

Nogood-based Asynchronous Distributed Optimization (ADOPT-ng)

Marius Silaghi
Florida Institute of Technology
msilaghi@fit.edu

Makoto Yokoo
Kyushu University
yokoo@is.kyushu-u.ac.jp

ABSTRACT

This work proposes an asynchronous algorithm for solving Distributed Constraint Optimization problems (DCOPs) using a new kind of nogoods, namely valued nogoods. The proposed technique is an extension of the asynchronous distributed optimization (ADOPT) where valued nogoods enable more flexible reasoning, leading to important speed-up. Valued nogoods are an extension of classic nogoods that associates each nogood with a threshold and optionally with a set of references to culprit constraints.

DCOPs have been shown to have very elegant distributed solutions, such as ADOPT, distributed asynchronous overlay (DisAO), or DPOP. These algorithms are typically tuned to minimize the longest causal chain of messages, as a measure of how the algorithms will scale for systems with remote agents (with large latency in communication). Among the mentioned techniques, DPOP performs very well for the chosen metric (requiring a number of such sequential messages linear in the number of agents), but in general has exponential space requirements. DisAO and ADOPT have the advantage of needing only polynomial space. ADOPT has the property of maintaining the initial distribution of the problem. ADOPT needs a preprocessing step consisting of computing a depth first search (DFS) tree on the agent graph. We show that valued nogoods reduce the practical importance/need of this preprocessing since independent subproblem are now dynamically detected and exploited.

1. INTRODUCTION

Distributed Constraint Optimization (DCOP) is a formalism that can model naturally distributed problems. These are problems where agents try to find assignments to a set of variables subjects to constraints. The natural distribution comes from the assumption that only a subset of the agents has knowledge of each given constraint. Nevertheless, in DCOP it is assumed that agents try to maximize their cumulated satisfaction by the chosen solution. This is different from other related formalisms where agents try to maximize

the satisfaction of the least satisfied among them [21].

Several synchronous and asynchronous distributed algorithms have been proposed for distributedly solving DCOPs. Since a DCOPs can be viewed as a distributed version of the common centralized Valued Constraint Satisfaction Problems (VCSPs), it is normal that successful techniques for VCSPs were ported to DCOPs. However the effectiveness of such techniques has to be evaluated from a different perspective (and different measures) as imposed by the new requirements. Typically research has focused on techniques in which reluctance is manifested towards modifications to the distribution of the problem (modification accepted only when some reasoning infers it is unavoidable for guaranteeing to reach solutions). This criteria is largely believed valuable and adaptable for large, open, and/or dynamic distributed problems. It is also perceived as an alternative approach to problems with privacy requirements [14, 20, 25, 16].

A synchronous algorithm, synchronous branch and bound, was the first known distributed algorithm for solving DCOPs [7]. Stochastic versions have also been proposed [26]. From the point of view of efficiency, a distributed algorithm for solving DCOPs is typically evaluated with regard to applications to agents on the Internet, namely where latency in communication is significantly higher than local computations. A measure representing well this assumption is given by the number of cycles of a simulator that in turn lets each agent process all the messages that it receives [22]. Within the mentioned assumption, this measure is equivalent for real solvers to the longest causal chain of sequential messages, as used in [17].

From the point of view of this measure, a very efficient currently existing DCOP solver is DPOP [11], which is linear in the number of variables. However, in general that algorithm has message sizes and local computation costs that are exponential in the induced width of a chosen depth search tree of the constraint graph of the problem, clearly invalidating the assumptions that lead to the acceptance of the number of cycles as efficiency measure.

Two other algorithms competing as efficient solvers of DCOPs, are the asynchronous distributed optimization (ADOPT) and the distributed asynchronous overlay (DisAO). DisAO works by incrementally joining the subproblems owned by agents found in conflict. ADOPT implements a parallel version of A* [15]. While DisAO is typically criticized for its extensive abandon of the maintenance of the natural distributedness of the problem at the first conflict (and expensive local computations invalidating the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

above assumptions like DPOP [4, 8, 1]), ADOPT can be criticized for its strict message pattern that only provides reduced reasoning opportunities. ADOPT also works only on special orderings on agents, namely dictated by some Depth First Search tree on the constraint graph.

It is easy to construct huge problems whose constraint graphs are forests and that are easily solved by DPOP (in linear time), but unsolvable with the other known algorithms. It is also easy to construct relatively small problems whose constraint graph is full and therefore require unacceptable (exponential) space with DPOP, while being easily solvable with algorithms like ADOPT, e.g. for the trivial case where all tuples are optimal with cost zero.

In this work we address the aforementioned critiques of ADOPT showing that it is possible to define a message scheme based on a new type of nogoods, called *valued nogoods*, that not only virtually eliminates the need of precomputing a DFS tree of the constraint graph, but also leads to significant improvement in efficiency. Nogoods are at the basis of much flexibility in asynchronous algorithms. A nogood specifies a set of assignments that conflict constraints [19]. A basic version of the valued nogoods consists in associating each nogood to a threshold, namely a cost limit violated due to the assignments of the nogood. It is significant to note that the valued nogoods lead to efficiency improvements even if used in conjunction with a DFS tree, instead of the less semantically explicit cost messages of ADOPT. We find that a version of valued nogoods that is associated with a list of culprits constraints [12] produces additional important improvements. Each of these incremental concepts and improvements is described in the following sections.

