a value to the current assignment without causing the lower bound to reach the upper bound. To this end, the costs of all the constraints between the new assigned value and the current assignments need to be added to the lower bound. Therefore, counting constraints checks as in the case of DisCSP measures seems adequate [15].
2. *AFB*. In the case of AFB besides the sequential assignment procedure, which is similar to SynchBB, agents perform asynchronous concurrent checks of bounds. In order to generate these bounds agents holding unassigned variables receive partial assignments and calculate their bound (the lowest cost of any of the values they can assign). Since in order to calculate a bound an agent must check the constraints between the assignment received and its own values we have agents performing constraints checks concurrently. Thus, non-concurrent constraints checks is an adequate measure for this algorithm too [2].
3. *ADOPT*. The atomic operation which agents perform in ADOPT is similar to the asynchronous checks of AFB. An agent checks its best assignment according to the context it holds each time the context changes. Again, searching for the assignment with the lowest cost includes checking the cost of constraints. Thus, non-concurrent constraints checks is an adequate measure in the case of ADOPT. Notice that in the case of ADOPT as in the case of AFB one neglects to count the processing of information which was received via messages from lower priority agents (i.e., cost messages). This makes the performance of ADOPT, in units of *NCCCs*, a lower bound on the implementation independent concurrent run-time measure.
4. *OptAPO*. Most of the computation in OptAPO is done by mediators that solve sub-problems of the DisCOP. A mediator centralizes a relevant portion of the problem, and conducts a branch and bound search on it. Similarly to the case of SynchBB, counting constraints checks is an adequate measure for the computation that a mediator agent performs while conducting the branch and bound search. The rest of the computation in OptAPO is done when an agent checks its agent view to detect suboptimal conditions that trigger the mediations. Such checking of the agent view can also be measured using constraints checks. Since several agents can perform as mediators at the same time, non-concurrent constraints checks must be used to measure the run-time performance of the OptAPO algorithm.
5. *DPOP*. The case of DPOP differs drastically from former versions of both synchronous and asynchronous branch and bound search. In DPOP agents do not perform search and the steps agents perform are computations that are needed in order to fill a matrix which expresses their optimal assignments' cost to each combinations of assignments of constrained agents with higher priority. The generation of every cell in this matrix relies on the checking of constraints with higher priority agents against every possible local assignment of the current agent. This is similar to the operation of calculating a cost for a context in the case of ADOPT and AFB.



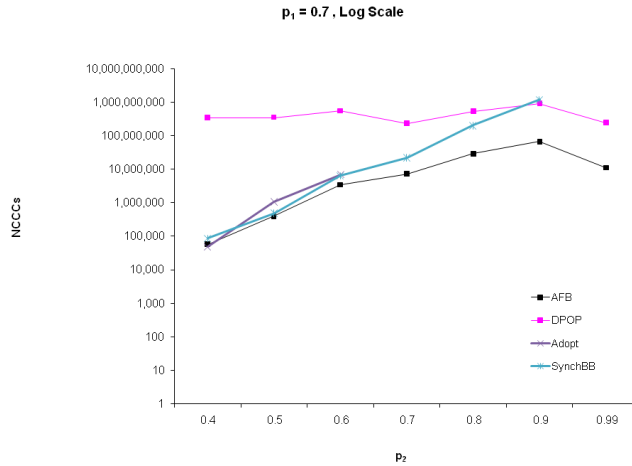
**Fig. 1.** Empirical evaluation on random *Max-DisCSP* of low constraints density ( $p_1 = 0.4$ ).

In the experimental evaluation of the run-time performance of DPOP one needs to count the number of constraint checks performed by the agent in order to generate each cell in its matrix. The constraints checks that are performed during the generation of each matrix are the atomic operations that DPOP performs. Similarly to the other algorithms, operations that could have been performed concurrently are counted only once.

## 4 Experimental Evaluation

All experiments were performed on a simulator in which agents are simulated by threads which communicate only through message passing. The Distributed Optimization problems used in all of the presented experiments are random *Max-DisCSPs*. *Max-DisCSP* is a subclass of *DisCOP* in which all constraint costs (weights) are equal to one [9]. The network of constraints, in each of the experiments, is generated randomly by selecting the probability  $p_1$  of a constraint among any pair of variables and the probability  $p_2$ , for the occurrence of a violation (a non zero cost) among two assignments of values to a constrained pair of variables. Such uniform random constraints networks of  $n$  variables,  $k$  values in each domain, a constraints density of  $p_1$  and tightness  $p_2$  are commonly used in experimental evaluations of CSP algorithms (cf. [11]). *Max-CSPs* are commonly used in experimental evaluations of constraint optimization problems (*COPs*) [5]. Other experimental evaluations of *DisCOPs* include graph coloring problems [9, 13], which are a subclass of *Max-DisCSP*. The method described in [8] is used for counting non-concurrent computational constraint checks and steps.

Figures 1 and 2 presents our preliminary empirical evaluation of the algorithms described in section 3, except for *OptAPO*. Our implementation of the *OptAPO* algo-



**Fig. 2.** Empirical evaluation on random *Max-DisCSP* of high constraints density ( $p_1 = 0.7$ ).

rithm is not yet complete and thus not included in the evaluation, however we plan on adding it in future work. The problems solved are randomly generated *Max-DisCSP* with  $n = 10$  variables and agents,  $k = 10$  values in each domain, a constraint density of either  $p_1 = 0.4$  (figure 1) or  $p_1 = 0.7$  (figure 2), and varying constraint tightness  $0.4 \leq p_2 \leq 1$ . Notice that in both figures, the Y-Axis is in logarithm scale.

