ADOPT-ng: Unifying Asynchronous Distributed Optimization with Asynchronous Backtracking

Marius C. Silaghi[†] and Makoto Yokoo[‡]
[†]Florida Institute of Technology
[‡]Kyushu University
msilaghi@fit.edu, yokoo@is.kyushu-u.ac.jp

October 9, 2006

Abstract

This article presents an asynchronous algorithm for solving Distributed Constraint Optimization problems (DCOPs). The proposed technique unifies asynchronous backtracking (ABT) and asynchronous distributed optimization (ADOPT) where valued nogoods enable more flexible reasoning and more opportunities of communication, leading to an important speed-up. The concept of valued nogood is an extension of the concept of classic nogood that associates the list of conflicting assignments 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). 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 constraint graph. Valued nogoods allow for automatically detecting and exploiting the best DFS tree compatible with the current ordering and it is sufficient to ensure that a short such DFS tree exists. Also, the inference rules available for valued nogoods help to exploit schemes of communication where more feedback is sent to higher priority agents. Together they result in an order of magnitude improvement.

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 that are subject to constraints. The natural distribution comes from the assumption that only a subset of the agents has

knowledge of each given constraint. Nevertheless, in DCOPs 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 [40].

Several synchronous and asynchronous distributed algorithms have been proposed for solving DCOPs in a distributed manner. Since a DCOP 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 toward modifications to the distribution of the problem (modification accepted only when some reasoning infers it is unavoidable for guaranteeing that a solution can be reached). This criteria is widely believed to be valuable and adaptable for large, open, and/or dynamic distributed problems. It is also perceived as an alternative approach to privacy requirements [31, 39, 44, 34].

A synchronous algorithm, synchronous branch and bound, was the first known distributed algorithm for solving DCOPs [17]. Stochastic versions have also been proposed [45]. 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 more time consuming than local computations. A measure representing this assumption well is given by the number of cycles of a simulator that lets each agent in turn process all the messages that it receives [41]. Within the mentioned assumption, this measure is equivalent for real solvers to the longest causal chain of sequential messages, as used in [36].

From the point of view of this measure, a very efficient currently existing DCOP solver is DPOP [24, 23], which is linear in the number of variables. However, that algorithm generally has message sizes and local computation costs that are exponential in the induced width of a chosen depth-first search tree of the constraint graph of the problem. This clearly invalidates the assumptions that lead to the acceptance of the number of cycles as an efficiency measure. Some of the agents are also very disadvantaged in DPOP with respect to their privacy [15]. Effort is currently directed toward reducing these drawbacks [25].

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 sub-problems owned by agents found in conflict [20]. ADOPT can be described as a parallel version of (Iterative Deepening) A* [33]. While DisAO is typically criticized for its significant abandon of the maintenance of the natural distribution of the problem at the first conflict (and expensive local computations invalidating the above assumptions as for DPOP [9, 19, 1]), ADOPT can be criticized for its strict message pattern that only provides reduced reasoning opportunities. ADOPT works with orderings on agents dictated by some Depth-First Search tree on the constraint graph, and allows cost communication from an agent only to its parent node.

It is easy to construct huge problems whose constraint graphs are forests and which can be easily solved by DPOP (in linear time), but are 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 type of nogoods, called *valued nogoods* [8], that besides automatically detecting and exploiting the DFS tree of the constraint graph coherent with the current order, helps to exploit additional communication leading to significant improvement in efficiency. The examples given of additional communication are based on allowing each agent to send feedback via valued nogoods to several higher priority agents in parallel. The usage of nogoods is a source of much flexibility in asynchronous algorithms. A nogood specifies a set of assignments that conflict with existing constraints [38]. A basic version of the valued nogoods consists of 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 described use of valued nogoods leads to efficiency improvements even if exploited in conjunction with a previously known DFS tree, instead of the less semantically explicit cost messages of ADOPT. Valued nogoods that are associated with a list of culprit constraints produce additional important improvements. Each of these incremental concepts and improvements is described in the following sections.

We start by defining the general DCOP problem, followed by introduction of the immediately related background knowledge consisting of the ADOPT algorithm and use of Depth-First Search trees in optimization. In Section 5 we also describe valued nogoods together with the simplified version of valued global nogoods. In Section 6 we present our new algorithm that unifies ADOPT with the older Asynchronous Backtracking (ABT). The algorithm is introduced by first describing the goals in terms of new communication schemes to be enabled. Then the data structures needed for such communication are explored together with the associated flow of data. Finally the pseudo-code and the proof of optimality are provided before discussing other existing and possible extensions. Several different versions mentioned during the description are compared experimentally in the last section.

2 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 (DCOP) *A distributed constraint optimization problem (DCOP), aka distributed valued CSP, is defined by a set of agents* $A_1, A_2, ..., A_n$*, a set* X *of variables,* $x_1, x_2, ..., x_n$*, and a set of functions* $f_1, f_2, ...f_i, ..., f_n$*,*

$f_i : X_i \rightarrow \mathbb{R}_+$, $X_i \subseteq X$, *where only* A_i *knows* f_i. *We assume that* x_i *can only take values from a domain* $D_i = \{1, ..., d\}$.

Denoting with x *an assignment of values to all the variables in* X, *the problem is to find* $\underset{x}{\operatorname{argmin}} \sum_{i=1}^{n} f_i(x_{|X_i})$.

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

By $x_{|X_i}$ we denote the projection the set of assignments in x on the set of variables in X_i. Our idea can be easily applied to general valued CSPs.

3 DFS-trees

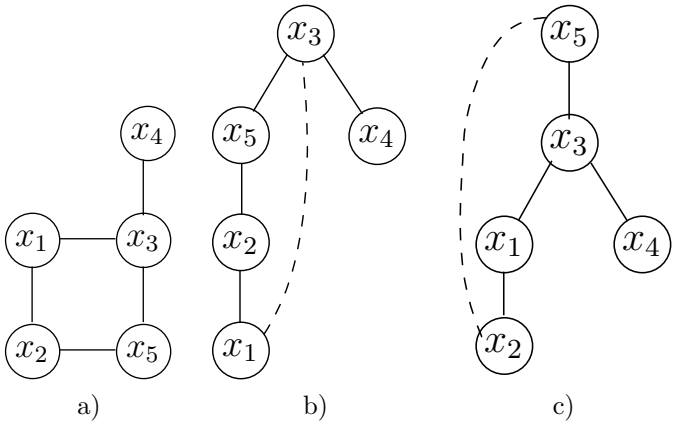

Figure 1: For a DCOP with primal graph depicted in (a), two possible DFS trees (pseudo-trees) are (b) and (c). Interrupted lines show constraint graph neighboring relations not in the DFS tree.

The primal graph of a DCOP is the graph having the variables in X as nodes and having an arc for each pair of variables linked by a constraint [11]. A Depth-First Search (DFS) tree associated with a DCOP is a spanning tree generated by the arcs used for first visiting each node during some Depth-First Traversal of its primal graph. DFS trees were first successfully used for distributed constraint problems in [7]. 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 $\{x_1, x_4, x_5\}$. 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, $\{x_3, x_5\}$ are ancestors of x_2. x_3 has no ancestors. If

a variable x_i is an ancestor of a variable x_j, then x_j is a *descendant* of x_i. For example, in Figure 1.b, $\{x_1, x_2\}$ are descendants of x_5.

4 ADOPT and ABT

ADOPT. ADOPT [21] is an asynchronous complete DCOP solver, which is guaranteed to find an optimal solution. Here, we only show a brief description of ADOPT. Please consult [21] for more details. First, ADOPT organizes 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 sub-tree 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 of 0. A cost message also contains the information of the upper bound of the cost of the sub-tree, i.e., the actual cost of the sub-tree. 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.

