DFS ordering in Nogood-based Asynchronous Distributed Optimization (ADOPT-ng)

Marius C. Silaghi[†] **and Makoto Yokoo**[‡]
[†]**Florida Institute of Technology**
[‡]**Kyushu University**

Abstract. This work proposes an asynchronous algorithm for solving Distributed Constraint Optimization problems (DCOPs) using a generalized kind of nogoods, called 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.

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 besides bringing significant speed-up, valued nogoods allow for automatically detecting and exploiting DFS trees compatible with the current ordering since independent subproblems are now dynamically detected and exploited (DFS trees do not need to be specified/computed explicitly). However, not all possible orderings on variables are compatible with good DFS trees, and we find that on randomly ordered problems ADOPT-ng runs orders of magnitude slower than on orderings that are known to be compatible with short DFS trees.

Being an extension of ABT, ADOPT-ng can also profit of the dynamic ordering heuristics enabled by Asynchronous Backtracking with Reordering (ABTR). However, our experiments imply that efficient dynamic ordering heuristics for ADOPT-ng will have to maintain compatibility with some DFS tree (e.g., to be decided by rebuilding DFS trees based on current search state).

Experiments comparing ADOPT-ng with Valued Dynamic Backtracking show that ADOPT-ng also brings significant improvements over the old valued nogood-based algorithm.

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 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 [27].

Several synchronous and asynchronous distributed algorithms have been proposed for distributedly solving DCOPs. 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 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 for addressing privacy requirements [18, 26, 30, 20].

A synchronous algorithm, synchronous branch and bound, was the first known distributed algorithm for solving DCOPs [9]. Stochastic versions have also been proposed [31]. 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 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 [28]. Within the mentioned assumption, this measure is equivalent for real solvers to the longest causal chain of sequential messages (i.e., logic time), as used in [22].

From the point of view of this measure, a very efficient currently existing DCOP solver is DPOP [14, 13], 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-first 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) [11]. DisAO works by incrementally joining the subproblems owned by agents found in conflict. ADOPT implements a parallel version of A* [19]. 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 above assumptions like DPOP [6, 10, 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 general type of nogoods, called *valued nogoods* [5], that not only virtually eliminates the need of knowing 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 [25]. 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. Each of these incremental concepts and improvements is described in the following sections.

2 DFS-trees

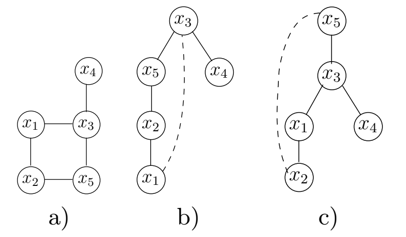

Figure 1. For a DCOP with primal graph depicted in (a), two possible DFS trees (pseudotrees) are (b) and (c). Interrupted lines show constraint graph neighboring relations not in the DFS tree.

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 [7]. A Depth First Search (DFS) tree associated to a DCOP is a spanning tree generated by the arcs used for entering 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.

3 ADOPT

ADOPT [12] 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 [12] for the detail. 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 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.

4 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.*

The problem is to find $\operatorname{argmin}_x \sum_{i=1}^{n} f_i(x_{|X_i})$. We assume that x_i can only take values from a domain $D_i = \{1, ..., d\}$.

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

Our idea can be easily applied to general valued CSPs.

5 Cost of nogoods

A generalization of the nogoods concept to the case of valued CSPs was first proposed in [5]. Here we reintroduce the main definitions in the versions appearing in [24].

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 $[c, (\langle x_1, v_1\rangle, ..., \langle x_t, v_t\rangle)]$, 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 global value nogood, (i.e. the inference):

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

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, $\overline{N} = \{x_1, ..., x_t\}$.

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:* $[\min_v c_v, \cup_v S_v]$.

Definition 3 (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.*

Definition 4 (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 the assignment of x_i to some value v.*

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 $N{=}\cup_i N_i$.*

Proposition 3 (min-resolution) *A set of cost assessments for x_i, (SRC_i, v_i, c_i, N_i) where $\cup_i\{v_i\}$ covers 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 $N{=}\cup_i N_i$.*

Remark 1 *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., $x \notin \overline{N}$.*

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

6 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).

