Global Constraints in Distributed CSP: Concurrent GAC and Explanations in ABT

Mohamed Wahbi, Kenneth N. Brown
{mohamed.wahbi,ken.brown}@insight-centre.org

Insight Centre for Data Analytics, School of Computer Science and IT,
University College Cork, Ireland

Abstract. The expressiveness of Distributed CSP has been recently enhanced to include global constraints. Careful reformulation of contractible global constraints has been shown to improve efficiency. In this paper, we first show that explained global constraints further improves the efficiency in distributed problems, sometimes by over two orders of magnitude. We then propose maintaining GAC concurrently for any global constraint, without reformulation. We show empirically that concurrent GAC significantly reduces both message passing and computation time, achieving an order of magnitude improvement on some distributed meeting scheduling problems.

1 Introduction

The *distributed constraint satisfaction problem* models decision problems where physically distributed agents control different decision variables. To solve the problems, the agents must perform local computation and exchange messages, to communicate value choices, inferred no-goods, algorithm control decisions or problem descriptions. One of the main reasons for the success of centralized constraint programming is the use of *global constraints*, in which a relation between a group of variables comes equipped with powerful filtering algorithms, which quickly infer consequences of value assignments and greatly reduce search. Implementing global constraints in Distributed CSP has been less successful, because the distributed control over the variables has led to delayed filtering. If Distributed CSP is to be more widely applied to real problems, more effective filtering for global constraints in distributed problems is required.

Three ways to represent global constraints in DisCSP have been recently proposed [3]. The *nested* representation of *contractible* global constraints offers significant improvements over the other approaches. In addition, propagating unconditional no-goods (that is, values that cannot be in a solution, regardless of other choices) while enforcing generalized arc-consistency (GAC) was also shown to offer an improvement.

Here, we propose two further improvements to the handling of global constraints in Distributed CSP, implemented in ABT.[1] First, we use *explained* global constraints, which allows us to generate more efficient no-goods– that is, given an ordered partial

[1] We focus on ABT, since it is the only DisCSP algorithm (apart from MACA [34]) we know of that has been extended to maintain arc consistency. Our methods are applicable to any algorithm, but without global constraint filtering would provide little benefit.

assignment of values to variables, we identify the earliest inconsistent subset [13, 8]. Each agent evaluating a constraint maintains a copy of the domains of all other variables in the constraint, and whenever it receives either a value assignment or a no-good, it maintains GAC on these domains. Secondly, we introduce a full representation of each global constraint by each agent constrained by it. This allows us to maintain GAC concurrently at each agent. Our aim is to trade off this potentially redundant filtering for faster identification of no-goods, and reduce search and message passing.

We evaluate our algorithms empirically on a set of benchmarks from the literature, including random problems, quasi-groups with holes, and distributed meeting scheduling problems. We demonstrate that the use of explained constraints significantly improves performance over previous methods on the harder problems, sometimes achieving over two orders of magnitude reduction in both non-concurrent computation and messaging. We then show that concurrent filtering offers another significant improvement when added to explained constraints, achieving up to an order of magnitude improvement on the meeting scheduling problems in both non-concurrent computation and messaging, without requiring any reformulation. Finally, we consider the total computation load over all agents, and we show that, surprisingly, concurrent filtering can reduce the total effort – it appears that early identification of relevant no-goods outweighs any redundant filtering.

This paper is structured as follows. Section 2 gives the necessary background on centralized CSP, global constraints, and distributed CSP. It then discusses the use of global constraints in distributed CSP. We present our use of explained global constraints on DisCSP in Section 3, followed by a comparison to previous work. Section 4 introduces concurrent filtering using the full representation of global constraints. We report experimental results in Section 5. Finally, we conclude the paper in Section 6.

2 Background and Related Work

2.1 Centralised CSPs and global constraints

The *constraint satisfaction problem* is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ is a set of variables $\{x_1, \ldots, x_n\}$, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains, where D_i is a finite set of values from which one value must be assigned to variable x_i, and $\mathcal{C}$ is a set of constraints. A constraint $C(X) \in \mathcal{C}$, on the ordered subset of variables $X = (x_{j_1}, \ldots, x_{j_k})$, is $C(X) \subseteq D_{j_1} \times \cdots \times D_{j_k}$, and specifies the tuples of values which may be assigned simultaneously to the variables in X. $|X|$ is the *arity* of $C(X)$, and X is its *scope*. A tuple $\tau = (v_{j_1}, \ldots, v_{j_k}) \in C(X)$ is a *support* for $C(X)$, and $\tau[x_i]$ is the value of x_i in τ. We denote by $C_i \subseteq \mathcal{C}$ all constraints that involve x_i. A *solution* is an assignment to each variable of a value from its domain, satisfying all the constraints.

A *global constraint* captures a relation over an arbitrary number of variables. For example, the ALLDIFF constraint states that the values assigned to the variables in its scope must all be different [22]. Filtering algorithms which exploit the specific structure of global constraints are one of the main strengths of constraint programming [23]. A value $v_i \in D_i$, $x_i \in X$ is *generalized arc-consistent (GAC)* with respect to $C(X)$ iff there exists a support τ for $C(X)$ such that $v_i = \tau[x_i]$, and for every $x_j \in X$, $x_i \neq x_j$,

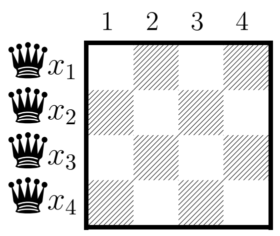
$\tau[x_j] \in D_j$. Variable x_i is GAC if all its values are GAC with respect to every constraint in C_i. A CSP is GAC if all its variables are GAC. During search, any value $v \in D_i$ that is not GAC can be removed from D_i.

Global constraint $C(X)$ is *binary-decomposable* without extra variables [4] if it is equivalent to a conjunction of binary constraints involving only variables in X. ALLDIFF is binary-decomposable. For example, ALLDIFF(x_1,x_2,x_3,x_4) is equivalent to $(x_1 \neq x_2) \wedge (x_1 \neq x_3) \wedge (x_1 \neq x_4) \wedge (x_2 \neq x_3) \wedge (x_2 \neq x_4) \wedge (x_3 \neq x_4)$. Global constraint $C(X)$, where $X = (x_{j_1}, \ldots, x_{j_{k+1}})$, is *contractible* [16] iff for any support $(v_{j_1}, \ldots, v_{j_k}, v_{j_{k+1}})$ for $C(X)$, then $(v_{j_1}, \ldots, v_{j_k})$ is a support for $C(x_{j_1}, \ldots, x_{j_k})$. ALLDIFF is a contractible constraint, since the projection of the supports for ALLDIFF(x_1, x_2, x_3, x_4) onto (x_1, x_2, x_3) are also supports for ALLDIFF(x_1, x_2, x_3). However, EXACTLY(k, X, v) that specifies that value v is assigned exactly to k variables in X is not contractible [2].