ABT. Distributed constraint satisfaction problems are special cases of DCOPs where the constraints f can return only values in $\{0, \infty\}$. The basic asynchronous algorithm for solving distributed constraint satisfaction problems is asynchronous backtracking (ABT) [42]. ABT uses a total priority order on agents where agents announce new assignments to lower priority agents using **ok?** messages, and announce conflicts to lower priority agents using **nogood** messages. New dependencies created by dynamically learned conflicts are announced using **add-link** messages. An important difference between ABT and ADOPT is that, in ABT, conflicts (the equivalents of cost) can be freely sent to any higher priority agent.

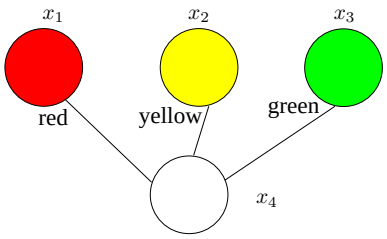

Figure 2: MIN resolution on valued global nogoods

5 Cost of nogoods

Previous flexible algorithms for solving distributed constraint satisfaction problems exploit the inference power of nogoods (e.g., ABT, AWC, ABTR [41, 42, 37])[1]. A nogood $\neg N$ stands for a set N of assignments that was proven impossible, by inference, using constraints. If $N = (\langle x_1, v_1\rangle, ..., \langle x_t, v_t\rangle)$ where $v_i \in D_i$, then we denote by $\overline{N}$ the set of variables assigned in N, $\overline{N} = \{x_1, ..., x_t\}$.

5.1 Valued Global Nogoods

In order to apply nogood-based algorithms to DCOP, 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.*

Example 5.1 *For the graph coloring problem in Figure 2 (assume it has a constraint $x_1 \neq x_2$ with weight 10), a possible valued global nogood is $[10, \{(x_1, r), (x_4, r)\}]$. It specifies that if x_1=r and x_2=r then there exists no solution with a cost lower than 10.*

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

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

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., $x \notin \overline{N}$.*

[1] Other algorithms, like AAS, exploit generalized nogoods (i.e., extensions of nogoods to sets of values for a variable), and the extension of the work here for that case is suggested in [27]

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) *Given a minimization VCSP, assume that we have a set of GCAs of the form (v, c_v, N_v) that has the property of containing exactly one GCA for each value v in the domain of variable x_i and that for all k and j, the assignments for variables $\overline{N_k} \cap \overline{N_j}$ are identical in both N_k and N_j. Then one can resolve a new valued global nogood: $[\min_v c_v, \cup_v N_v]$.*

Example 5.2 *For the graph coloring problem in Figure 2 (weighted constraints 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): $[10, \{(x_1, r), (x_4, r)\}]$, obtaining for x_4 the GCA $(r, 10, \{(x_1, r)\})$

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

(g): $[7, \{(x_3, g), (x_4, g)\}]$, obtaining for x_4 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 to valued global nogoods:

Corollary 1.1 *Assume $\mathcal{S}$ is a set of nogoods associated with the variable x_i, such that for each $[c_v, S_v]$ in $\mathcal{S}$, $\exists \langle x_i, v \rangle \in S_v$. If $\mathcal{S}$ contains exactly one global valued nogood $[c_v, S_v]$ for each value v in the domain of variable x_i of a minimization VCSP, then one can resolve a new valued global nogood: $[\min_v c_v, \cup_v (S_v \setminus \langle x_i, v \rangle)]$.*

5.2 Valued Nogoods

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 sub-trees [22, 7].*

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 inference from 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. This can be done by tagging cost assessments with the identifiers of the constraints used to infer them.

Definition 3 *A* set of references to constraints *(***SRC***) is a set of identifiers, each 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). [2]

SRCs help to define a generalization of the concept of *valued global nogood* named *valued nogood* [8].

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. Valued global nogoods are valued nogoods whose SRCs contain the references of all the constraints.

Once we decide that a nogood $[SRC, c, (\langle x_1, v_1\rangle, ..., \langle x_i, v_i\rangle)]$ will be applied to a certain variable x_i, we obtain a cost assessment tagged with the set of references to constraints SRC[3], denoted $(SRC, v_i, c, (\langle x_1, v_1\rangle, ..., \langle x_{i-1}, v_{i-1}\rangle))$.

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 where the assignment of* x_i *is set to the value v.*

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

Remark 3 *Given a valued nogood* $[SRC, c, N]$*, one can infer the 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., where* $x \notin \overline{N}$*.*

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

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

Proposition 2 (sum-inference [8]) *A set of cost assessments of type* (SRC_i, v, c_i, N_i) *for a value v of some variable, 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 all* 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 5.3 *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)*:*

[2] 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.

[3] This is called a *valued conflict list* in [27]

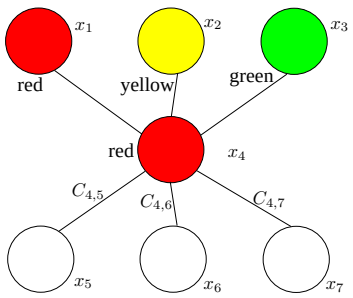

Figure 3: SUM-inference resolution on CAs

- $[\{C_{4,5}\}, 5, \{(x_2, y), (x_4, r)\}]$ *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* $(\{C_{4,7}\}, r, 9, \{(x_2, y)\})$

Also assume that based on x_4's constraint with x_1, one has obtained for $\langle x_4, r\rangle$ 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 4 (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 extended to the case where assignments are generalized to sets [27].*

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

Proposition 3 (min-resolution [8]) *Assume that we have a set of cost assessments for x_i of the form (SRC_v, v, c_v, N_v) that has the property of containing exactly one CA for each value v in the domain of variable x_i and that for all k and j, the assignments for variables $\overline{N_k} \cap \overline{N_j}$ are identical in both N_k and N_j. Then the CAs in this set 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 5.4 *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 to valued nogoods:

Corollary 3.1 (min-resolution on nogoods) *From a set of valued nogoods* $[SRC_v, c_v, S_v)]$ *(such that* $\exists v, \langle x_i, v\rangle \in S_v$*) 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:* $[\cup_v SRC_v, \min_v c_v, \cup_v (S_v \setminus \langle x_i, v\rangle)]$.

6 ADOPT with nogoods

We now present a distributed optimization algorithm whose efficiency is improved by exploiting the increased flexibility brought by the use of valued nogoods. 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).

As in ABT, agents communicate with **ok?** messages proposing new assignments of the variable of the sender, **nogood** messages announcing a nogood, and **add-link** messages announcing interest in a variable. As 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 (as enforced by the Internet transport control protocol). Attachment of counters to proposed assignments and nogoods also ensures this requirement (i.e., older assignments and older nogoods for the currently proposed value are discarded).

6.1 Exploiting DFS trees for Feedback

In ADOPT-ng, agents are totally ordered as in ABT, A_1 having the *highest priority* and A_n the lowest priority. The *target* of a valued nogood is the position of the lowest priority agent among those that proposed an assignment referred by that nogood. Note that the basic version of ADOPT-ng does not maintain a DFS tree, but each agent can send messages with valued nogoods to any predecessor. We also propose hybrid versions that can spare network bandwidth by exploiting an existing DFS tree. We have identified two ways of exploiting such an existing structure. The first is by having each agent send its valued nogood only to its parent in the tree and it is roughly equivalent to the original

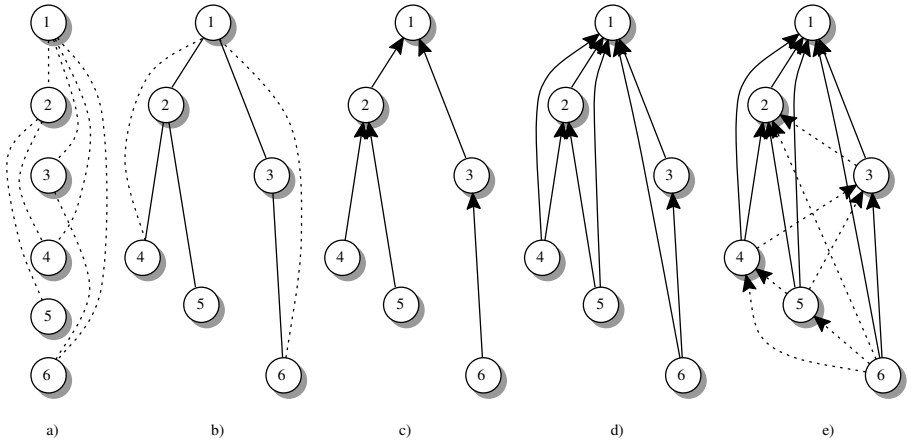

Figure 4: Feedback modes in ADOPT-ng. a) a constraint graph on a totally ordered set of agents; b) a DFS tree compatible with the given total order; c) ADOPT-p_: sending valued nogoods only to parent (graph-based backjumping); d) ADOPT-d_ and ADOPT-D_: sending valued nogoods to any ancestor in the tree; e) ADOPT-a_ and ADOPT-A_: sending valued nogoods to any predecessor agent.