2. PRIOR STATE OF THE ART

2.1 DFS-trees

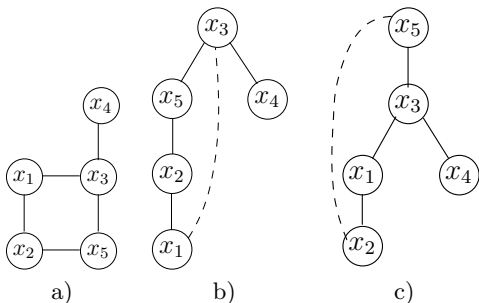


Figure 1: For a DCOP with primal graph depicted in (a), two possible DFS trees (pseudotrees) are (b) and (c). Interrupted lines show constraint graph neighboring relations not in the DFS tree.

The primal graph of a DCOP is the graph having the variables as nodes and having an arc for each pair of variables linked by a constraint [5]. A Depth First Search (DFS) tree associated to a DCOP is a spanning tree generated by the arcs used for visiting once each node during some depth first traversal of its primal graph. DFS trees were first successfully used for Distributed Constraint problems in [3]. The property exploited there is that separate branches of the DFS-tree are completely independent once the assignments

of common ancestors are decided. Two examples of DFS trees for a DCOP primal graph are shown in Figure 1.

Nodes directly connected to a node in a primal graph are said to be its *neighbors*. In Figure 1.a, the neighbors of x_3 are $\{x_1, x_5, x_4\}$. The *ancestors* of a node are the nodes on the path between it and the root of the DFS tree, inclusively. In Figure 1.b, $\{x_5, x_3\}$ are ancestors of x_2 . x_3 has no ancestors.

2.2 ADOPT

The Adopt [9] is an asynchronous complete DCOP algorithm, which is guaranteed to find an optimal solution. Here, we only show a brief description of the ADOPT. Please consult [9] for the detail. First, the ADOPT organize agents into a Depth-first Search (DFS) tree, in which constraints are allowed between a variable and any of its ancestors or descendants, but not between variables in separate sub-trees.

ADOPT uses three kinds of messages: VALUE, COST, and THRESHOLD. A VALUE message communicates the assignment of a variable from ancestors to descendants who share constraints with the sender. When the algorithm starts, each agent takes a random value for its variable and sends appropriate VALUE messages. A COST message is sent from a child to its parent, which indicates the estimated lower-bound of the cost of the subtree rooted at the child. Since communication is asynchronous, a cost message contains a context, i.e., a list of the value assignments of the ancestors. The THRESHOLD message is introduced to improve the search efficiency. An agent tries to assign its value so that the estimated cost is lower than the given threshold communicated by the THRESHOLD message from its parent. Initially, the threshold is 0. When the estimated cost is higher than the given threshold, the agent opportunistically switches its value assignment to another value that has the smallest estimated cost. Initially, the estimated cost is 0. Therefore, an unexplored assignment has an estimated cost 0. A cost message also contains the information of the upper-bound of the cost of the subtree, i.e., the actual cost of the subtree. When the upper-bound and the lower-bound meet at the root agent, then a globally optimal solution has been found and the algorithm is terminated.

3. DISTRIBUTED VALUED CSPS

Constraint Satisfaction Problems (CSPs) are described by a set X of variables and a set of constraints on the possible combinations of assignments to these variables with values from their domains.

DEFINITION 1 (DisVCSP/DCOP). A *Distributed Valued CSP (DisVCSP)*, also known as *distributed constraint optimization problem (DCOP)*, is defined by a set of agents A_1, A_2, \dots, A_n , a set X of variables, x_1, x_2, \dots, x_n , and a set of functions $f_1, f_2, \dots, f_i, \dots, f_n$, $f_i : X_i \rightarrow \mathbb{R}$, $X_i \subseteq X$, where only A_i knows f_i .

The problem is to find $\arg\min_x \sum_{i=1}^n f_i(x|_{X_i})$. We assume that x_i can only take values from a domain $D_i = \{1, \dots, d\}$.

For simplification and without loss of generality one typically assumes that $X_i \subseteq \{x_1, \dots, x_i\}$

Our idea can be easily applied to general valued CSPs.

4. COST OF NOGOODS

Previous flexible algorithms for solving distributed constraint satisfaction problems exploit the inference power

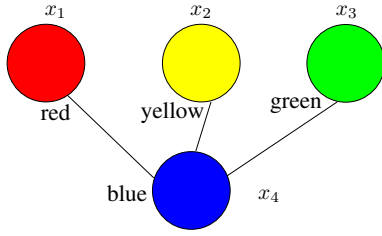


Figure 2: MIN resolution on valued global nogoods

of nogoods (e.g., ABT, AWC, ABTR [22, 23, 18])¹. A nogood $\neg N$ stands for a set N of assignments that was proven impossible, by inference using constraints. If $N = (\langle x_1, v_1 \rangle, \dots, \langle x_t, v_t \rangle)$ where $v_i \in D_i$, then we denote by \overline{N} the set of variables assigned in N , $\overline{N} = \{x_1, \dots, x_t\}$.

In order to apply nogood-based algorithms to DisVCSP, we redefine the notion of nogoods as follows. First we attach a value to each nogood obtaining a *valued global nogood*.

DEFINITION 2 (VALUED GLOBAL NOGOOD). A *valued global nogood* has the form $[c, N]$, and specifies that the (global) problem has cost at least c , given the set of assignments, N , for distinct variables.

Given a valued global nogood $[c, (\langle x_1, v_1 \rangle, \dots, \langle x_t, v_t \rangle)]$, one can infer a *global cost assessment (GCA)* for the value v_t from the domain of x_t given the assignments $S = \langle x_1, v_1 \rangle, \dots, \langle x_{t-1}, v_{t-1} \rangle$. This GCA is denoted (v_t, c, S) , and is semantically equivalent to an applied global value nogood, (i.e. the inference):

$$(\langle x_1, v_1 \rangle, \dots, \langle x_{t-1}, v_{t-1} \rangle) \rightarrow (\langle x_t, v_t \rangle \text{ has cost } c).$$

We will denote with $vn2ca(N, k, v)$ the procedure that takes as parameter a valued global nogood $[c, N]$ and returns a GCA for the assignment of $\langle x_k, v \rangle$.