An *explanation* is a subset of the original constraints of the problem plus a set of decision constraints (variable assignments) made during the search which together are sufficient to justify domain reductions [13, 11]. Explanations were originally introduced [14] to improve intelligent backtracking, allowing the search to jump back to the cause of a failure. Computing an explanation for a reduction caused by binary constraints is relatively simple, but is more complex for a global constraint, where the explanation will depend on the chosen filtering algorithm. For example, for the ALLDIFF constraint [23], the filtering algorithm is based on computing a residual graph constructed from the maximum matching on the variable-value bipartite graph and from the possible values of variables in the constraint. [25] shows how to generate corresponding explanations: given the residual graph, the removal of an arc starting from a vertex belonging to a strongly connected component $\mathcal{S}_1$ to a distinct strongly connected component $\mathcal{S}_2$ is explained by all missing arcs from a descendant component of $\mathcal{S}_2$ to an ancestor component of $\mathcal{S}_1$ (since any one of these arcs would merge $\mathcal{S}_1$ and $\mathcal{S}_2$ into the same strongly connected component). Other global constraints have also been explained [14, 24, 13, 11, 27, 8, 9, 10], with the explanations used to improve the efficiency of backjumping.

2.2 Distributed CSPs

Distributed CSP (*DisCSP*) [37] models problems where distinct agents control different variables. DisCSP is a 4-tuple $(\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$ are as above, and $\mathcal{A}$ is a set of agents $\{A_1, \ldots, A_a\}$, where each variable $x_i \in \mathcal{X}$ is controlled by a single agent in $\mathcal{A}$. During a solution process, only the agent which controls a variable can assign a value to this variable. Without loss of generality, we assume each agent controls exactly one variable $(a = n)$, so we use the terms agent and variable interchangeably and no longer distinguish between A_i and x_i. Each agent A_i knows all constraints relevant to its variable (C_i) and the domains of other variables involved in these constraints (its *neighbours*). A variety of problems have been tackled using DisCSP, including tracking in sensor networks [1], distributed resource allocation [20] and distributed meeting scheduling [18]. Many different algorithms have been proposed, including asynchronous backtracking [36, 5], asynchronous forward checking



$\mathcal{A} = \{A_1, \ldots, A_4\}$

$\mathcal{X} = \{x_1, \ldots, x_4\}$

$\mathcal{D} = \{D_1, \ldots, D_4\}$,
where $D_i = \{1, 2, 3, 4\}$

$\mathcal{C} = \{c_1, \ldots, c_7\}$

$c_1 : | x_1 - x_2 | \neq 1$
$c_2 : | x_1 - x_3 | \neq 2$
$c_3 : | x_1 - x_4 | \neq 3$
$c_4 : | x_2 - x_3 | \neq 1$
$c_5 : | x_2 - x_4 | \neq 2$
$c_6 : | x_3 - x_4 | \neq 1$
$c_7 :$ AllDiff$(\mathcal{X})$

Fig. 1: The distributed n-queens problem (where $n = 4$)

[19], asynchronous aggregation [29], dynamic programming [21], partially centralised search [17], and dynamic ordering [28, 39, 31].

The original algorithm for DisCSP was Asynchronous Backtracking (ABT) [36, 5]. ABT is an asynchronous algorithm executed autonomously by each agent in the problem, and is guaranteed to converge to a global consistent solution (or detect inconsistency) in finite time. Each agent proposes values for its own variable to other agents, and reports no-goods. Agents operate asynchronously, but are subject to a known total priority order, $\prec$. For simplicity, we assume $\prec$ is the lexicographic ordering $(A_1, A_2, \ldots, A_n)$, with A_1 having highest priority. $\prec$ induces a directed acyclic graph, and constraints are directed according to $\prec$.

ABT uses the priority ordering to control the asynchronous search. Each agent selects an assignment for its variable that is consistent with known choices of higher priority neighbours, and then sends its selected value across the directed arcs to its lower priority neighbours. When no value is possible for a variable, the inconsistency is reported to a higher priority agent closest in the ordering, in the form of a no-good. The higher agent then adds the no-good to its constraint store. For a non-binary constraint $C(X)$ (arity > 2), only the lowest agent in the ordering in the scope of the constraint will evaluate $C(X)$, and only when it is totally instantiated [7, 3].

Fig. 1 shows a model of a distributed n-queens problem using an AllDiff constraint. The goal is to put 4 queens on the board such that no queen attacks another queen. There is an integer variable x_i for every row i. There are four agents, each of which controls one queen in one row. The domain of each x_i is $D_i = \{1, 2, 3, 4\}$. There exists a global AllDiff constraint (i.e., AllDiff(x_1, x_2, x_3, x_4)) that forbids the queens being placed in the same column. There is a binary constraint (e.g., c_1) between each pair of queens that forbids those queens being placed on the same diagonal (i.e., $| x_i - x_j | \neq | i - j | \; \forall i, j_{\neq i} \in \{1..4\}$).

2.3 Global Constraints in distributed CSPs

[3] formulated three different ways to model a global constraint in a DisCSP.

Direct Representation: applicable to all global constraints. The direct representation of the constraint $C(X)$ is a single copy of $C(X)$ to be evaluated by the lowest agent in its scope, A_i. Directed links from other agents in $C(X)$ to A_i are established.

In the example of Fig. 1, A_1 starts with no constraints. A_2 evaluates constraint c_1, A_3 evaluates c_2 and c_4, while A_4 evaluates all other constraints (i.e., c_3, c_5, c_6 and c_7).

Nested Representation: restricted to contractible global constraints. The nested representation of the constraint $\{C(x_{j_1}, \ldots, x_{j_k})\}$ is the set of constraints $C(x_{j_1}, \ldots, x_{j_m}) \mid m \in 2..k$. In the example of Fig. 1, constraint c_7 will be represented by 3 different constraints: (c_7) $\textsc{AllDiff}(x_1, x_2, x_3, x_4)$, ($c_8^n$) $\textsc{AllDiff}(x_1, x_2, x_3)$, and ($c_9^n$) $\textsc{AllDiff}(x_1, x_2)$. Constraints c_1 to c_7 will be evaluated by the same agents as in the direct representation. c_8^n is evaluated by agent A_3 and c_9^n by agent A_2.