ADOPT. The other way is by sending valued nogoods only to ancestors. This later hybrid approach can be seen as a fulfillment of a direction of research suggested in [21], namely communication of costs to higher priority parents.

The versions of ADOPT-ng described in this article are differentiated using the notation **ADOPT-XYZ**. **X** shows the destinations of the messages containing valued nogoods. X has one of the values $\{p, a, A, d, D\}$ where p stands for *parent*, a and A stand for *all predecessors*, and d and D stand for *all ancestors in a DFS trees*. **Y** marks the optimization criteria used by sum-inference in selecting a nogood when the inputs have the same threshold. For now we use a single criterion, denoted o, which consists of choosing the nogood whose target has the highest priority. **Z** specifies the type of nogoods employed and has possible values $\{n, s\}$, where n specifies the use of valued global nogoods (without SRCs) and s specifies the use of valued nogoods (with SRCs).

The different schemes are described in Figure 4. The total order on agents is described in Figure 4.a where the constraint graph is also depicted with dotted lines representing the arcs. Each agent (representing its variable) is depicted with a circle. A DFS tree of the constraint graph which is compatible to this total order is depicted in Figure 4.b. ADOPT gets such a tree as input, and each agent sends COST messages (containing information roughly equivalent to a valued global nogood) only to its parent. As mentioned above, the versions of ADOPT-ng that replicate this behavior of ADOPT when a DFS tree is provided are called ADOPT-p_, where p stands for *parent* and the underscores stand

for any legal value defined above for Y and Z respectively. This method of announcing conflicts based on the constraint graph is depicted in Figure 4.c and is related to the classic Graph-based Backjumping algorithm [10, 16].

In Figure 4.d we depict the nogoods exchange schemes used in ADOPT-d__ and ADOPT-D_ where, for each new piece of information, valued nogoods are separately computed to be sent to each of the ancestors in the known DFS tree. These schemes are enabled by valued nogoods and are shown by experiments to bring large improvements. As for the initial version of ADOPT, the proof for ADOPT-d_ and ADOPT-D__ shows that the only mandatory nogood messages for guaranteeing optimality in this scheme are the ones to the parent agent. However, agents can infer from their constraints valued nogoods that are based solely on assignments made by shorter prefixes of the ordered list of ancestor agents. The agents try to infer and send valued nogoods separately for all such prefixes.

Figure 4.e depicts the basic versions of ADOPT-ng, when a DFS is not known (ADOPT-a_ and ADOPT-A__), where nogoods can be sent to all predecessor agents. The dotted lines show messages, which are sent between independent branches of the DFS tree, and which are expected to be redundant. Experiments show that valued nogoods help to remove the redundant dependencies whose introduction would otherwise be expected from such messages. The provided proof for ADOPT-a_ and ADOPT-A__ shows that the only mandatory nogood messages for guaranteeing optimality in this scheme are the ones to the immediately previous agent. However, agents can infer from their constraints valued nogoods that are based solely on assignments made by shorter prefixes of the ordered list of all agents. As in the other case, the agents try to infer and send valued nogoods separately for all such prefixes.

The valued nogood computed for the prefix $A_1, ..., A_k$ ending at a given predecessor A_k may not be different from the one of the immediately shorter prefix $A_1, ..., A_{k-1}$. Sending that nogood to A_k may not affect the value choice of A_k, since the cost of that nogood applies equally to all values of A_k according to Remark 3. Exceptions appear in the case where such nogoods cannot be composed by sum-inference with some valued nogoods of A_k. The versions ADOPT-D__ and ADOPT-A__ correspond to the case where optional nogood messages are only sent when the target of the payload valued nogood is identical to the destination of the message. The versions ADOPT-d_ and ADOPT-a__ correspond to the case where optional nogood messages are sent to all possible destinations each time that the payload nogood has a non-zero threshold. I.e., in those versions **nogood** messages are sent even then the target of the transported nogood is not identical to the destination agent but has a higher priority.

6.2 Data Structures

Each agent A_i stores its *agent-view* (received assignments), and its outgoing links (agents of lower priority than A_i and having constraints on x_i). The instantiation of each variable is tagged with the value of a separate counter incremented each time the assignment changes. To manage nogoods and CAs, A_i

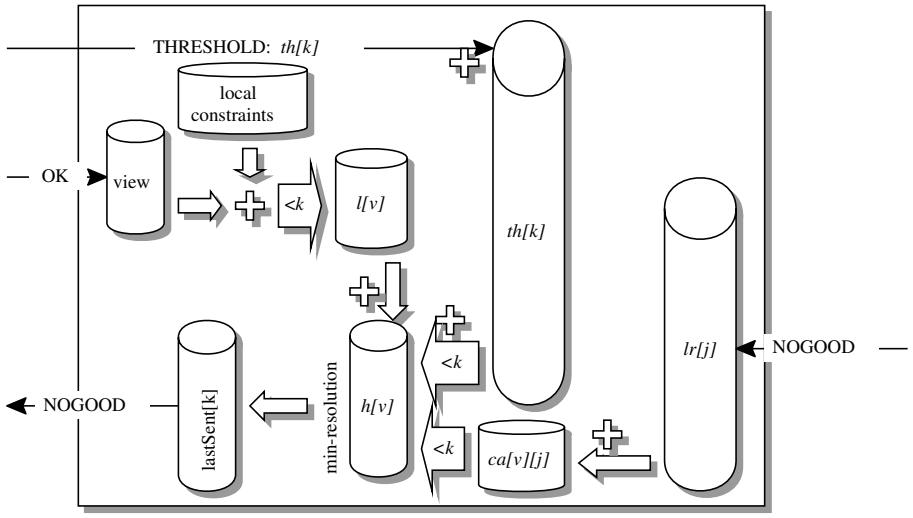

Figure 5: Schematic flow of data through the different data structures used by an agent A_i in ADOPT-ng.

uses matrices l[1..d], h[1..d], ca[1..d][i+1..n], th[1..i], 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. These matrices have the following usage.

- l[k] stores a CA for $x_i = k$, which is inferred solely from the local constraints between x_i and prior variables.
- ca[k][j] stores a CA for $x_i = k$, which is obtained by sum-inference from valued nogoods received from A_j.
- th[k] stores nogoods coming via **threshold/ok?** messages from A_k.
- h[v] stores a CA for x_i=v, which is inferred from ca[v][j], l[v] and th[t] for all t and j.
- lr[k] stores the last valued nogood received from A_k.
- lastSent[k] stores the last valued nogood sent to A_k.

The names of the structures were chosen by following the relation of ADOPT with A* search [28, 33]. Thus, h stands for the "heuristic" estimation of the cost due to constraints maintained by future agents (equivalent to the $h()$ function in A*) and l stands for the part of the standard $g()$ function of A* that is "local" to the current agent. Here, as in ADOPT, the value for $h()$ is estimated by aggregating the equivalent of costs received from lower priority agents. Since the costs due to constraints of higher priority agents are identical for each value, they are irrelevant for the decisions of the current agent. Thus, the function $f()$

of this version of A* is computed combining solely l and h. We currently store the result of combining h and l in h itself to avoid allocating a new structure for $f()$.