REMARK 1. Given a valued global nogood $[c, N]$, one can infer the (GCA) (v, c, N) for any value v from the domain of any variable x , where x is not assigned in N , i.e., $x \notin \overline{N}$.

E.g., if A_3 knows the valued global nogood $[10, \{(x_1, r), (x_2, y)\}]$, then it can infer for the value r of x_3 the GCA $(r, 10, \{(x_1, r), (x_2, y)\})$.

PROPOSITION 1 (MIN-RESOLUTION). From a set $\{(v, c_v, S_v)\}$ containing exactly one GCA for each value v in the domain of variable x_i and $\forall k, j$ assignments for $\overline{S_k} \cap \overline{S_j}$ are identical in both S_k and S_j of a minimization VCSP, one can resolve a new valued global nogood: $[\min_v c_v, \cup_v S_v]$.

Example 4.1 For the graph coloring problem in Figure 2 (costs are not shown), x_1 is colored red (r), x_2 yellow (y) and x_3 green (g). Assume that the following valued global nogoods are known for each of the values $\{r, y, g\}$ of x_4 :

$$(r): [10, \{(x_1, r), (x_4, r)\}], \text{ obtaining for } x_4 \text{ the GCA } (r, 10, \{(x_1, r)\})$$

$$(y): [8, \{(x_2, y), (x_4, y)\}], \text{ obtaining for } x_4 \text{ the GCA } (y, 8, \{(x_2, y)\})$$

¹Other algorithms, like AAS, exploited generalizes nogoods (i.e., extensions of nogoods to sets of values for a variable), and the extension of the work here for that case is pointed to in [12]

$$(g): [7, \{(x_3, g), (x_4, g)\}], \text{ obtaining for } x_4 \text{ the GCA } (g, 7, \{(x_3, g)\})$$

By min-resolution on these GCAs one obtains the valued global nogood $[7, \{(x_1, r), (x_2, y), (x_3, g)\}]$, meaning that given the coloring of the first 3 nodes there is no solution with (global) cost lower than 7.

Min-resolution can be applied on valued global nogoods:

COROLLARY 1.1. From a set \mathcal{S} of no-goods, such that for each $[c_v, S_v]$ in \mathcal{S} $\langle x_i, v, \in \rangle S_v$, containing exactly one global valued nogood for each value v in the domain of variable x_i of a minimization VCSP, one can resolve a new valued global nogood: $[\min_v c_v, \cup_v (S_v \setminus \langle x_i, v \rangle)]$.

REMARK 2 (DFS SUBTREES). Given two GCAs (v, c'_v, S'_v) and (v, c''_v, S''_v) for a value v in the domain of variable x_i of a minimization VCSP, if one knows that the two GCAs are inferred from different constraints then one can infer a new GCA: $\langle c'_v + c''_v, S'_v \cup S''_v \rangle$. This is similar to what ADOPT does to combine cost messages coming from disjoint problem subtrees [10, 3].

This powerful reasoning can be applied when combining a nogood obtained from the local constraints with a valued nogood received from other agents (and obtained solely by inferred on other agents' constraints). When a DFS tree of the constraint graph is used for constraining the message pattern as in ADOPT, this powerful inference applies, too.

The question is how to determine that the two GCAs are inferred from different constraints, in a more general setting. We introduce tagging of cost assessments with the identifiers of the constraints used to infer them.

DEFINITION 3 (SRC). A set of references to constraints is a set of identifiers, each of them for a distinct constraint.

Note that several constraints of a given problem description can be composed in one constraint (in a different description of the same problem).

REMARK 3. For privacy, a constraint can be represented by several constraint references and several constraints of an agent can be represented by a single constraint reference.

Now we generalize the concept of *valued global nogood* into the concept of *valued nogood*.

DEFINITION 4 (VALUED NOGOOD). A *valued nogood* has the form $[SRC, c, N]$ where SRC is a set of references to constraints having cost at least c , given a set of assignments, N , for distinct variables.

Valued nogoods are generalizations of valued global nogoods. Namely, valued global nogoods are valued nogoods whose SRCs contain the references of all the constraints of the problem.

Once we decide that a nogood $[SRC, c, (\langle x_1, v_1 \rangle, \dots, \langle x_i, v_i \rangle)]$ will be applied to a certain variable x_i , we obtain a cost assessment tagged with the set of references to constraints SRC^2 , denoted $(SRC, v_i, c, (\langle x_1, v_1 \rangle, \dots, \langle x_{i-1}, v_{i-1} \rangle))$.

²This is called a *valued conflict list* in [12]

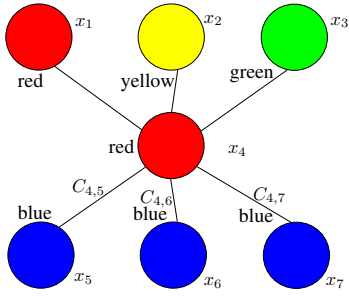


Figure 3: SUM-inference resolution on CAs

DEFINITION 5 (COST ASSESSMENT (CA)). A cost assessment of variable x_i has the form (SRC, v, c, N) where SRC is a set of references to constraints having cost with lower bound c , given a set of assignments, N , for distinct variables, and some value v in the domain of x_i .

As for valued nogoods and valued global nogoods, cost assessments are generalizations of global cost assessments.