Binary Representation: restricted to binary-decomposable global constraints. The binary representation of a constraint $C(X)$ is the set of constraints in its binary decomposition. In the example of Fig. 1, constraint c_7 will be represented by 6 constraints: (c_8^b) $(x_1 \neq x_2)$, (c_9^b) $(x_1 \neq x_3)$, (c_{10}^b) $(x_1 \neq x_4)$, (c_{11}^b) $(x_2 \neq x_3)$, (c_{12}^b) $(x_2 \neq x_4)$ and (c_{13}^b) $(x_3 \neq x_4)$. c_1 to c_6 will be evaluated by the same agents as in the direct representation. Agent A_2 will evaluate c_8^b, agent A_3 will evaluate c_9^b and c_{11}^b and A_3 will evaluate c_{10}^b, c_{12}^b and c_{13}^b.

In addition, in order to take advantage of the standard filtering algorithms for global constraint in DisCSPs, [3] proposed ABT-UGAC (ABT with unconditional GAC). ABT-UGAC maintains a limited form of GAC restricted to unconditional deletions (values removed by a no-good with an empty precondition). In a preprocessing step, the DisCSP is made GAC. Unconditional GAC is then enforced in ABT as follows. When receiving a no-good with an empty precondition justifying the removal of its value, an agent A_i can unconditionally delete its value from D_i. This deletion may then propagate, and cause further deletions (see [3] for details).

[3] showed that the direct representation is the least efficient, while the nested one performs best. ABT-UGAC always improves the performance.

3 Maintaining GAC in ABT

In ABT, the lowest priority agent in the scope X is in charge of evaluating a global constraint $C(X)$ when it is fully instantiated [7, 3]. This method of evaluating global constraints is a major weakness of representing global constraints in ABT. First, it does not take advantage of the global constraints' filtering power. Both nested and direct representations only use global constraints as checkers instead of using their filtering algorithm to prune inconsistent values. Second, it produces chronological backtracks because the deduced no-good will contain all assignments of the agents on the global constraints. Thus, it creates unnecessary work for the agents because it does not address the real reason for the failure.

For example, in the direct representation of the problem of Fig. 1, agent A_4 will not evaluate constraint c_7 until all assignments of variables x_1, x_2 and x_3 are received. Thus, agent A_4 may receive the assignments $x_1 = 1 \wedge x_2 = 1 \wedge x_3 = 1$. In this situation, A_4 will send a no-good $x_1 = 1 \wedge x_2 = 1 \rightarrow x_3 \neq 1$ to A_3. Then, A_3 will change its value to 4 because 2 is removed by c_4 and 3 by c_2. The new assignment ($x_3 = 4$) will lead to another deadend in A_4 with the following no-good ($x_1 = 1 \wedge x_2 = 1 \rightarrow x_3 \neq 4$). Once it receives this no-good, A_3 discovers the real reason for the failure (i.e., $x_1 = 1 \rightarrow x_2 \neq 1$). Thus, unnecessary work is performed when using this evaluation mechanism.

Instead of evaluating a constraint only when it is fully instantiated, we propose maintaining GAC each time an event occurs on a variable involved in a constraint in the agent's constraint store. In order to get more precise no-goods we use explained global constraints, so that all domain reductions are justified [13, 8], and when a dead-end occurs, we will get more informative no-goods. For ALLDIFF, we implement the explained filtering algorithm described previously [23, 25]. For ATMOST(k, X, v) and ATLEAST(k, X, v), we modify the implementations of [14] as below, and then use both methods together for the EXACTLY(k, X, v) constraint.

ATMOST(k, X, v): whenever a constrained variable $x_i \in X$ is assigned the occurrence value (v) or its domain is reduced to the occurrence value ($D_i = \{v\}$), we label it as 'sure' ($sure(i) = true$). A 'sure' variable can be either assigned (i.e, $x_i = v \mid x_i \in X \wedge sure(i)$) or unassigned (i.e, $D_j = \{v\} \mid x_j \in X \wedge sure(j)$). If the number of 'sure' variables equals the number of occurrences (k), the value v will be removed from the domain of all other variables in X. These removals are explained by the union of the set of decision constraints of 'sure' assigned variables x_i (i.e., $x_i = v \mid x_i \in X \wedge sure(i)$) with the union of the explanations that reduces the other sure variable domains to v (i.e., $D_j = \{v\} \mid x_j \in X \wedge sure(j)$).

ATLEAST(k, X, v): all constrained variables $x_i \in X$ that can be assigned v are labelled as 'possible' ($possible(i) = true$). If a variable $x_j \in X$ can not be assigned v its 'possible' flag is false ($possible(j) = false$). Whenever the number of 'possible' variables equals the number of occurrences (k), the domains of all these 'possible' variables are reduced to v (i.e., $D_i = \{v\} \mid x_i \in X \wedge possible(i)$). These reductions are explained by the union of explanations that justify the removal of v from the domain of other variables (i.e., $v \notin D_j \mid x_j \in X \wedge \neg possible(j)$).

To maintain GAC, some minor changes to ABT are required. First, when an agent A_i receives a message containing an assignment or a no-good, it updates its AgentView with the given assignment. The AgentView of an agent stores the most up-to-date assignments of its higher neighbours [5]. Then, A_i adds the received constraint (assignment or no-good) to its constraint store (no-goods inconsistent with the AgentView are removed to keep the space complexity polynomial [5]). Next, A_i maintains GAC. Second, whenever an agent A_i assigns a value to its variable ($x_i = v_i$) it adds its assignment to its constraint store (C_i). Next, it maintains GAC. When agent A_i maintains GAC using explained constraints each domain reduction is justified by a precise explanation. Thus, whenever a dead-end occurs, the no-good that will be generated will be more precise (a subset of the AgentView). However, if non-explained constraints are used, whenever a dead-end occurs, the generated no-good contains all assignments in the AgentView.[2] If the domain of a variable is emptied while maintaining GAC, agent A_i generates a new no-good from the explanations stored for value removals of that variable. If the generated no-good contains the assignment of agent A_i (i.e., v_i is not consistent after maintaining GAC), A_i tries to assign another value to x_i. If no value is possible for x_i or if the generated no-good doesn't contain its assignment, A_i back-

[2] Note that replacing the global constraint checker with the global constraint filter does not change the operation of ABT, apart from detecting earlier no-goods and generating more informed messages.

tracks by sending the no-good justifying the failure to the closest agent in the no-good. If C_i is GAC, A_i sends its new assignment ($x_i = v_i$) to all its lower priority neighbours.

Now, in the direct representation example, when agent A_4 maintains GAC on its constraints (i.e., c_3, c_5, c_6 and c_7) it can directly discover the no-good (i.e., $x_1 = 1 \rightarrow x_2 \neq 1$) without the assignment of x_3.