The structures lr and th store received valued nogoods and ca stores intermediary valued nogoods used in computing h. The reason for storing lr, th and ca is that change of context may invalidate some of the nogoods in h while not invalidating each of the intermediary components from which h is computed. Storing these components (which is optional) saves some work and offers better initial heuristic estimations after a change of context. The cost assessments stored in $ca[v][j]$ of A_i also maintain the information needed for THRESHOLD messages, namely the heuristic estimate for the value v of the variable x_i at successor A_j (to be transmitted to A_j if the value v is proposed again).

The array $lastSent$ is used to store at each index k the last valued nogood sent to the agent A_k. The array lr is used to store at each index k the last valued nogood received from the agent A_k. Storing them separately guarantees that in case of changes in context, they are discarded at the recipient only if they are also discarded at the sender. This property guarantees that an agent can safely avoid retransmitting to A_k messages duplicating the last sent nogood, since if it has not yet been discarded from $lastSent[k]$ then the recipients have not discarded it from $lr[k]$ either.

6.3 Data flow in ADOPT-ng

The flow of data through these data structures of an agent A_i is illustrated in Figure 5. Arrows $\Leftarrow$ are used to show a stream of valued nogoods being copied from a source data structure into a destination data structure. These valued nogoods are typically sorted according to some parameter such as the source agent, the target of the valued nogood, or the value v assigned to the variable x_i in that nogood (see Section 6.2). The + sign at the meeting point of streams of valued nogoods or cost assessments shows that the streams are combined using sum-inference. The $\stackrel{+}{\Leftarrow}$ sign is used to show that the stream of valued nogoods is added to the destination using sum-inference, instead of replacing the destination. When computing a nogood to be sent to A_k, the arrows marked with $<k$ restrict the passage to allow only those valued nogoods containing solely assignments of the variables of agents $A_1, ..., A_k$. Our current implementation recomputes the elements of h and l separately for each target agent A_k by discarding the previous values.

6.4 ADOPT-ng pseudo-code and proof

The pseudo-code for the procedures in ADOPT-ng is given in Algorithms 1 and 2. To extract the cost of a CA, we introduce the function $cost()$, $cost((SRC, v, c, N))$ returns c. The $min\_resolution(j)$ function applies the min-resolution over the CAs associated to all the values of the variable of the current agent, but uses only CAs having no assignment from agents with lower priority than A_j. More exactly it first re-computes the array h using only CAs in

```
when receive ok?(⟨x_j, v_j⟩, tvn) do
    integrate(⟨x_j, v_j⟩);
    if (tvn no-null and has no old assignment) then
        k:=target(tvn); // threshold tvn as common cost;
        th[k]:=sum-inference(tvn,th[k]);
    check-agent-view();
when receive add-link(⟨x_j, v_j⟩) from A_j do
    add A_j to outgoing-links;
    if (⟨x_j, v_j⟩) is old, send new assignment to A_j;
when receive nogood(rvn, t) from A_t 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;
    lr[t]:=rvn; foreach value v of x_i such that rvn_|v is not ∅ do
        vn2ca(rvn, i, v) → rca (a CA for the value v of x_i);
        ca[v][t]:=sum-inference(rca,ca[v][t]);
        update h[v] and retract changes to ca[v][t] if h[v]'s cost decreases;
    check-agent-view();
```

Algorithm 1: Receiving messages of A_i in ADOPT-ng

ca and l that contain only assignments from $A_1, ..., A_j$, and then applies min-resolution over the obtained elements of h. As mentioned above, in the current implementation we recompute l and h at each call to $min\_resolution(j)$.

The $sum\_inference()$ function used in Algorithm 2 applies the sum-inference to its parameters whenever this is possible (it detects disjoint SRCs); otherwise, it selects the nogood with the highest threshold or whose lowest priority assignment has the highest priority (this has been previously used in [4, 37]). The function $vn2ca(vn, i)$ transforms a valued nogood vn in a cost assessment for x_i. Its inverse is function $ca2vn$. If vn has no assignment for x_i, then a cost assessment can be obtained according to Remark 3. The function $vn2ca(vn, i, v)$ translates vn into a cost assessment for the value v of x_i, using the technique in Remark 3 if needed. The function $target(N)$ gives the index of the lowest priority variable present in the assignment of nogood N. As with file expansion, when "*" is present in an index of a matrix, the notation is interpreted as the set obtained for all possible values of that index (e.g., ca[v][*] stands for $\{ca[v][t] \mid \forall t\}$). Given a valued nogood ng, the notation $ng_{|v}$ stands for vn2ca(ng) when ng's value for x_i is v, and $\emptyset$ otherwise.

Each agent A_i starts by calling the init() procedure in Algorithm 2, which initializes 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 to any received message with the corresponding procedure in Algorithm 1: "**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 within the call to the function *integrate()* in Algorithm 2. Any discarded element of *ca* is recomputed from *lr*. Received nogoods are stored in matrices *lr* and *th* (Algorithm 1). A_i always sets its *crt_val* to the index with the lowest CA threshold in vector h (preferring the previous assignment in case of ties). On each change that propagates to h, and for each higher priority agent A_j (or for each ancestor in versions using DFS trees), the elements of h are recomputed separately by min-resolution(j) to generate new nogoods for A_j. The simultaneous generation and use of multiple nogoods is already known to be useful for the constraint satisfaction case [43].

The threshold valued nogood *tvn* delivered with **ok?** messages sets a common cost on all values of the receiver (see Remark 3), effectively setting a threshold on costs below which the receiver does not change its value. This achieves the effect of THRESHOLD messages in ADOPT.

The procedure described in the following remark is used in the proof of termination and optimality.

Remark 5 *The order of combining CAs matters. To compute h[v]:*

1. *a) When maintaining DFS trees, for each value v, CAs are combined separately for each set s of agents defining a DFS sub-tree of the current node:*
   tmp[v][s]=sum-inference$_{t\in s}$(ca[v][t]).
   b) Otherwise, with ADOPT-a__ and ADOPT-A__, we act as if we have a single sub-tree:
   tmp[v]=sum-inference$_{t\in[i+1,n]}$(ca[v][t]).

2. *CAs from step 1 (a or b) are combined:*
   In case (a) this means: $\forall v, s; h[v]$=sum-inference$_{\forall s}$(tmp[v][s]).
   Note that the SRCs in each term of this sum-inference are disjoint and therefore we obtain a valued nogood with threshold given by the sum of the individual thresholds obtained for each DFS sub-tree (or larger).

   For case (b) we obtain h[v]=tmp[v].
   This makes sure that at quiescence the threshold of $h[v]$ is at least equal to the total cost obtained at the next agent.

3. *Add l[v]: h[v]=sum-inference(h[v], l[v]).*

4. *Add threshold: h[v]=sum-inference(h[v], th[*]).*

Lemma 1 (Infinite Cycle) *At a given agent, assume that the agent-view no longer changes and that its array h (used for min-resolution and for deciding the next assignment) is computed only using cost assessments that are updated solely by sum-inference. In this case the thresholds of the elements of its h cannot be modified in an infinite cycle due to incoming valued nogoods.*

Proof. Valued nogoods that are updated solely by sum-inference have thresholds that can only increase (which can happen only a finite number of times). For a given threshold, modifications can only consist of modifying assignments to obtain lower target agents, which again can happen only a finite number of times. Therefore, after a finite number of events, the cost assessments used to infer h will not be modified any longer and therefore h will no longer be modified. □

Corollary 3.2 *If ADOPT-ng uses the procedure in Remark 5, then for a given agent-view, the elements of the array h for that agent cannot be modified in an infinite cycle.*

Remark 6 *Since lr contains the last received valued nogoods via messages other than* **ok?** *messages which change the agent-view, that array is updated by assignment with recently received nogoods without sum-inference. Therefore, it cannot be used directly to infer h.*

Note that with the described procedure, a newly arriving valued nogood can decrease the threshold of certain elements of h. This is because, while increasing the threshold of some element in ca, it can also modify its SRC and therefore forbid its composition by sum-inference with other cost assessments.