REMARK 4. Given a valued nogood $[SRC, c, N]$, one can infer the cost assessment (CA) (SRC, v, c, N) for any value v from the domain of any variable x , where x is not assigned in N , i.e., $x \notin N$.

E.g., if A_6 knows the valued nogood $[\{C_{4,7}\}, 10, \{(x_2, y), (x_4, r)\}]$, then it can infer for the value b of x_6 the CA $(\{C_{4,7}\}, b, 10, \{(x_2, y), (x_4, r)\})$.

We can now detect and perform the desired powerful reasoning on valued nogoods and/or CAs coming from disjoint subtrees, mentioned at Remark 2.

PROPOSITION 2 (SUM-INFERENCE). A set of cost assessments for the value v of some variable, (SRC_i, v, c_i, N_i) where $\forall i, j : i \neq j \Rightarrow SRC_i \cap SRC_j = \emptyset$, and the assignment of any variable x_k is identical in any N_i where x_k is present, can be combined into a new cost assessment. The obtained cost assessment is (SRC, v, c, N) such that $SRC = \cup_i SRC_i$, $c = \sum_i (c_i)$, and $N = \cup_i N_i$.

Example 4.2 For the graph coloring problem in Figure 3, x_1 is colored red, x_2 yellow, x_3 green, and x_4 red. Assume that the following valued nogoods are known for (x_4, r) :

- $[\{C_{4,5}\}, 5, \{(x_2, y), (x_4, r)\}]$ obtaining CA $(\{C_{4,5}\}, r, 5, \{(x_2, y)\})$
- $[\{C_{4,6}\}, 7, \{(x_1, r), (x_4, r)\}]$ obtaining CA $(\{C_{4,6}\}, r, 7, \{(x_1, r)\})$
- $[\{C_{4,7}\}, 9, \{(x_2, y), (x_4, r)\}]$ obtaining CA $(\{C_{4,7}\}, r, 9, \{(x_2, y)\})$

Due to x_4 's constraint with X_1 , one can infer for (x_4, r) the following valued nogood:

- $[\{C_{1,4}\}, 10, \{(x_1, r), (x_4, r)\}]$ obtaining CA $(\{C_{1,4}\}, r, 10, \{(x_1, r)\})$

Then, by sum-inference on these CAs one obtains for x_4 the CA $[\{C_{1,4}, C_{4,5}, C_{4,6}, C_{4,7}\}, r, 31, \{(x_1, r), (x_2, y)\}]$, meaning that given the coloring of the first 2 nodes, coloring x_4 in red leads to a cost of at least 31 for the constraints $\{C_{1,4}, C_{4,5}, C_{4,6}, C_{4,7}\}$.

REMARK 5 (SUM-INFERENCE FOR VALUED NOGOODS). Sum inference can be similarly applied to any set of valued nogoods with disjoint SRCs and compatible assignments. The result of combining nogoods $[SRC_i, c_i, S_i]$ is $[\cup_i SRC_i, \sum_i c_i, \cup_i S_i]$. This can also be applied when assignments are generalized to sets [12].

The min-resolution proposed for GCAs translates straightforwardly for CAs as follows.

PROPOSITION 3 (MIN-RESOLUTION). A set of cost assessments for x_i , (SRC_i, v_i, c_i, N_i) where $\cup_i \{v_i\}$ cover the whole domain of x_i and $\forall k, j$ assignments for $\overline{N_k} \cap \overline{N_j}$ are identical in both N_k and N_j , can be combined into a new valued nogood. The obtained valued nogood is $[SRC, c, N]$ such that $SRC = \cup_i SRC_i$, $c = \min_i (c_i)$ and $N = \cup_i N_i$.

Example 4.3 For the graph coloring problem in Figure 2, x_1 is colored red, x_2 yellow and x_3 green. Assume that the following valued nogoods are known for the values of x_4 :

- (r) : $[\{C_{1,4}\}, 10, \{(x_1, r), (x_4, r)\}]$ obtaining CA $(\{C_{1,4}\}, r, 10, \{(x_1, r)\})$
- (y) : $[\{C_{2,4}\}, 8, \{(x_2, y), (x_4, y)\}]$ obtaining CA $(\{C_{2,4}\}, y, 8, \{(x_2, y)\})$
- (g) : $[\{C_{3,4}\}, 7, \{(x_3, g), (x_4, g)\}]$ obtaining CA $(\{C_{3,4}\}, g, 7, \{(x_3, g)\})$

By min-resolution on these CAs one obtains the valued global nogood $[\{C_{1,4}, C_{2,4}, C_{3,4}\}, 7, \{(x_1, r), (x_2, y), (x_3, g)\}]$, meaning that given the coloring of the first 3 nodes there is no solution with cost lower than 7 for the constraints $\{C_{1,4}, C_{2,4}, C_{3,4}\}$.

As with valued global nogoods, the min-resolution could be applied directly on valued nogoods:

COROLLARY 3.1 (MIN-RESOLUTION ON VALUED NOGOODS). From a set $\{[SRC_v, c_v, S_v] \mid (x_i, v) \in S_v\}$ containing exactly one valued nogood for each value v in the domain of variable x_i of a minimization VCSP, one can resolve a new valued nogood: $[\cup_v SRC_v, \min_v c_v, \cup_v (S_v \setminus \{x_i, v\})]$.

5. ADOPT WITH NOGOODS

We will now present a distributed optimization algorithm using valued nogoods, to maximize the efficiency of reasoning by exploiting increased flexibility. The algorithm can be seen as an extension of both ADOPT and ABT, and will be denoted Asynchronous Distributed OPTimization with valued nogoods (ADOPT-ng).

Like in ABT, agents communicate with **ok?** messages proposing new assignments of sender's variable, **nogood** messages announcing a nogood, and **add-link** messages announcing interest in a variable. Like in ADOPT, agents can also use **threshold** messages, but their content can be included in **ok?** messages.