3.1 Evaluation of explained global constraints in ABT

We experimentally compare ABT(direct), ABT(nested), ABT-UGAC(direct) and ABT-UGAC(nested) as presented in [3] (evaluating constraints when they are totally instantiated) to ABT-GAC(direct) and ABT-GAC(nested) that maintain GAC thanks to explained constraints as presented above. Algorithms are tested using the static *max-degree* agent ordering. For all ABT versions we implemented an improved version of Silaghi's solution detection [30] and counters for tagging assignments. All experiments were performed on the DisChoco 2.0 platform [33],[3] in which agents are simulated by Java threads that communicate only through message passing.

We reproduce the benchmark problems of [3]. Algorithms are evaluated on uniform random binary DisCSP where some global constraints are injected. We evaluate the performance of the algorithms by the total number of exchanged messages among agents during algorithm execution ($\#msg$) [15] and non-concurrent computation. The non-concurrent computation is measured by the number of non-concurrent constraint checks ($\#ncccs$) [38], as a proxy for elapsed time. All GAC effort is counted in $\#ncccs$, as in [3], where for each call of the ALLDIFF propagator, which computes a maximum matching in a graph, we increase $\#ncccs$ by the degree of that graph. For other constraints we increase $\#ncccs$ by the size of the data structure used in that constraint.

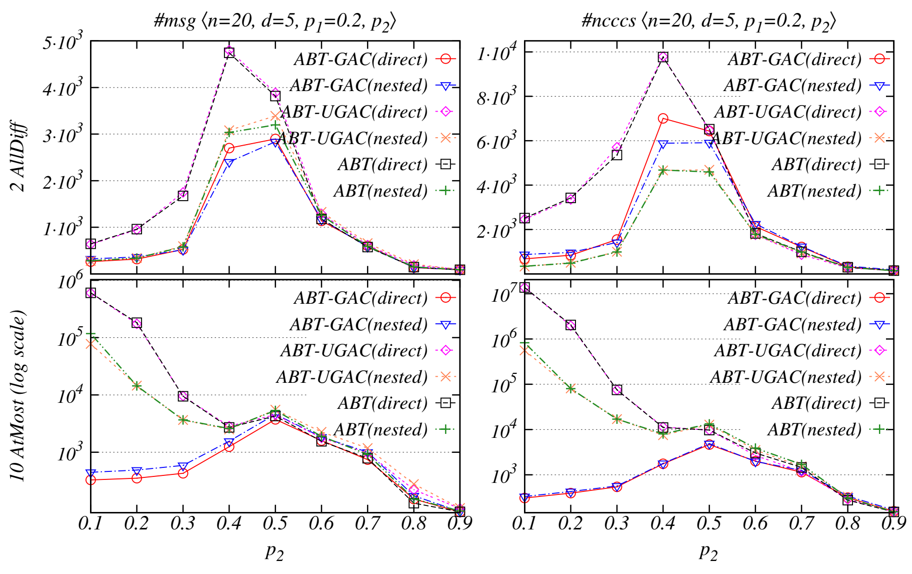
Uniform binary random DisCSPs are characterized by $\langle n,\ d,\ p_1,\ p_2\rangle$, where n is the number of agents/variables, d is the number of values in each domain, p_1 is the network connectivity defined as the ratio of existing binary constraints to possible binary constraints, and p_2 is the constraint tightness defined as the ratio of forbidden value pairs to all possible pairs. We solved instances of two classes of constraint graphs: sparse graphs $\langle 20, 5, 0.2, p_2\rangle$ and dense graphs $\langle 20, 5, 0.7, p_2\rangle$. We varied the tightness from 0.1 to 0.9 by steps of 0.1. For each pair of fixed density and tightness (p_1, p_2) we report the average over 100 instances.

We generate two types of benchmarks: the ALLDIFF benchmark and the ATMOST benchmark. An ATMOST(k, X, v) constraint specifies that at most k variables in X are assigned value v. In the ALLDIFF benchmark, each binary instance includes 2 ALLDIFF constraints, each involving 5 randomly chosen variables. In the ATMOST benchmark, each binary instance includes 10 ATMOST(k, X, v) constraints, each involving from 3 to 10 randomly chosen variables (i.e., $3 \leq |X| \leq 10$). The value v is randomly chosen in the set of values in domains and its number of occurrences $k = 2$.[4]

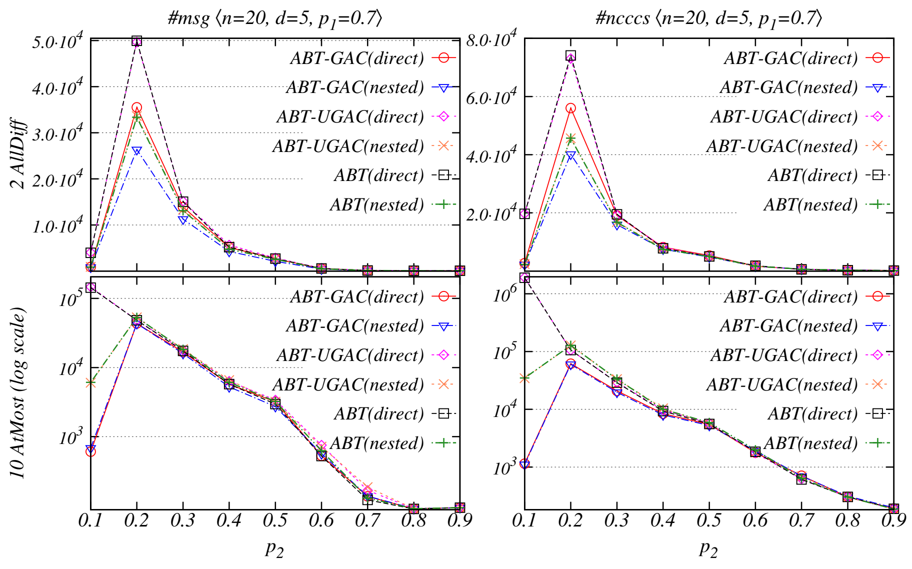
Results are shown in Fig. 2. ABT-GAC(direct) and ABT-GAC(nested) always require fewer messages than other algorithms, achieving up to two orders of magnitude

[3] http://dischoco.sourceforge.net/

[4] ATMOST is not a contractible constraint [2]. To use nested representation, we changed the number of occurrences on nested constraints.



(a) sparse instances



(b) dense instances

Fig. 2: Performance on the AllDiff benchmark and AtMost benchmark (in logarithmic scale).