Remark 7 (Obtaining Monotonic Increase) *The implementation used for the experiments reported here avoids the undesired aforementioned effect, where incoming nogoods decrease thresholds of cost assessments in h. Namely, after a newly received valued nogood is added by sum-inference to the corresponding element of ca[v] for some value v, if the threshold of h[v] decreases then the old content of ca[v] is restored. Each new valued nogood is used for updating lr. On each change of the agent-view (set of known valid assignments), all values of ca are updated using the valued nogoods found in lr and th.*

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 this 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: thresholds of the nogoods only increase since they only change by sum-inference.

Lemma 2 *ADOPT-ng terminates in finite time.*

```
procedure init do
    h[v] := l[v]:=initialize CAs from unary constraints;
    crt_val=argmin_v(cost(h[v]));
    send ok?(⟨x_i, crt_val⟩,∅) to all agents in outgoing-links;
procedure check-agent-view() do
    for every A_j with higher priority than A_i (respectively ancestor in the
    DFS tree, when one is maintained) do
        for every(v ∈ D_i) update l[v] and recompute h[v];
                // with valued nogoods using only instantiations of {x_1, ..., x_j};
        if (h has non-null cost CA for all values of D_i) then
            vn:=min_resolution(j);
            if (vn ≠ lastSent[j]) then
                if (target(vn) == j) then
                    send nogood(vn,i) to A_j;
                    lastSent[j] = vn;
    crt_val=argmin_v(cost(h[v]));
    if (crt_val changed) then
        send ok?(⟨x_i, crt_val⟩, ca2vn(ca[crt_val][k]),i)
              to each A_k in outgoing_links;
procedure integrate(⟨x_j, v_j⟩) do
    discard elements in ca, th, lastSent and lr based on other values for x_j;
    use lr[t]|_v to replace each discarded ca[v][t];
    store ⟨x_j, v_j⟩ in agent-view;
```

Algorithm 2: Procedures of A_i in ADOPT-ng

Proof.

Given the list of agents $A_1, ..., A_n$, define the suffix of length m of this list as the last m agents. Then the result follows immediately by induction for an increasingly growing suffix (increasing m), assuming the other agents reach quiescence.

The basic case 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.

Let us now assume that the induction assertion is true for a suffix of k agents. Based on this assumption we now prove the induction step, namely that the property 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; otherwise, the assignment's CA threshold increases. By construction, thresholds for CAs associated with the values of A_{n-k} can only grow (see Remark 7). Even without the technique in Remark 7, thresholds for CAs associated with the values of A_{n-k} will eventually stop being modified as a consequence of Lemma 1. After values are proposed in turn and

the smallest threshold reaches its highest estimate, agent A_{n-k} selects the best value and reaches quiescence. The other agents reach quiescence according to the induction step. □

Lemma 3 *The last valued nogoods sent by each agent additively integrate the non-zero costs of the constraints of all of the agent's successors.*

Proof. At quiescence, each agent A_k has received the valued nogoods describing the costs of each of its successors (or descendants in the DFS tree when a DFS tree is maintained).

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

Assuming that it is true for 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) than the result of adding A_{k-1}'s local cost to one from any successor of A_k. Respecting the order in Remark 5 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}. □

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

Proof. We prove by induction on an ever increasing suffix of the list of agents that this suffix converges to a solution that is optimal for the union of the sub-problems of the agents in that suffix.

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 its 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 the union of their sub-problems. □

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

6.5 Optimizing valued nogoods

Both for the versions of ADOPT-ng using DFS trees, as well as for the version that does not use such DFS tree preprocessing, if valued nogoods are used for managing cost inferences, then a lot of effort can be saved at context switching by keeping nogoods that remain valid [14]. The amount of effort saved is higher if the nogoods are carefully selected (to minimize their dependence on

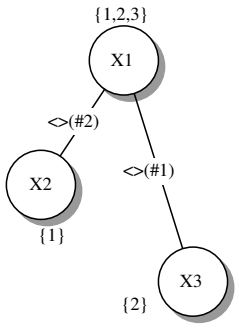

Figure 6: A DisCOP with three agents and two inequality constraints. The fact that the cost associated with not satisfying the constraint $x_1 \neq x_2$ is 2, is denoted by the notation (#2). The cost for not satisfying the constraint $x_1 \neq x_3$ is 1.

1.	A_1	**ok?**$\langle x_1, 1\rangle$	A_2, A_3
2.	A_2	**nogood**$[\lvert F, T, F\rvert, 2, \langle x_1, 1\rangle]$	A_1
3.	A_1	**ok?**$\langle x_1, 2\rangle$	A_2, A_3
4.	A_3	**nogood**$[\lvert F, F, T\rvert, 1, \langle x_1, 2\rangle]$	A_1, A_2
5.	A_1	**ok?**$\langle x_1, 3\rangle$	A_2, A_3
6.	A_2	**nogood**$[\lvert F, F, T\rvert, 1, \langle x_1, 2\rangle]$	A_1

Figure 7: Trace of ADOPT-aos and ADOPT-Aos on the problem in Figure 6

assignments for low priority variables, which change more often). We compute valued nogoods by minimizing the index of the least priority variable involved in the context. At sum-inference with intersecting SRCs, we keep the valued nogoods with lower priority target agents only if they have better thresholds. Nogoods optimized in similar manner were used in several previous DisCSP techniques [4, 37]. A similar effect is achieved by computing min_resolution(j) with incrementally increasing j and keeping new nogoods only if they have higher thresholds than previous ones with lower targets.

6.6 Example

Now we give a detailed example of a run of ADOPT-ng basic versions ADOPT-aos and ADOPT-Aos. Let us take the problem in Figure 6. Note that in this simple case the two versions do not differ since any optional nogood message can only leave from A_3 to A_1. Such a message is sent in ADOPT-aos only if it has a non-zero threshold, which happens only when A_1 is a target of the message, which means that it will also be sent in ADOPT-Aos. A trace is shown in Figure 7 where identical messages sent simultaneously to several agents are grouped by displaying the list of recipients. The agents start selecting values for their variables and announce them to interested lower priority agents. A_3

has no constraint between x_3 and x_2; therefore the first exchanged messages are **ok?** messages sent by A_1 to both successors A_2 and A_3 and proposing the assignment x_1=1.

After receiving the assignment from A_1, the best (and only) assignment for A_2 is x_2=1 at a cost of 2 due to the conflict with the constraint $x_1 \neq x_2$. Similarly A_3 instantiates x_3 with 2 and with a local cost of 0.

Since the best local cost of A_2 is not null, A_2 performs a min-resolution. Since a single value exists for A_2 and ca is empty, this min-resolution simply obtains a valued nogood defined by the existing local nogood: $h[1] = l[1] = [C_{1,2}, 2, \langle x_1, 1\rangle]$. In our implementation we decide to maintain a single reference for each agent's secret constraints. SRCs are represented as Boolean values in an array of size n. A value at index i in the SRC array set to T signifies that the constraints of A_i are used in the inference of that nogood. A_2 also stores the sent valued nogood in $lastSent[1]$ such that it avoids resending it without modification as a result of receiving other messages. A_1 stores this received valued nogood in $lr[2]$, from where it is used to update $ca[1][2]$, by sum-inference. Since $ca[1][2]$ is empty, it becomes equal to this valued nogood.