For simplicity we assume in this algorithm that the communication channels are FIFO. Addition of counters to proposed assignments and nogoods can help to remove this requirement with minimal additional changes (i.e., older assignments and older nogoods for the currently proposed value are discarded).

5.1 Data Structures

Each agent A_i stores its agent view (received assignments), and its outgoing links (agents of lower priority than A_i having constraints on x_i). Instantiations may be tagged with counters. To manage nogoods and CAs, A_i uses matrices $l[1..d]$, $h[1..d]$, $ca[1..d][i+1..n]$, $lvn[1..i][i..n]$, $lr[i+1..n]$ and $lastSent[1..i-1]$ where d is the domain size for x_i . crt_val is the current value A_i proposes for x_i .

- $l[k]$ stores a CA for $x_i = k$, that is inferred solely from the constraints between x_i and prior variables.
- $ca[k][j]$ stores a CA for $x_i = k$, that is obtained from valued nogoods received from A_j .
- $lvn[k][j]$ stores the last valued nogood for variables with higher and equal priority than k and that is received from A_j , $j > i$. $lvn[k][i]$ stores nogoods coming via **threshold/ok?** messages.
- $lr[k]$ stores the last valued nogood received from A_k .
- $h[k]$ stores a CA for $x_i = k$ that is inferred from $ca[k][j]$, $lr[t]$, $lvn[t][j]$, and $l[k]$, for all j and t . Before inference, the valued nogoods in $lvn[t][j]$ need to be first translated into a CAs as described in Remark 4.
- $lastSent[k]$ stores the last valued nogood sent to A_k .

5.2 ADOPT with valued nogoods

The pseudocode for ADOPT-ng is given in Algorithm 1. To extract the cost of a CA we introduce the function $cost()$, $cost((SRC, T, c, v))$ returns c . The $min_resolution(j)$ function applies the min-resolution over all values of the current agent, but using only CAs having no assignment from agents with lower priority than A_j (e.g., not using $lvn[t][k]$ for $t > j$). The $sum_inference()$ function used in Algorithm 1 applies the sum-inference on its parameters whenever this is possible (detects disjoint SRCs), otherwise selects the nogood with highest threshold or whose lowest priority assignment has the highest priority (this has been previously used in [2, 18]). Function $vn2ca$ transforms a valued nogood in a cost assesment for x_i . Its inverse is function $ca2vn$. Function $target(N)$ gives the index of the lowest priority variable present in the nogood N . Like with file expansion, “*” in an index of a matrix means the set obtained for all possible values of that index (e.g., $lvn[*][t]$ stands for $\{lvn[k][t] \mid \forall k\}$).

An agent A_i stores several nogoods (CAs) coming from the same source A_t and applicable to the same value v (namely the one at $ca[v][t]$, all those at $lvn[k][t]$ for any k , and $lr[t]$ when v is crt_val).

REMARK 6. *The order of combining CAs matters. Notation $lr[t]_v$ stands for $vn2ca(lr[t])$ when $lr[t]$'s value for x_i is v , and is not used otherwise. To compute $h[v]$:*

1. a) When maintaining DFS trees, for a value v , CAs are combined for each DFS subtree s :
 $tmp[v][s] = sum_inference_{t \in s}(ca[v][t], lr[t]_v, lvn[*][t]);$
 b) Else, CAs coming from each agent A_t are combined:
 $tmp[v][t] = sum_inference(ca[v][t], lr[t]_v, lvn[*][t]);$
2. CAs from step 1 (a or b) are combined:
 $h[v] = sum_inference(tmp[v][*]);$
3. Add $l[v]$: $h[v] = sum_inference(h[v], l[v]);$

4. Add threshold: $h[v] = sum_inference(h[v], lvn[*][i])$

In ADOPT-ng agents are totally ordered, A_1 having the highest priority. Each agent A_i starts by calling the $init()$ procedure, which initializes its l with valued nogoods inferred from local (unary) constraints. It assigns x_i to a value with minimal local cost, crt_val , announcing the assignment to lower priority agents in outgoing-links. The agents answer any received message with the corresponding procedure: “**when receive ok?**”, “**when receive nogood**”, and “**when receive add-link**”.

When a new assignment is learned from **ok?** or **nogood** messages, valued nogoods based on older assignments for the same variables are discarded and the l vector is updated. Received nogoods are stored in matrices ca , lr and lvn . The vector h is updated on any modification of l , lvn , lr or ca . A_i always sets its crt_val to the index with the lowest CA threshold in vector h (preferring the previous assignment). On each change to the vector h , its values are combined by min-resolution to generate new nogoods for each higher priority agent (or ancestor, in versions using DFS trees). The generation and use of multiple nogoods at once is already shown useful for the constraint satisfaction case in [24].

The threshold valued nogood tnv delivered with **ok?** messages sets a common cost on all values of the receiver (see Remark 4), effectively setting a threshold on costs below which the receiver does not change its value. This achieves the effect of THRESHOLD messages in ADOPT.

Intuitively, the convergence of ADOPT-ng can be noticed from the fact that valued nogoods can only monotonically increase valuation for each subset of the search space, and therefore has to terminate since such valuations can be covered by a finite number of values. If agents A_j , $j < i$ no longer change their assignments, valued nogoods can only monotonically increase at A_i for each value in D_i : nogoods’ thresholds can only increase since they only change by sum-inference.

LEMMA 1. *ADOPT-ng terminates in finite time.*

PROOF. It follows immediately by induction for an increasingly growing suffix of the list of agents (in the order used by the algorithm), assuming the other agents reach quiescence.