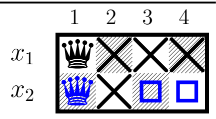
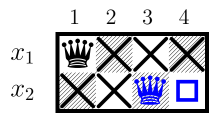
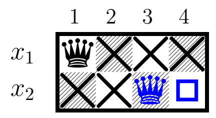
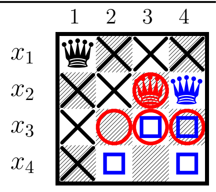
	Constraints	Explanations	Chessboard
direct	$c_1 : \mid x_1 - x_2 \mid \neq 1$ $c_{x_1} : x_1 = 1$	$c_{x_1} \to D_1 = \{1\}$ $c_{x_1} \wedge c_1 \to x_2 \neq 2$	
nested	$c_1 : \mid x_1 - x_2 \mid \neq 1$ c_9^n :AllDiff(x_1, x_2) $c_{x_1} : x_1 = 1$	$c_{x_1} \to D_1 = \{1\}$ $c_{x_1} \wedge c_9^n \to x_2 \neq 1$ $c_{x_1} \wedge c_1 \to x_2 \neq 2$	
binary	$c_1 : \mid x_1 - x_2 \mid \neq 1$ $c_8^b : x_1 \neq x_2$ $c_{x_1} : x_1 = 1$	$c_{x_1} \to D_1 = \{1\}$ $c_{x_1} \wedge c_8^b \to x_2 \neq 1$ $c_{x_1} \wedge c_1 \to x_2 \neq 2$	
full	$c_1 : \mid x_1 - x_2 \mid \neq 1$ $c_4 : \mid x_2 - x_3 \mid \neq 1$ $c_5 : \mid x_2 - x_4 \mid \neq 2$ $c_{x_1} : x_1 = 1$ c_7 : AllDiff(x_1, x_2, x_3, x_4)	$c_{x_1} \to D_1 = \{1\}$ $c_{x_1} \wedge c_7 \to x_2 \neq 1$ $c_{x_1} \wedge c_7 \to x_3 \neq 1$ $c_{x_1} \wedge c_7 \to x_4 \neq 1$ $c_{x_1} \wedge c_1 \to x_2 \neq 2$	

Fig. 3: Agent A_2 running ABT with different representation after receiving the assignment of $x_1 = 1$ when solving the distributed 4-queen (Fig. 1).

improvement in the harder region for the non-contractible AtMost constraint. For $\#ncccs$, the results are similar, except for the sparse instances of AllDiff, where ABT-GAC(direct) and ABT-GAC(nested) are improved by ABT(nested) and ABT-UGAC(nested), and for dense instances of AllDiff, where ABT(nested) and ABT-UGAC(nested) improve ABT-GAC(direct).

4 Maintaining GAC Concurrently using a Full Representation of Global Constraints

In the direct representation each global constraint will be represented by a single constraint. In the nested representation a contractible global constraint will be represented by a set of constraints, which allows concurrent processing and earlier identification of some inconsistencies. However, the filtering in these new constraints is weaker than that of the original constraint (AllDiff for example). More importantly, not all global constraints are contractible.

We now present a new way to represent a global constraint in DisCSPs, which we call the *full* representation. In the full representation, the original constraint is evaluated in the DisCSP by all agents that are involved in it, using their own copies of other agents' domains. As with direct, the full representation is applicable to all global constraints. The motivation is to benefit from the strength of filtering algorithms for global constraints and to do more concurrent computation, to detect unfruitful decisions earlier, and thus decrease the number of messages and $\#ncccs$. Much of this concurrent pruning may be redundant, though, and so we may increase the total amount of work averaged over all agents. We note that the full representation weakens the domain privacy of lower priority agents, but is fair for all agents involved in the constraint.

In the example of Fig. 1, A_1 evaluates the constraints c_1, c_2, c_3 and c_7. A_2 evaluates the constraints c_1, c_4, c_5 and c_7. A_3 evaluates the constraints c_2, c_4, c_6 and c_7. A_4 evaluates the constraints c_3, c_5, c_6 and c_7. Fig. 3 shows the reasoning by agent A_2 after receiving the assignment of A_1 (i.e., $x_1 = 1$) when running ABT with direct, nested, binary and full representation. Once it receives this assignment, A_2 adds the constraint $c_{x_1} : (x_1 = 1)$ to its constraint store then maintains GAC. In direct representation only value 2 is removed from D_2. Thus, A_2 assigns 1 to its variable. In nested and binary representation, two values (1 and 2) are removed from D_2. Thus, A_2 assigns 3 to its variable. In full representation, two values (1 and 2) are removed from D_2, and value 1 from D_3 and D_4. Thus, A_2 tries to assign value 3 to its variable. A_2 again maintains GAC and removes value 3 from D_2 because it has no support in D_3 (Fig. 3 (full), circles). It then assigns 4 to its variable, i.e., $x_2 = 4$.

We also extend ABT-GAC with the propagation of unconditional value deletions from [3], to get ABT-GAC+U. ABT-GAC(full) and ABT-GAC+U(full) inherit the correctness, completeness and termination of ABT(direct) and ABT-UGAC(direct) [3]. The only changes we make are adding redundant copies of constraints and allowing agents to do more powerful correct filtering.

4.1 Theoretical Analysis

We demonstrate that ABT-GAC(full) is sound, complete and terminates.

Theorem 1. *ABT-GAC(full) is sound.*

Proof. (Sketch) When the state of quiescence is reached, all agents know the assignments of all their higher priority neighbours. Thus, any constraint has been successfully checked by the lowest priority agent in its scope when it is fully instantiated. Otherwise, that agent would have tried to change its value and would have either sent an message containing its new value or a no-good, breaking the quiescence.

Theorem 2. *ABT-GAC(full) is complete.*

Proof. All explanation and no-goods are generated by logical inferences from existing constraints. Therefore, an empty no-good cannot be inferred if a solution exists.

Theorem 3. *ABT-GAC(full) terminates.*

Proof. ABT-GAC(full) inherits the termination of ABT [37]. We can prove by induction on the agent ordering that agents can never fall into an infinite loop . First, we can show that agent A_1, never falls into an infinite loop. Then, assuming that all agents higher that an agent A_i ($i > 2$) are in a stable state, we can show that agent A_i never falls into an infinite loop.

Table 1: Performance on hard region of sparse 2 ALLDIFF benchmark ($p_1 = 0.2, p_2 = 0.7$).

$p_2 = 0.7$	$\#msg$		$\#ncccs$		$\#ccs$	
	ABT-GAC	ABT-UGAC	ABT-GAC	ABT-UGAC	ABT-GAC	ABT-UGAC
full	**6 984**	**7 086**	30 439	28 012	203 934	193 403
nested	8 173	8 605	26 099	26 859	170 798	179 706
direct	7 774	8 253	**25 875**	**26 560**	**164 106**	**173 683**
binary	7 857	8 358	28 491	29 499	188 500	198 213

Table 2: Performance on hard regions of instances with 5 ATMOST benchmark .