Agent A_1 now updates its $h[1]$ by setting it to $ca[1][2]$ (since $l[1]$ and $ca[1][3]$ are empty). Since the threshold of $h[1]$ becomes 2 and is higher than the threshold of the other two values, {2,3}, in the domain of x_1, A_1 changes the assignment of x_1 to one of them, here 2. This is announced through another **ok?** message to A_2 and A_3.

On the receipt of the **ok?** messages, the agents update their agent-view with the new assignment. Each agent tries to generate valued nogoods for each prefix of its list of predecessor agents: $\{A_1\}$ and $\{A_1, A_2\}$ respectively. This time it is A_2 whose only possible assignment leads to a non-zero local cost. Based on its agent-view and constraints, A_2 generates a corresponding valued nogood $[C_{1,3}, 1, \langle x_1, 2\rangle]$ with threshold 1 due to the weight 1 of its constraint. This valued nogood is sent to the agent A_1 whose assignment is involved in this nogood. To guarantee optimality the nogood is also sent to its immediate predecessor, namely the agent A_2, making sure that at quiescence all the costs of its children are summed.

After receiving this second nogood, A_1 stores it in $lr[3]$, used further by sum-inference to set $ca[2][3]$, and finally used to update $h[2]$. As a result, A_1 now switches its assignment to its value that has the lowest threshold in h, namely the value 3. The new assignment is again sent by **ok?** messages to its successors. Meanwhile, the agent A_2 also processes the valued nogood received from A_3 storing it in its own $lr[3]$, $ca[2][3]$ and $h[2]$. The nogood is not changed by sum inference or min-resolution at this agent; it is sent on to A_1 which stores it in $lr[2]$ and $ca[2][2]$. However, it does not lead to any modification in the $h[2]$ of A_1 since the SRCs of $ca[2][2]$ and $ca[2][3]$ have a nonempty intersection.

After receiving the third assignment from A_1, the other two agents reach quiescence with cost 0; thus an optimal solution is found. Note that the existence of message 6 depends on whether the message 5 (with the last assignment from A_1) reaches A_2 before or after the nogood from A_3, that the message 5 invalidates. The solution is found in 5 half-round-trips of messages (a logic time

of 5).

6.7 Possible Extensions

We addressed ADOPT-ng as an asynchronous version of A*, more exactly a version of iterative deepening A*, where the heuristic is computed by recursively using ADOPT-ng itself, and where the composition of the results of recursive ADOPT-ng is based on backtracking.

A proposed extension to this work consists of composing the recursive asynchronous heuristic estimator by using consistency maintenance. This can be done with the introduction of *valued consistency nogoods*. Details and variations are described in [27, 29, 33, 13, 12]. Another possible extension is by further generalizing the nogoods such that each variable can be assigned a set of values. This type of aggregation was shown in [32] to improve search, and the extension is detailed in [27].

In our implementation we concentrated on minimizing the logic time of the computation, evaluated as the number of rounds on a simulator. The optimization of local processing (which is polynomial in the number of variables) is not at the center of attention at this stage. Local computations can be optimized, for example, by reusing values of structures l and h computed at min-resolution for a given target agent in obtaining values of these structures at the min-resolution for messages sent to lower priority target agents. Further work can determine whether improvements could be made by storing separately the nogoods of h for each value of k.

Other extensions seem possible by integrating additive branch and bound searches on DFS sub-trees, as proposed by [6]. This can be added to ADOPT-ng by maintaining solution-based nogoods as suggested in [27]. It remains to be seen if the quality of solutions with a certain value can be predicted with the technique in [25]. Further improvements are possible by running ADOPT-ng in parallel for several orderings of the agents [26, 3].

ADOPT-ng can be seen as an extension of ABT. The extension of ABT called ABTR [35, 30] proposes a way to extend ABT-based algorithms to allow for dynamic ordering of the agents [2]. Work in the area consistent with this approach, but mainly favoring static ordering, appears in [18, 5]. Finding good heuristics was shown to be a difficult problem [37, 46] and here one will need to take into account the importance of the existence of a short DFS tree compatible to the current ordering.

7 Experiments

We implemented several versions of ADOPT-ng. Some versions use valued nogoods while other versions use valued global nogoods. Some versions maintain an optional DFS tree precomputed on the constraint graph. Some versions send

Agents	ADOPT	aos	Aos	dos	Dos
8	922.2	429.48	427.92	429.2	427.76
10	779.84	354.12	365.76	351.16	357.48
12	1244.56	544.76	562.96	544.24	552.88
14	1591	674.56	704.96	656.24	669.44
16	2453.8	839.92	852.6	814.76	845.48
18	4666.4	1777.44	1815.6	1727.84	1765.16
20	*6264.71	1711.84	1701.6	1718.36	1703.88
25	*33919.5	7499.32	7498.12	7434.96	7276.4
30	*58459.1	16707.48	17618.48	16097.36	17154.4
40	*	96406.76	90747.6	93678.76	90951.56

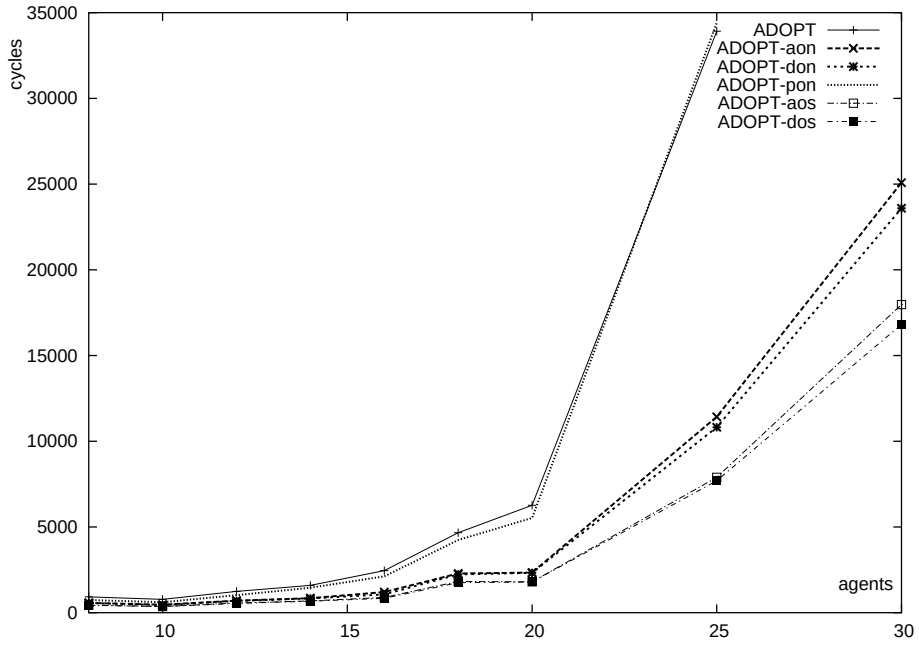

Figure 8: Longest causal chain of messages (cycles) used to solve versions of ADOPT using CAs, averaged over problems with density .3. Table entries containing * specify that the corresponding algorithm did not manage to solve all instances of that size in 2 weeks, and the eventually present value is based on the subset of problems solved in that time.

more optional **nogood** messages[4] than others. In the version ADOPT-pon, valued global nogoods are sent only to the parent of the current agent in a

[4]Messages to predecessors other than the previous agent (or parent agent for versions with DFS trees).