6.1 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). 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 (local) 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 CAs as described in Remark 1.
- lastSent[k] stores the last valued nogood sent to A_k.

6.2 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 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 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, 23]). Function $vn2ca(vn, i)$ transforms a valued nogood vn 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). Notation $lr[t]_{|v}$ stands for vn2ca(lr[t]) when lr[t]'s value for x_i is v, and $\emptyset$ otherwise.

Remark 2 *The order of combining CAs matters. 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]);
   b) Else, CAs coming from each agent A_t are considered: tmp[v][t]=ca[v][t];
2. *CAs from step 1 (a or b) are combined: h[v]=sum-inference(tmp[v][*]); for (b), start with t=i+*1
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 and A_n the lowest priority. Each agent A_i starts by calling the init() procedure, which initializes its l with valued nogoods infered 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 correponding 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, or ca. A_i always sets its *crt_val* to the index with the lowest CA threshold in vector h (on ties 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 [29].

The threshold valued nogood tvn delivered with **ok?** messages sets a common cost on all values of the receiver (see Remark 1), 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 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: changes to ca are accepted only when nogoods' thresholds increase.

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

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 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 assignment's CA threshold increases. By construction, thresholds for CAs associated to the values of A_{n-k} can only grow. 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. □

```
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;
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;
        lvn[k][i]:=sum-inference(tvn,lvn[k][i]);
    check-agent-view();
when receive add-link(⟨x_j, v_j⟩),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;
    if ((j := target(rvn)) ≠ i) then
        lvn[j][t]:=rvn; values = D_i;
        vn2ca(rvn, i) → CA rca for any value of x_i;
    else
        vn2ca(rvn, i) → CA rca for a value μ of x_i;
        values = {μ};lr[t]:=rca;
    foreach (value v in values) do
        if v == crt_val then
            either: ca[v][t]:=sum-inference(rca,ca[v][t]);
                or ca[v][t]:=rca;
            optional: // used in our experiments
            update h[v][t] and retract changes if h[v][t]'s cost
            decreases;
        else
            optional: // used in our experiments
            ca[v][t]:=sum-inference(rca,ca[v][t]);
            update h[v][t] and retract changes if h[v][t]'s cost
            decreases;
    check-agent-view();
```

Algorithm 1: Procedures of A_i for receiving messages in ADOPT-ng. Subroutines are described in Algorithm 2.

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

```
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
            vn:=min_resolution(j);
            if ($vn \neq lastSent[j]$) then
                send nogood(vn,i) to $A_j$;
    crt_val=$argmin_v(cost(h[v]))$;
    if (crt_val changed) then
        send ok?($\langle x_i, crt\_val \rangle$,
                ca2vn(sum-inference(lvn[*][k],ca[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 $x_j$;
    use $lr[t]_{|v}$ and lvn to replace discarded ca;
    store $\langle x_j, v_j \rangle$ in agent view;
```

Algorithm 2: Procedures of A_i in ADOPT-ng

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).

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 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 2 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 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. □

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

6.3 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 [8]. 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, 23].

6.4 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 its 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 [12], 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 [15, 16].

6.5 Dynamic Ordering of agents/variables

ADOPT-ng can be seen as an extension of ABT. The extension of ABT called ABTR [21, 17] proposes a way to extend ABT-based algorithms to allow for dynamic ordering on agents variables. ABTR was defined as a simple family of algorithms that can be parametrized with any complex dynamic ordering heuristic that respects certain simple guidelines. Aheuristic in ABTR is defined by a very general framework described by a tuple ($\mathbb{K}$,$\mathbb{S}$,$\mathbb{M}$,$\mathbb{P}$,$\mathbb{H}$):

$\mathbb{K}$: a knowledge domain of interest for the heuristic (e.g., current domain sizes, existence of a current domain wipe-out, positions of agents).

$\mathbb{S}$: a set of integers. Only its intersection with $[0..(n-2)]$ is relevant.

$\mathbb{M}$: a policy dynamically mapping a set of counters, $\{C_k^r | k \in \mathbb{S}\}$, to the n agents as function of $\mathbb{K}$ (e.g., the counter C_k^r goes to the agent on position k).