	$(p_1 = 0.2, p_2 = 0.7)$				$(p_1 = 0.7, p_2 = 0.3)$			
	$\#msg$		$\#ncccs$		$\#msg$		$\#ncccs$	
	GAC	GAC +U	GAC	GAC +U	GAC	GAC +U	GAC	GAC +U
full	7 352	**7 755**	24 847	23 927	**737 854**	**737 590**	2 534 734	2 530 072
nested	8 509	9 153	21 756	22 709	747 080	745 303	1 745 611	1 739 681
direct	**7 288**	7 897	**21 157**	**22 061**	752 748	752 669	**1 737 450**	**1 733 302**

5 Experimental Results

We empirically compare the full representation to direct, nested and binary representations, all implemented within ABT-GAC and ABT-GAC+U. In our experiments, we imposed a cut-off on $\#ncccs$ of 10^9 and a cut-off on $\#msg$ of 10^9. In almost all instances, ABT without explanations (ABT(direct), ABT(nested) and ABT(binary)) exceeds this limit. Moreover, it runs out of memory in many instances. Thus, here we only present results on explained versions (ABT-GAC). For the same reason, we only show results for ABT-GAC+U.

5.1 Uniform Binary Random DisCSPs with Global Constraints

We solved instances of two classes of constraint graphs: sparse graphs $\langle 20, 10, 0.2, p_2\rangle$ and dense graphs $\langle 20, 10, 0.7, p_2\rangle$. We varied the tightness from 0.1 to 0.9 by steps of 0.1. For each pair of fixed density and tightness (p_1, p_2) we report averages over 100 instances. From a binary instance, we generate 4 types of benchmarks: the ALLDIFF, ATMOST, ATLEAST and EXACTLY benchmarks. In the ALLDIFF benchmark, each binary instance includes 2 ALLDIFF constraints, each involving 5 randomly chosen variables. In the ATMOST benchmark, each binary instance includes 5 ATMOST$(3, X, v)$ constraints, each involving from 5 to 7 randomly chosen variables. The value v is randomly chosen from the set of values in domains. In the ATLEAST benchmark, each binary instance includes 10 ATLEAST$(3, X, v)$ constraints, each involving from 5 to 7 randomly chosen variables. The value v is randomly chosen from the set of values in domains. In the EXACTLY benchmark, each binary instance includes 5 EXACTLY$(3, X, v)$ constraints, each involving from 5 to 7 randomly chosen variables. The value v is randomly chosen from the set of values in domains.

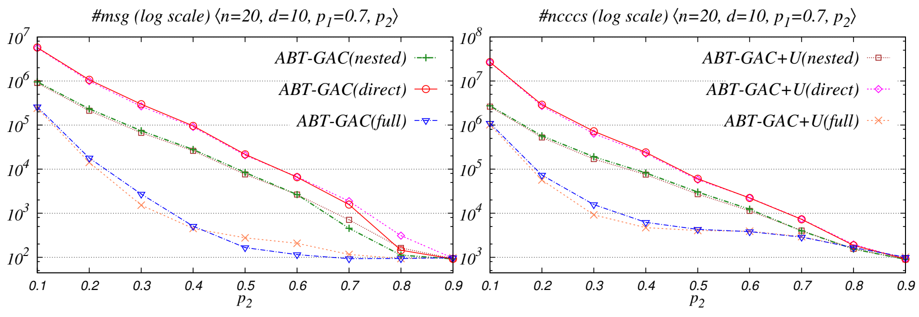


Fig. 4: Performance on dense instances with 10 ATLEAST benchmark.

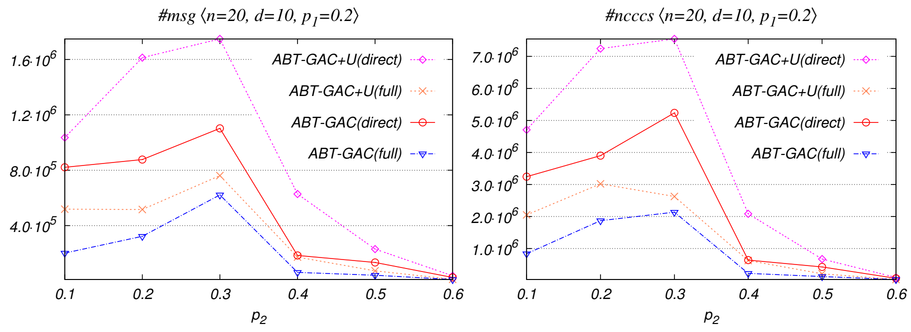


Fig. 5: Performance on sparse instances with 5 EXACTLY benchmark.

For space reasons, we only show a selection of results. In Table 1, for sparse ALLDIFF problems, in the harder region ($p_2 = 0.7$), we see a small improvement in $\#msg$ and a small deterioration in $\#ncccs$ compared to direct, for the full representation over the explained versions of the other representations. In Table 1, we also show the total computational effort (total number of constraint checks by all agents $\#ccs$), and as expected, we show a small increase. For ATMOST problems (Table 2), the three representations are similar in $\#msg$, but full is almost 50% poorer on $\#ncccs$ for dense problems. On dense ATLEAST problems (Fig. 4), we see at least an order of magnitude improvement for the full representation over the nested for both $\#msg$ and $\#ncccs$, while direct is consistently worst. On sparse EXACTLY problems Fig. 5, we gain a two-fold improvement compared to direct representation. ABT-GAC+U deteriorates compared to ABT-GAC. It seems that extra messages needed to propagate the unconditional removals slows the search for both representations.

5.2 Quasi-Groups With Holes

We also evaluate ABT-GAC and ABT-GAC+U on a set of satisfiable balanced quasi-groups with holes (QGWH) instances [26, 6].[5] The set contains 100 different instances.

[5] http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html

Table 3: Performance on Quasi-Groups With Holes problems.

	#instances solved		$\#msg$		$\#ncccs$	
	ABT-GAC	ABT-GAC+U	ABT-GAC	ABT-GAC+U	ABT-GAC	ABT-GAC+U
full	**98**	**99**	**5 882 353**	**5 190 937**	**737 296**	**782 601**
nested	95	**99**	10 495 843	6 852 796	1 413 990	1 087 459
direct	96	**99**	10 044 646	7 060 771	1 971 401	1 380 700
binary	87†	84‡	11 765 454	11 910 842	3 930 123	3 884 403

† ABT-GAC(binary) runs out of memory for 8 instances

‡ ABT-GAC+U(binary) runs out of memory for 11 instances

Each instance contains 106 variables and 30 ALLDIFF constraints, and as before, each variable is controlled by a different agent. Only 71 instances were solved by all algorithms, and the average performance over these instances is presented in Table 3. We see that the binary representation in QGWH performs relatively poorly. The concurrent filtering (i.e., *full*) outperforms all other strategies.