Agents	ADOPT	aos	Aos	dos	Dos
8	45.2	31.4	31.4	31.32	31.32
10	60.2	30.92	29.56	30.24	30.44
12	69.12	39.32	39.6	39.48	39.52
14	75.64	42.32	42.8	42.44	42.72
16	97.84	44.24	46.2	44.04	45.16
18	162.16	75.08	75.36	73.08	74.8
20	71.8	36.48	35.16	36.48	34.84
25	221.44	83.12	83.96	80.64	84.2
30	433.92	112.68	122.64	112.52	114.84
40	720.04	117.28	108.4	107.64	112.24

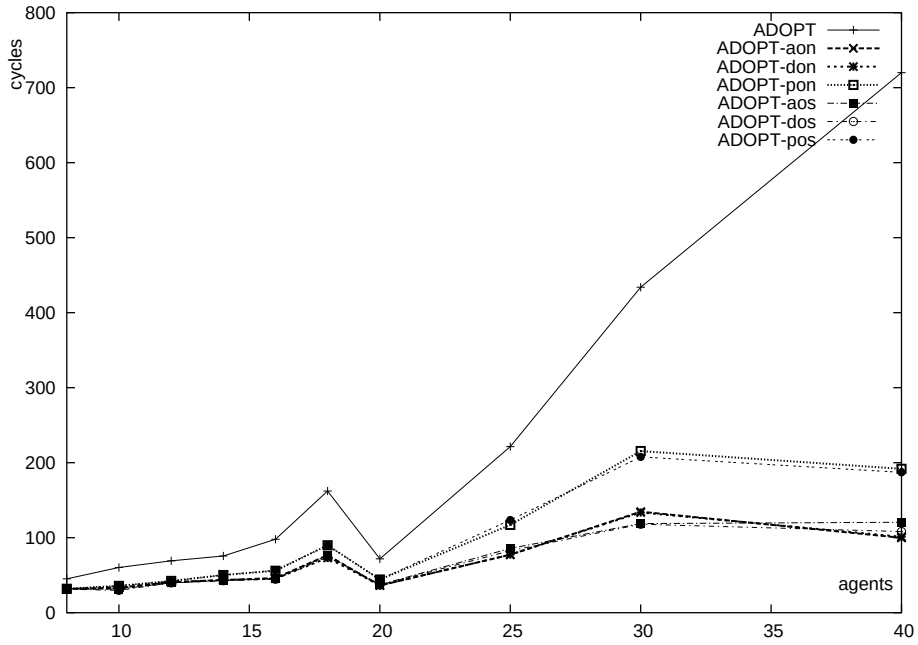

Figure 9: Longest causal chain of messages (cycles) used to solve versions of ADOPT using CAs, averaged on 25 problems with density .2.

maintained DFS tree. In ADOPT-don, each agent A_i tries to compute a valued global nogood after each change, for each of its ancestors A_j in the DFS tree, and sends it to A_j if the nogood is new and with non-zero threshold. ADOPT-aon can be seen as a version of ADOPT-don where the DFS tree is reduced to the linear list of agents (each having the predecessor as parent). ADOPT-Aon is a version of ADOPT-aos where an optional **nogood** message is sent only if the destination of the message is the same with the target of the nogood in the

Nodes	aos	Aos	dos	Dos
8	6137.8	5242.44	5381.04	4854.92
10	7330.6	5466.4	6191.92	5005.4
12	14201.8	9588.68	11566.92	8705.64
14	21981.96	14696.88	15760.4	12427.52
16	35710.8	22057.12	24552.24	19553.64
18	93368.6	50861.08	64610.96	44328.36
20	116468.8	56852.32	85127.44	49630.32
25	863145.12	350337.64	602437.08	291927.88
30	3640811.36	1137317.08	1853420	881049.76
40	49802812.56	9046121.88	22413986.4	7141719.28

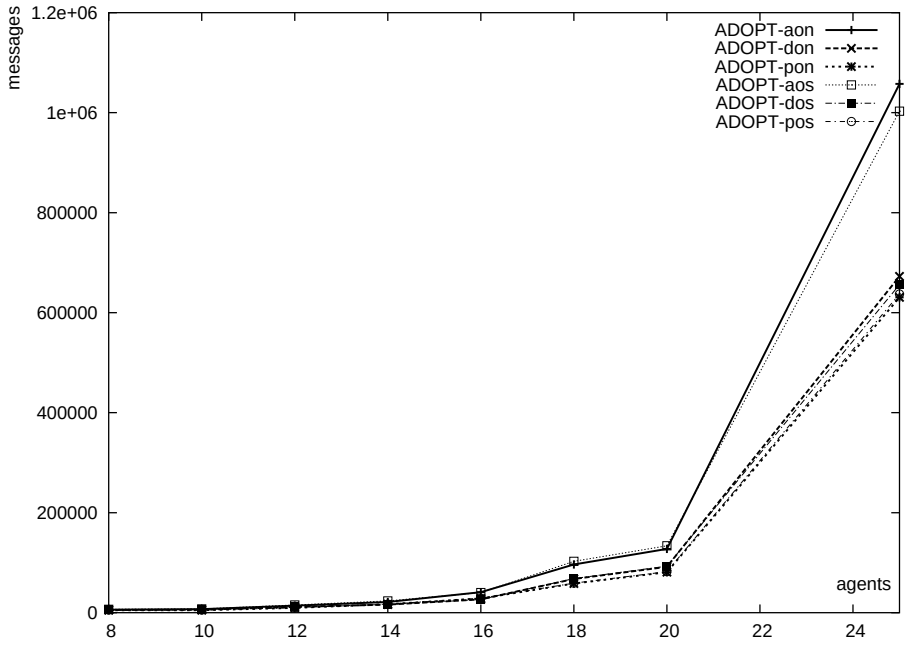

Figure 10: Total number of messages used by versions of ADOPT-ng using CAs to solve problems with density .3.

payload. The same holds for the relation between ADOPT-Don and ADOPT-don. ADOPT-pos, ADOPT-Dos, ADOPT-dos, ADOPT-Aos, and ADOPT-aos are variations of ADOPT-pon, ADOPT-Don, ADOPT-don, ADOPT-Aon, and ADOPT-aon where valued nogoods are used instead of valued global nogoods. We also experimented with versions of ADOPT-aos and ADOPT-dos where threshold valued nogoods are not used. This helps to isolate and evaluate the importance of threshold valued nogoods in ADOPT-ng.

Nodes	Aos	aos	dos	Dos
8	2	3	2	2
10	2	4	2	2
12	6	9	6	5
14	10	17	10	8
16	18	33	19	15
18	56	111	70	45
20	74	161	115	61
25	674	1615	1198	539
30	2889	8474	4907	2101

Figure 11: Total number of seconds used on a simulator by versions of ADOPT-ng, on the 25 problems with density .3.

Agents	16	18	20	25	30	40
ADOPT-aos	839.92	1777.44	1711.84	7499.32	16707.48	96406.76
no threshold	849.76	1783.6	1763.6	7641.84	16917.72	96406.64
ADOPT-dos	814.76	1727.84	1718.36	7434.96	16097.36	93678.76
no threshold	847.76	1779.6	1741.28	7500.04	16958.28	98932.72

Figure 12: Impact of threshold valued nogoods on the longest causal chain of messages (cycles) for versions of ADOPT-ng, averaged on problems with density .3.

Agents	16	18	20	25	30	40
DFS compatible	839.92	1777.44	1711.84	7499.32	$16*10^3$	$96*10^3$
random order	$461*10^3$	$1.5*10^6$	$3.7*10^6$	$48*10^6$	$128*10^6$	—

Figure 13: Impact of choice of order according to a DFS tree on the longest causal chain of messages (cycles) for versions of ADOPT-ng, averaged on problems with density .3.

The algorithms are compared on the same problems that are used to report ADOPT's performance in [21]. To correctly compare our techniques with the original ADOPT, we have used the same order (or DFS trees) on agents for each problem. The impact of the existence of a good DFS tree compatible with the used order is tested separately by comparison with a random ordering. The set of problems distributed with ADOPT and used here contains 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 numbers of agents it contains test sets with density .2 and 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 length of the longest causal (sequential) chain of messages of each solver, computed as the number of cycles of our simulator and averaged on problems with density .3, is given in Figure 8. Results for problems with density .2 are given in Figure 9. It took more than two weeks for the original ADOPT implementation to solve one of the problems for 20 agents and density .3, and one of the problems for 25 agents and density .3 (at which moment the solver was interrupted). Therefore, it was evaluated using only the remaining 24 problems at those problem sizes.