The first step of the induction (for the last agent) follows from the fact that the last agent terminates in one step if the previous agents do not change their assignments.

Assuming that the previous induction assertion is true for any suffix of k agents. Let us prove that it is also true for a suffix of $k+1$ agents: For each assignment of the agent A_{n-k} , the remaining k agents will reach quiescence, according to the assumption of the induction step, or the assignments CA cost increases. After values are proposed in turn and the smallest threshold reaches its highest estimate, agent A_{n-k} will select the best value and reaches quiescence. The other agents reach quiescence according to the induction step. \square

LEMMA 2. *The last valued nogood sent by each agent integrates additively the non-zero costs of the constraints of all of its successors.*

PROOF. At quiescence, each agent A_k has received the valued nogoods describing the costs of each of its successors, in the list given by the used ordering on agents (or descendants in the DFS tree when a DFS tree is maintained).

```

procedure init do
   $h[v] := l[v] := \text{initialize CAs from unary constraints};$ 
   $\text{crt\_val} = \text{argmin}_v(\text{cost}(h[v]));$ 
  send ok?( $\langle x_i, \text{crt\_val} \rangle, \emptyset$ ) to all agents in outgoing-links;
when receive ok?( $\langle x_j, v_j \rangle, \text{tvn}$ ) do
  integrate( $\langle x_j, v_j \rangle$ );
  if (tvn no-null and has no old assignment) then
     $k := \text{target}(\text{tvn});$  // threshold tvn as common cost;
     $\text{lvn}[k][i] := \text{sum-inference}(\text{tvn}, \text{lvn}[k][i]);$ 
  check-agent-view();
when receive add-link( $\langle x_j, v_j \rangle, A_j$ ) do
  add  $A_j$  to outgoing-links;
  if ( $\langle x_j, v_j \rangle$ ) is old, send new assignment to  $A_j$ ;
when receive nogood(rvn, t) from  $A_i$  do
  foreach new assignment a of a linked variable  $x_j$  in  $rvn$  do
    integrate(a); // counters show newer assignment;
  if (an assignment in  $rvn$  is outdated) then
    if (some new assignment was integrated now) then
      check-agent-view();
    return;
  foreach assignment a of a non-linked variable  $x_j$  in  $rvn$  do
    send add-link(a) to  $A_j$ ;
  if ( $(j := \text{target}(rvn)) == i$ ) then
     $\text{vn2ca}(rvn, i) \rightarrow$  CA rca for a value v of  $x_i$ ;
     $\text{ca}[v][t] := \text{sum-inference}(rca, \text{ca}[v][t]);$ 
     $lr[t] := rca;$ 
  else
     $\text{lvn}[j][t] := rvn;$ 
  check-agent-view();
procedure check-agent-view() do
  for every ( $v \in D_i$ ) update  $l[v]$  and recompute  $h[v]$ ;
  for every  $A_j$  with higher priority than  $A_i$  (respectively ancestor in the DFS tree, when one is maintained) do
    if (h has non-null cost CA for all values of  $D_i$ ) then
       $\text{vn} := \text{min\_resolution}(j);$ 
      if ( $\text{vn} \neq \text{lastSent}[j]$ ) then
        send nogood( $\text{vn}, i$ ) to  $A_j$ ;
   $\text{crt\_val} = \text{argmin}_v(\text{cost}(h[v]));$ 
  if (crt_val changed) then
    send ok?( $\langle x_i, \text{crt\_val} \rangle,$ 
       $\text{ca2vn}(\text{sum-inference}(\text{lvn}[*][k], \text{ca}[\text{crt\_val}][k]))$ 
      to each  $A_k$  in outgoing_links;
procedure integrate( $\langle x_j, v_j \rangle$ ) do
  discard CAs and nogoods in ca, lvn, and lr that are based on other values for  $x_j$ ;
  store  $\langle x_j, v_j \rangle$  in agent view;

```

Algorithm 1: Procedures of A_i in ADOPT-ng

The lemma results by induction for an increasingly growing suffix of the list of agents (in the order used by the algorithm): It is obvious for the last agent.

Assuming that it is true for the agent A_k , it follows that it is also true for agent A_{k-1} since adding A_{k-1} 's local cost

to the cost received from A_k will be higher (or equal when removing zero costs) to the result of adding A_{k-1} 's local cost to one from any successor of A_k . Respecting the order in Remark 6 guarantees this value is obtained. Therefore the sum between the local cost and the last valued nogood coming from A_k defines the last valued nogood sent by A_{k-1} . \square

THEOREM 4. *ADOPT-ng returns an optimal solution.*

PROOF. We prove by induction on an increasingly growing suffix of the list of agents that this suffix converges to a solution that is optimal for their subproblem.

The induction step is immediate for the suffix composed of the agent A_n alone. Assume now that it is true for the suffix starting with A_k . Following the previous two lemmas, one can conclude that at quiescence A_{k-1} knows exactly the cumulated cost of the problems of its successors for its chosen assignment, and therefore knows that this cumulated cost cannot be better for any of its other values.

Since A_{k-1} has selected the value leading to the best sum of costs (between his own local cost and the costs of all subsequent agents), it follows that the suffix of agents starting with A_{k-1} converged to an optimal solution for their subproblem. \square

The space complexity is basically the same as for ADOPT. The SRCs do not change the space complexity of the valued nogood.

5.3 Optimizing valued nogoods

Both for the version of ADOPT-ng using DFS trees, as well as for the version that does not use such trees preprocessing, if valued nogoods are used for managing cost inferences, then a lot of effort can be saved at context switching by not discarding nogoods that remain valid [6]. The amount of saved effort is higher if the nogoods are carefully selected (to minimize their dependence on changes in often switched low priority variables). Computing valued nogoods by minimizing the index of the least priority variable involved in the context is shown by our experiments to perform well in this case. This is done by computing the valued nogoods using incrementally lower priority variables, and keeping the valued nogoods with lower priority agents only if they have better thresholds. Nogoods optimized in similar manner were used in several previous DisCSP techniques [2, 18].