5.3 Distributed Meeting Scheduling Problem

The *distributed meeting scheduling problem* (DMSP) [18, 35] consists of a set of n agents having a personal private calendar and a set of m meetings each taking place in a specified location. Each agent knows the set of the k among m meetings she must attend, and knows the traveling time between the locations where her meetings will be held. The traveling time between two meetings m_i and m_j is denoted by $TT(m_i, m_j)$. The following constraints apply: (i) all agents attending a meeting must agree on when it will occur, (ii) an agent cannot attend two meetings at the same time, (iii) an agent must have enough time to travel from one meeting to the next.

We encode the DMSP in DisCSP as follows. Each DisCSP agent represents a real agent and contains k variables representing the k meetings in which the agent participates. These k meetings are selected randomly among the m meetings. The domain of each variable contains the $d \times h$ slots when a meeting can be scheduled. A slot is one hour long, and there are h slots per day and d days. There is an ALLEQUAL constraint for all variables corresponding to the same meeting in different agents (constraint (i)). There is an *arrival-time* constraint between all variables/meetings belonging to the same agent. The arrival-time constraint between two variables m_i and m_j is $\mid m_i - m_j \mid -duration > TT(m_i, m_j)$, where $duration$ is the duration of every meeting. This arrival-time constraint allows us to express both constraints (ii) and (iii). We place meetings randomly on the nodes of a uniform grid of size $g \times g$ and the traveling time between two adjacent nodes is 1 hour. The traveling time between two meetings equals the Euclidean distance between their locations. To vary the tightness of the arrival-time constraint we vary the size of the spatial grid.

Problems are characterized by $\langle n, m, k, d, h, g\rangle$, where n is the number of agents, m is the number meetings, k is the number of meetings/variables per agent, d is the

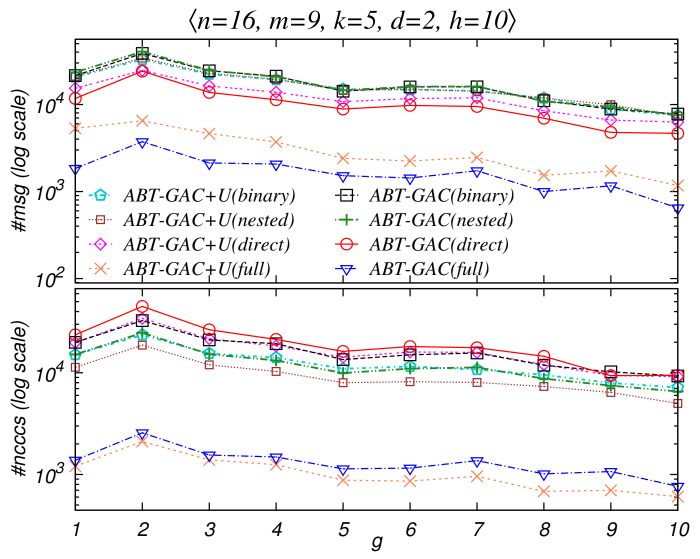


Fig. 6: Performance (log scale) on 16-agent DMSP.

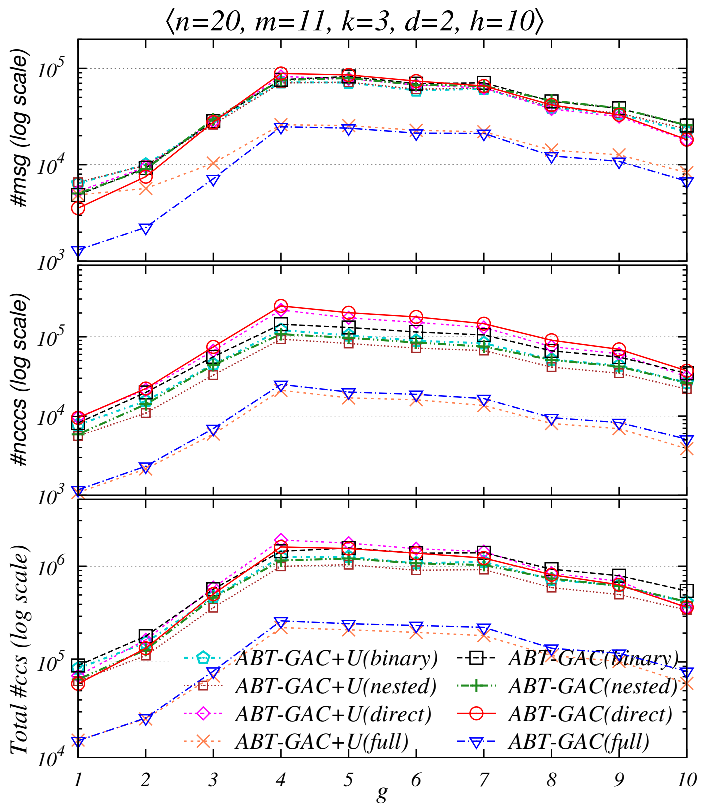


Fig. 7: Performance (log scale) on 20-agent DMSP.

number of days and h is the number of hours per day, and g is the grid size. The duration of each meeting is one hour. In our implementation this encoding is translated into an equivalent formulation where we have k (number of meetings per agent) virtual agents for each real agent. Each virtual agent handles a single variable but $\#msg$ does not take into account messages exchanged between virtual agents belonging to the same real agent [35]. We solved instances for two classes $\langle 20,\ 11,\ 3,\ 2,\ 10,\ g\rangle$ and $\langle 16,\ 9,\ 5,\ 2,\ 10,\ g\rangle$ where we vary g from 1 to 10 by steps of 1. For each g we generated 100 instances.

In Fig. 6, for the 16-agent problems, we see a consistent order of magnitude improvement for the full representation over the other approaches for both $\#msg$ and $\#ncccs$ across all grid sizes. In Fig. 7, for the larger 20 agent problems, but with fewer meetings, we see an improvement in both measures between a factor of 5 and a factor of 10. We also plot in the same figure the total number of constraint checks over all agents. Surprisingly, we see a reduction in total computational effort up to a factor of 5. It appears that the gains from more powerful filtering, more efficient no-goods and the reduced number of messages outweighs the extra effort of the redundant filtering.