We can note that the use of valued nogoods brought an improvement of approximately 7 times on problems of density 0.2, and an approximately 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 cycles), than sending nogoods to several ancestors.

Figure 8 shows that, with respect to the number of cycles, the use of SRCs practically replaces the need to maintain the DFS tree since ADOPT-aos and ADOPT-Aos are comparable in efficiency with ADOPT-dos and ADOPT-Dos. SRCs bring improvements over versions with valued global nogoods, since SRCs allow detection of dynamically obtained independence.

Versions using DFS trees require fewer parallel/total messages, being more network friendly, as seen in Figure 10. Figure 10 shows that refraining from sending too many optional nogoods messages, as done in ADOPT-Aos and ADOPT-Dos, is comparable or even better than ADOPT-pon in terms of total number of messages, while maintaining the efficiency in cycles comparable to ADOPT-aos and ADOPT-dos.

We do not perform any run-time comparison with the original ADOPT 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). A comparison between the total times required by versions of ADOPT-ng on a simulator is shown in Figure 11. It reveals the computational load of the agents, which, as expected, is proportional to the total number of exchanged messages.

A separate set of experiments was run for isolating and evaluating the contribution of threshold valued nogoods. Figure 12 shows that the contribution of threshold nogoods is higher when a DFS tree is maintained, but still it is no more than 5%.

Another experiment, whose results are shown in Figure 13, is meant to evaluate the impact of the guarantees that the ordering on agents is compatible with some short DFS tree. We evaluate this by comparing ADOPT-aos with an ordering that is compatible with the DFS tree built by ADOPT, versus a random ordering. The results show that random orderings are unlikely to be compatible with short DFS trees and that verifying the existence of a short DFS tree compatible to the ordering on agents to be used by ADOPT-ng is highly recommended.

Figure 8 clearly show 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) is the use of SRCs. The use of the structures of the DFS tree makes slight improvements in number of cycles (when nogoods reach all ancestors) and slight improvements in total message exchange. To obtain a low total message traffic and to reduce computation at agent level, we found that it is best not to announce any possible valued nogoods to each interested ancestor. Instead, one can reduce the communication without a significant penalty in number of cycles by only announcing valued nogoods to the highest priority agent to which they are relevant (besides the communication with the parent, which is required for guaranteeing optimality).

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, d the domain size, and S the available computer memory.

8 Conclusions

The ADOPT distributed constraint optimization algorithm can be used efficiently (in number of cycles) without explicitly maintaining a DFS tree of the constraint graph. This can be done by using valued nogoods tagged with sets of references to culprit constraints. The generalized algorithm is denoted ADOPT-ng. Tagging costs with sets of references to culprit constraints (SRCs) allows detection and exploitation of dynamically created independence between subproblems. Such independence can be caused by assignments. Experimentation shows that it is important for an agent to infer and send in parallel several valued nogoods to different higher priority agents. It also shows that exaggerating this principle by sending each valued nogood to all ancestors able to handle it produces little additional gain while increasing the network traffic and the computational load. Instead, each inferred valued nogood should be sent only to the highest priority agent that can handle it (its target). DFS trees can still be used in conjunction with the valued nogood paradigm for optimization, thereby 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.

We isolated and evaluated the contribution of using threshold valued nogoods in ADOPT-ng, which was found to be at most 5%. In addition, we determined the importance of precomputing and maintaining a short DFS tree of the constraint graph, or at least of guaranteeing that a DFS tree is compatible with the order on agents, which is almost an order of magnitude in our problems.

The use of SRCs to dynamically detect and exploit independence and the generalized communication of valued nogoods to several ancestors bring elegance and flexibility to the description and implementation of ADOPT in ADOPT-ng. They also produced experimental improvements of an order of magnitude.

References

[1] S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *AAMAS*, 2005.

[2] A. Armstrong and E. F. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of 15th IJCAI*, 1997.

[3] M. Benisch and N. Sadeh. Examining dcsp coordination tradeoffs. In *AAMAS*, 2006.

[4] 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.

[5] A. Chechetka and K. Sycara. A decentralized variable ordering method for distributed constraint optimization. In *AAMAS*, 2005.

[6] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *AAMAS*, 2006.

[7] Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*, 2000.

[8] P. Dago. Backtrack dynamique valué. In *JFPLC*, pages 133–148, 1997.

[9] J. Davin and P. J. Modi. Impact of problem centralization in distributed cops. In *DCR*, 2005.

[10] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *AI'90*, 1990.

[11] R. Dechter. *Constraint Processing*. Morgan Kaufman, 2003.

[12] W. R. Evan Sultanik, Pragnesh Jay Modi. Constraint propagation for domain bounding in distributed task scheduling. In *CP*, 2006.

[13] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward-bounding for distributed constraints optimization. In *ECAI*, 2006.

[14] M. L. Ginsberg. Dynamic backtracking. *Journal of AI Research*, 1, 1993.

[15] R. Greenstadt, J. Pearce, E. Bowring, and M. Tambe. Experimental analysis of privacy loss in dcop algorithms. In *AAMAS*, pages 1024–1027, 2006.

[16] Y. Hamadi and C. Bessière. Backtracking in distributed constraint networks. In *ECAI'98*, pages 219–223, 1998.

[17] 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.

[18] J. Liu and K. P. Sycara. Exploiting problem structure for distributed constraint optimization. In *ICMAS*, 1995.

[19] 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.

[20] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–445, 2004.

[21] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *AIJ*, 161, 2005.

[22] 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.

[23] A. Petcu and B. Faltings. Approximations in distributed optimization. In *Principles and Practice of Constraint Programming CP 2005*, 2005.

[24] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, 2005.

[25] A. Petcu and B. Faltings. Odpop: An algorithm for open/distributed constraint optimization. In *AAAI*, 2006.

[26] G. Ringwelski and Y. Hamadi. Multi-directional distributed search with aggregation. In *IJCAI-DCR*, 2005.

[27] M.-C. Silaghi. *Asynchronously Solving Distributed Problems with Privacy Requirements*. PhD Thesis 2601, (EPFL), June 27, 2002. `http://www.cs.fit.edu/~msilaghi/teza`.

[28] M.-C. Silaghi. Asynchronous PFC-MRDAC±Adopt —consistency-maintenance in ADOPT—. In *IJCAI-DCR*, 2003.

[29] M.-C. Silaghi. Howto: Asynchronous PFC-MRDAC –optimization in distributed constraint problems +/-ADOPT–. In *IAT*, Halifax, 2003.

[30] M.-C. Silaghi. Framework for modeling reordering heuristics for asynchronous backtracking. In *IAT*, 2006.

[31] M.-C. Silaghi and B. Faltings. A comparison of DisCSP algorithms with respect to privacy. In *AAMAS-DCR*, 2002.

[32] M.-C. Silaghi and B. Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence Journal*, 161(1-2):25–53, October 2004.

[33] 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.

[34] 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.

[35] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. ABT with asynchronous reordering. In *IAT*, 2001.

[36] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Consistency maintenance for ABT. In *Proc. of CP'2001*, pages 271–285, Paphos,Cyprus, 2001.

[37] 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.

[38] 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.

[39] R. Wallace and M.-C. Silaghi. Using privacy loss to guide decisions in distributed CSP search. In *FLAIRS'04*, 2004.

[40] M. Yokoo. Constraint relaxation in distributed constraint satisfaction problem. In *ICDCS'93*, pages 56–63, June 1993.

[41] 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.

[42] 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.

[43] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of 3rd ICMAS'98*, pages 372–379, 1998.

[44] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.

[45] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proc. of AAAI*, Edmonton, July 2002.

[46] R. Zivan and A. Meisels. Dynamic ordering for asynchronous backtracking on discsps. In *CP*, pages 161–172, 2005.