5.4 Exploiting DFS trees

Note that while our versions of ADOPT work better than the original DFS-tree based version, they can also create hybrids by using an existing DFS tree. We have identified two ways of exploiting such an existing structure. The first way is by having each agent send his valued nogood only to its parent in the tree (less efficient in length of longest causal chain of messages but more efficient in number of total messages), roughly equivalent to the original ADOPT. The other way is by sending valued nogood to all the ancestors. This later hybrid can be seen as a certain fulfillment of a direction of research suggested in [9], namely communication of costs to higher priority parents.

An extension proposed to this work consists in integrating consistency maintenance in nogood-based optimization. This can be done with the introduction of *valued consistency nogoods*, as described in [12, 13].

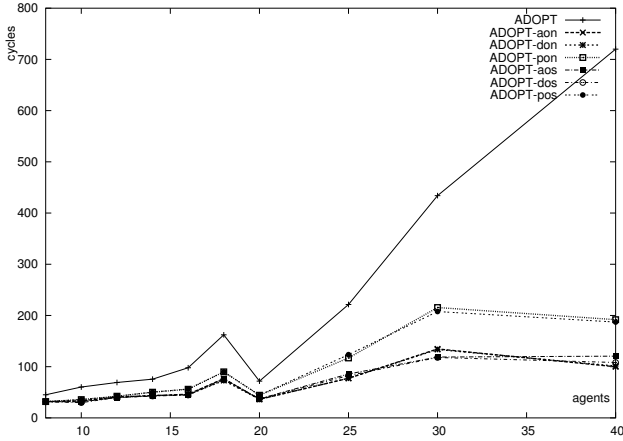


Figure 4: Longest causal chain of messages (cycles) used to solve versions of ADOPT using CAs, on problems with density .2.

6. EXPERIMENTS

We implemented several versions of ADOPT-ng, differing by how the agents picks the targets of his nogoods. In one of the implementations, ADOPT-pon, valued global nogoods are sent only to the parent of the current agent in the DFS tree. In ADOPT-don, the valued global nogoods are sent to all the ascendent of the current agent in the DFS tree. ADOPT-aon is a version where the DFS tree is reduced to the linear list of agents (each having the predecessor as parent). ADOPT-pos, ADOPT-dos, and ADOPT-aos are the corresponding versions with valued nogoods rather than valued global nogoods. Our current implementation does not yet exploit the threshold nogoods in `ok?` messages and does not store the optional CAs in `ca[crt.val][k]` for any k .

The algorithms were compared on the set of problems posted together with ADOPT, which are the same problems that are used to report ADOPT’s performance in [9]. To correctly compare our techniques with the original ADOPT we have used the same order (or DFS trees) on agents for each problem. The set of problems distributed with ADOPT and used here contain 25 problems for each problem size. It contains problems with 8, 10, 12, 14, 16, 18, 20, 25, 30, and 40 agents, and for each of these number of agents it contains test sets with density .2 and respectively with density .3. The density of a (binary) constraint problem’s graph with n variables is defined by the ratio between the number of binary constraints and $\frac{n(n-1)}{2}$. Results are averaged on the 25 problems with the same parameters.

The number of cycles, i.e., longest causal (sequential) chain of messages, for problems with density .2 is given in Figure 4. Results for problems with density .3 are given in Figure 5. The original ADOPT for 20 and 25 agents and density .3 required more than 2 weeks for solving one of the problems, and it was therefore evaluated using only the remaining 24 problems at those problem sizes.

We can note that the use of valued nogoods brought up to 10 times improvement on problems of density 0.2, and up to 5 times improvement on the problems of density .3.

Another interesting remark is that sending nogoods only to the parent node is significantly worse (in number of cy-

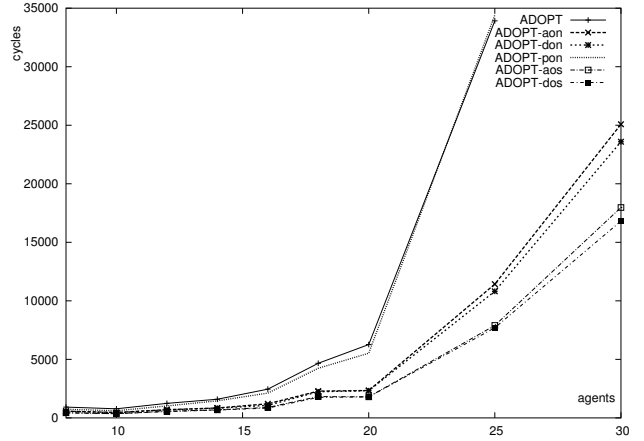


Figure 5: Longest causal chain of messages (cycles) used to solve versions of ADOPT using CAs, on problems with density .3.

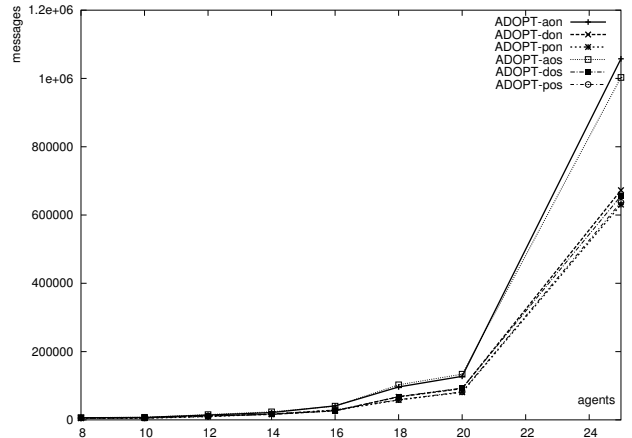


Figure 6: Total number of messages used by versions of ADOPT-ng using CAs to solve problems with density .3.

cles), than sending nogoods to all ancestors. Versions using DFS trees require less parallel/total messages, being more network friendly, as seen in Figure 6.