6 Conclusion

The power of filtering algorithms for global constraints is one of the main features of the success of constraint programming. In Distributed CSPs, however, global constraints have received little attention, because of the distributed control of the variables and their domains. Recently, a nested representation of contractible global constraints was shown to reduce computational effort and communication. In this paper, we make two contributions to the handling of global constraints in Distributed CSP. First, we show that maintaining GAC using explained constraints significantly improves the performance over the previous approaches. Secondly, we introduce a full representation for any global constraint, allowing every agent to evaluate any constraint it is involved in, and we use this to implement concurrent maintenance of GAC. We demonstrate empirically that concurrent GAC on the full representation offers a further significant improvement in both non-concurrent computation and messaging. This appears to contradict recent results that suggest reducing redundancy in Distributed CSP always improves performance [12, 32]. We also show that for some problems, despite the redundant filtering, we reduce the total computation cost over all the agents.

Future work will focus on extending other DisCSP algorithms to include MAC and then exploiting full concurrent GAC, and on implementing global constraints and full concurrent GAC with distributed dynamic ordering algorithms.

Acknowledgements This work is funded by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

Bibliography

[1] Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., Valls, M.: Sensor networks and distributed csp: communication, computation and complexity. Artif. Intel. 161, 117–147 (2005)

[2] Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog. SICS Research Report (2005)

[3] Bessiere, C., Brito, I., Gutierrez, P., Meseguer, P.: Global constraints in distributed constraint satisfaction and optimization. The Computer Journal (2013)

[4] Bessiere, C., van Hentenryck, P.: To be or not to be ... a global constraint. In: Proceedings of the CP'03. pp. 789–794 (2003)

[5] Bessiere, C., Maestre, A., Brito, I., Meseguer, P.: Asynchronous backtracking without adding links: a new member in the ABT family. Artif. Intel. 161, 7–24 (2005)

[6] Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting Systematic Search by Weighting Constraints. In: Proceedings of ECAI'04. pp. 146–150 (2004)

[7] Brito, I., Meseguer, P.: Asynchronous backtracking for non-binary discsp. In: DCR Workshop at ECAI-2006. DCR'06, Riva di Garda, Italia (2006)

[8] Downing, N., Feydy, T., Stuckey, P.J.: Explaining alldifferent. In: Proceedings of ACSC'12. pp. 115–124. Darlinghurst, Australia, Australia (2012)

[9] Downing, N., Feydy, T., Stuckey, P.J.: Explaining flow-based propagation. In: Proceedings of CPAIOR'12. pp. 146–162 (2012)

[10] Francis, K., Stuckey, P.: Explaining circuit propagation. Constraints 19(1), 1–29 (2014)

[11] Gaudin, E., Jussien, N., Rochart, G.: Implementing explained global constraints. In: Proceedings of the CP'04 Workshop on Constraint Propagation and Implementation (CPAI'04). pp. 61–76. Toronto, Canada, Canada (2004)

[12] Gutierrez, P., Meseguer, P.: Saving redundant messages in bnb-adopt. In: AAAI'10 (2010)

[13] Jussien, N.: The versatility of using explanations within constraint programming. Hdr, Université de Nantes (Sep 2003)

[14] Jussien, N., Barichard, V.: The palm system: explanation-based constraint programming. Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000 pp. 118–133 (2000)

[15] Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Series (1997)

[16] Maher, M.J.: Open contractible global constraints. In: Proceedings of IJCAI'09. pp. 578–583. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2009)

[17] Mailler, R., Lesser, V.R.: Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. JAIR 25(1), 529–576 (2006)

[18] Meisels, A., Lavee, O.: Using additional information in DisCSP search. In: Proceedings of DCR'04 (2004)

[19] Meisels, A., Zivan, R.: Asynchronous Forward-checking for DisCSPs. Constraints 12(1), 131–150 (2007)

[20] Petcu, A., Faltings, B.: A Value Ordering Heuristic for Distributed Resource Allocation. In: Proceedings of Joint Annual Workshop of ERCIM/CoLogNet on CSCLP'04. pp. 86–97 (2004)

[21] Petcu, A., Faltings, B.: DPOP: A Scalable Method for Multiagent Constraint Optimization. In: Proceedings of IJCAI'05. pp. 266–271 (2005)

[22] Régin, J.C.: A filtering algorithm for constraints of difference in csps. In: Proceedings of AAAI'94. pp. 362–367 (1994)

[23] Régin, J.C.: Global constraints: A survey. In: van Hentenryck, P., Milano, M. (eds.) Hybrid Optimization, vol. 45, pp. 63–134. Springer New York (2011)

[24] Rochart, G.: Explanations for global constraints. In: Proceedings of the CP'03. pp. 993–993 (2003)

[25] Rochart, G.: Explications et programmation par contraintes avancée. Ph.D. thesis, Nantes University, France (2005)

[26] Roussel, O., Lecoutre, C.: Xml representation of constraint networks: Format xcsp 2.1. CoRR (2009)

[27] Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints 16(3), 250–282 (Jul 2011)

[28] Silaghi, M.C.: Generalized Dynamic Ordering for Asynchronous Backtracking on DisCSPs. In: Proceedings of DCR'06 (2006)

[29] Silaghi, M.C., Faltings, B.: Asynchronous aggregation and consistency in distributed constraint satisfaction. Artif. Intel. 161, 25–53 (2005)

[30] Silaghi, M.C., Sam-Haroud, D., Faltings, B.: Asynchronous Search With Aggregations. In: Proceedings of AAAI'00/IAAI'00. pp. 917–922 (2000)

[31] Wahbi, M.: Algorithms and Ordering Heuristics for Distributed Constraint Satisfaction Problems. John Wiley & Sons, Inc. (2013)

[32] Wahbi, M., Ezzahir, R., Bessiere, C.: Asynchronous Forward Bounding Revisited. In: Proceedings of CP'13. pp. 708–723. Uppsala, Sweden (2013)

[33] Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: DisChoco 2: A Platform for Distributed Constraint Reasoning. In: Proceedings of workshop on DCR'11. pp. 112–121 (2011), http://dischoco.sourceforge.net/

[34] Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: Maintaining Arc Consistency Asynchronously in Synchronous Distributed Search. In: Proceedings of ICTAI'12. pp. 33–40. Athens, Greece (November 2012)

[35] Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: Nogood-Based Asynchronous Forward-Checking Algorithms. Constraints 18(3), 404–433 (2013)

[36] Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: Distributed constraint satisfaction for formalizing distributed problem solving. In: Proceedings of 12th IEEE Int'l Conf. Distributed Computing Systems. pp. 614–621 (1992)

[37] Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. IEEE Trans. on Knowledge and Data Engineering 10, 673–685 (1998)

[38] Zivan, R., Meisels, A.: Message delay and DisCSP search algorithms. Annals of Mathematics and Artificial Intelligence 46(4), 415–439 (2006)

[39] Zivan, R., Zazone, M., Meisels, A.: Min-Domain Retroactive Ordering for Asynchronous Backtracking. Constraints 14(2), 177–198 (2009)