In figure 2 the graph for *Adopt* is incomplete, since *Adopt* failed to terminate in a reasonable amount of time beyond a constraint tightness of  $p_2 \geq 0.7$ . *SynchBB* as well, failed to terminate in time beyond a constraint tightness of  $p_2 > 0.9$ .

#### 4.1 Analysis of Results

The following observations can be made, based on the above empirical evaluation:

- For random *Max-DisCSP*s of low constraint density and low constraint tightness *Adopt* performs well, with *AFB* and even *SynchBB* being fairly close. *DPOP* performs extremely poor. We believe that the weak performance of *DPOP* is due to the lack of pruning of the search space. All other branch and bound based algorithms prune the search space by the use of bounds.
- The run time of *Adopt* and *SynchBB* for random *Max-DisCSP*s of low constraint density and high constraint tightness grows at a high exponential rate. *DPOP* and *AFB* perform far better, by two orders of magnitude.
- *DPOP*'s performance remains fairly constant regardless of the problem's tightness. This is to be expected since *DPOP* does not perform search or pruning. *DPOP* computes the same size of matrices regardless of the tightness. A change in the problem's tightness would only effect the content of the matrices.

- For random *Max-DisCSPs* of high constraint density and low constraint tightness, the performance of *Adopt*, *AFB* and *SynchBB* is fairly similar, and again *DPOP*'s performance is much worse.
- For high constraint density and high constraint tightness the performance of *Adopt* and *SynchBB* deteriorates exponentially and does not even terminate on all our problems. *Adopt* is worse than *SynchBB* since it failed to terminate at a lower tightness value (of 0.7). *DPOP*'s run time is high but it always terminated and did not increase at a high exponential rate like *Adopt* and *SynchBB*. *AFB* is clearly the best performing algorithm in this case, outperforming *DPOP* by several orders of magnitude.
- The performance of *AFB* exhibits a "phase-transition". Its run time increases as the problem's difficulty (tightness) increases, and then suddenly decreases by an order of magnitude at the very extreme high values of  $p_2$ . This phenomena was previously observed and discussed in [2].
- *Adopt* appears to be effected the most by the increase of the problem's tightness. We believe that this is due to the fact that *Adopt* is unable to converge on a solution quickly (since the problem is harder). This leads to excess (greedy) context switches along with an exponential increase in the number of messages sent. Each agent in *Adopt* sends out two message following every single message received. This causes the algorithm's run time to increase at a very high exponential rate.

## 5 Discussion

Most former studies of complete DisCOP search algorithms compared the algorithms by the number of (non-concurrent) steps of computation they perform. The problem with this measure is that it is not atomic and thus in different algorithms can be of different computational cost. The present paper uses appropriate units for atomic computational steps - non-concurrent constraints checks. This allows us to put the different algorithms on the same scale, including algorithms which were never compared before. The experiments presented in this paper show a different behaviour for each of the evaluated algorithms. For lower density random *Max-DisCSPs* *DPOP* and *AFB* perform better than the other algorithms on the harder problem instances ( $p_2 > 0.6$ ). In contrast, *AFB* is the clear "winner" for the whole range of problem difficulty. The advantage of the *AFB* algorithm is probably related to its use of pruning techniques through asynchronous bounding. This raises an interesting observation about the *DPOP* algorithm whose performance depends on the problem's structure alone. We hope to present a more comprehensive comparison of the algorithms that were evaluated in this paper with the *OptAPO* algorithm in the near future.

## References

- [1] Y. Chong and Y. Hamadi. Distributed log-based reconciliation. In *Proc. ECAI-06*, pages 108–113, August 2006.
- [2] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward-bounding for distributed constraints optimization. In *Proc. ECAI-06*, pages 103–107, August 2006.

- [3] E. Kaplansky and A. Meisels. Distributed personnel scheduling: Negotiation among scheduling agents. *Annals of Operations Research*, 2005.
- [4] L. Lamport. Time, clocks, and the ordering of events in distributed system. *Communication of the ACM*, 2:95–114, April 1978.
- [5] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159:1–26, 2004.
- [6] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Series, 1997.
- [7] R. Mailer and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proc. AAMAS-2004*, pages 438–445, July 2004.
- [8] A. Meisels, I. Razgon, E. Kaplansky, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR*, pages 86–93, Bologna, July 2002.
- [9] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimization with quality guarantees. *Artificial Intelligence*, 161:1-2:149–180, January 2005.
- [10] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.
- [11] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.
- [12] M. Yokoo. Algorithms for distributed constraint satisfaction problems: A review. *Autonomous Agents & Multi-Agent Sys.*, 3:198–212, 2000.
- [13] 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*, 161:1-2:55–88, January 2005.
- [14] R. Zivan and A. Meisels. Dynamic ordering for asynchronous backtracking on discsps. *Constraints*, 11(2,3):179–197, 2006.
- [15] R. Zivan and A. Meisels. Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence (AMAI)*, (accepted for publication), 46:415–439, October 2006.