Figure 5 shows that, with respect to the number of cycles, the use of SRCs practically replaces the need of precomputing a DFS tree since ADOPT-aos is one of the best solvers, only slightly worse than ADOPT-dos. SRCs bring improvements over versions with valued global nogoods, since SRCs allow to detect dynamically obtained independences.

We do not perform any runtime comparison since our versions of ADOPT are implemented in C++, while the original ADOPT is in JAVA (which obviously leads to all our versions being an irrelevant order of magnitude faster).

It is visible from Figure 5 that the highest improvement in number of cycles is brought by sending valued nogoods to other ancestors besides the parent. The next factor for improvement with difficult problems (density .3) was the use of SRCs. The use of the structures of the DFS tree

bring slight improvements in number of cycles (when nogoods reach all ancestors) and large improvements in total message exchange.

Experimental comparison with DPOP is redundant since its performance can be easily predicted. DPOP is a good choice if the induced width γ of the graph of the problem is smaller than $\log_d T/n$ and smaller than $\log_d S$ where T is the available time, n the number of variables and d the domain size where S is the available memory of a computer.

7. CONCLUSIONS

The ADOPT distributed constraint optimization algorithm can be used efficiently (in number of cycles) without its preprocessing step that computes a DFS tree by using valued nogoods tagged with sets of references to culprit constraints (SRCs). The generalized algorithm is denoted ADOPT-ng. SRCs allow to detect and exploit dynamically created independences between subproblems (that are due to some assignments). Experimentation shows that it is important to send nogoods to several higher priority agents.

DFS trees can still be used in conjunction with the value nogood paradigm for optimization improving the total number of messages. ADOPT-ng versions exploiting DFS trees that we tested so far are also slightly better (in number of cycles) than the ones without DFS trees.

Besides that elegance brought by valued nogoods to the description and implementation of ADOPT in ADOPT-ng, use of SRCs to dynamically detect and exploit independences, as well as generalized communication of nogoods to several ancestors, brings experimental improvements of an order of magnitude.

8. REFERENCES

- [1] S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *AAMAS*, 2005.
- [2] C. Bessiere, I. Brito, A. Maestre, and P. Meseguer. Asynchronous backtracking without adding links: A new member in the abt family. *Artificial Intelligence*, 161:7–24, 2005.
- [3] Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*, 2000.
- [4] J. Davin and P. J. Modi. Impact of problem centralization in distributed cops. In *DCR*, 2005.
- [5] R. Dechter. *Constraint Programming*. Morgan Kaufman, 2003.
- [6] M. L. Ginsberg. Dynamic backtracking. *Journal of AI Research*, 1, 1993.
- [7] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *Proceedings of the Conference on Constraint Processing (CP-97), LNCS 1330*, pages 222–236, 1997.
- [8] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *AAMAS*, 2004.
- [9] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *AIJ*, 161, 2005.
- [10] P. J. Modi, M. Tambe, W.-M. Shen, and M. Yokoo. A general-purpose asynchronous algorithm for distributed constraint optimization. In *Distributed Constraint Reasoning, Proc. of the AAMAS'02 Workshop*, Bologna, July 2002. AAMAS.
- [11] A. Petcu and B. Faltings. Approximations in distributed optimization. In *Principles and Practice of Constraint Programming CP 2005*, 2005.
- [12] M.-C. Silaghi. *Asynchronously Solving Distributed Problems with Privacy Requirements*. PhD Thesis 2601, (EPFL), June 27, 2002. <http://www.cs.fit.edu/~msilaghi/teza>.
- [13] M.-C. Silaghi. Howto: Asynchronous PFC-MRDAC -optimization in distributed constraint problems +/-ADOPT-. In *IAT*, Halifax, 2003.
- [14] M.-C. Silaghi and B. Faltings. A comparison of DisCSP algorithms with respect to privacy. In *AAMAS-DCR*, 2002.
- [15] M.-C. Silaghi, J. Landwehr, and J. B. Larrosa. volume 112 of *Frontiers in Artificial Intelligence and Applications*, chapter Asynchronous Branch & Bound and A* for DisWCSPs with heuristic function based on Consistency-Maintenance. IOS Press, 2004.
- [16] M.-C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *3rd IC on Intelligent Agent Technology*, pages 531–535, 2004.
- [17] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Consistency maintenance for ABT. In *Proc. of CP'2001*, pages 271–285, Paphos, Cyprus, 2001.
- [18] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Hybridizing ABT and AWC into a polynomial space, complete protocol with reordering. Technical Report #01/364, EPFL, May 2001.
- [19] R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–193, 1977.
- [20] R. Wallace and M.-C. Silaghi. Using privacy loss to guide decisions in distributed CSP search. In *FLAIRS'04*, 2004.
- [21] M. Yokoo. Constraint relaxation in distributed constraint satisfaction problem. In *ICDCS'93*, pages 56–63, June 1993.
- [22] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *ICDCS*, pages 614–621, June 1992.
- [23] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE TKDE*, 10(5):673–685, 1998.
- [24] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of 3rd ICMAS'98*, pages 372–379, 1998.
- [25] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.
- [26] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proc. of AAAI*, Edmonton, July 2002.