$\mathbb{P}$: an ordering policy for each agent that holds a counter C_k^r. It proposes a given ordering of the last $n-k$ agents as function of $\mathbb{K}$. This reordering (new $\mathbb{K}$), when projected through $\mathbb{M}$, should impact only the mapping of counters $\{C_i^r | i \geq k\}$.

$\mathbb{H}$: a set of rules specifying when an agent that holds a counter C_k^r may use (or be told) new data from $\mathbb{K}$ for proposing a new order.

In ABTR, the counters C_k^r counts the reordering proposals for the last $n-k$ agents (made by the agent holding it) and all messages are tagged with the current order and with the (signature) vector clock induced by the value of the counters known to the sender. ABTR's proof guarantees that any heuristic that can be described with this general framework leads to a sound, complete and terminating asynchronous distributed algorithm if it respects the additional constraint on $\mathbb{H}$ that *the delay between the moment the last* **ok?** *message was sent by agents $A^k, k \leq i$, (or from start) and any subsequent reordering request sent by the owner of C_i^r,* ***must be*** *finite.* Such heuristics are called *ABTR compliant*.

Our experiments imply that most efficient dynamic ordering heuristics for ADOPT-ng will have to maintain compatibility with short DFS trees. For example, one can define a heuristic *dynamic-DFS* that rebuilds DFS sub-trees of the current node with a greedy approach where the next neighbor is selected using the current state of the search (domain sizes after enforcing weighted arc-consistency (WAC*)). Finding good heuristics was shown to be a difficult problem [23, 32].

7 Experiments

We implemented several versions of ADOPT-ng, differing by how the agents picks the targets of their nogoods. In the implementation 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 ancestors 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. These are early versions that do not yet exploit threshold nogoods in **ok?** messages.

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 [12]. 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 numbers of agents it contains test sets with density .2 and with density .3. 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 2. Results for problems with density .3 are given in Figure 3. 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 cycles),

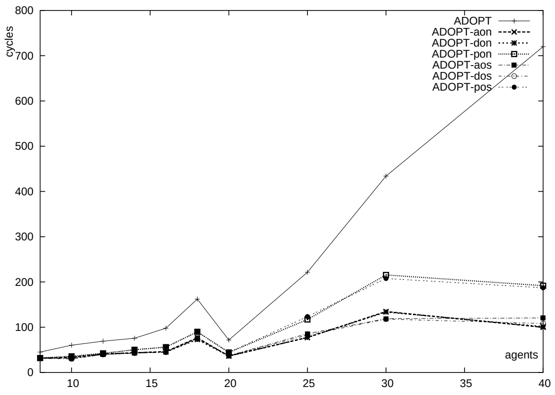

Figure 2. Longest causal chain of messages (cycles) used to solve versions of ADOPT using CAs, on problems with density .2.

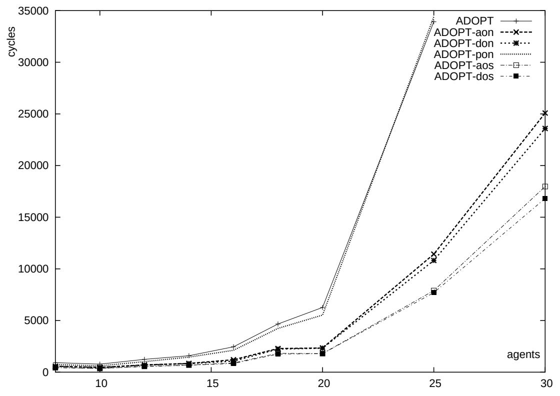

Figure 3. Longest causal chain of messages (cycles) used to solve versions of ADOPT using CAs, on problems with density .3.

than sending nogoods to all ancestors. Versions using DFS trees require less parallel/total messages, being more network friendly, as seen in Figure 4.

Figure 3 shows that, with respect to the number of cycles, the use of SRCs practically replaces the need of knowing the 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 detection of 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 3 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.

Additional experiments were executed where we compared the performance of ADOPT-ng on randomly ordered problems versus the performace when the static ordering of the variables was chosen to be compatible with a precomputed DFS tree. Results show 15 times improvements for the case where compatibility with a DFS tree order was enforced. Also,

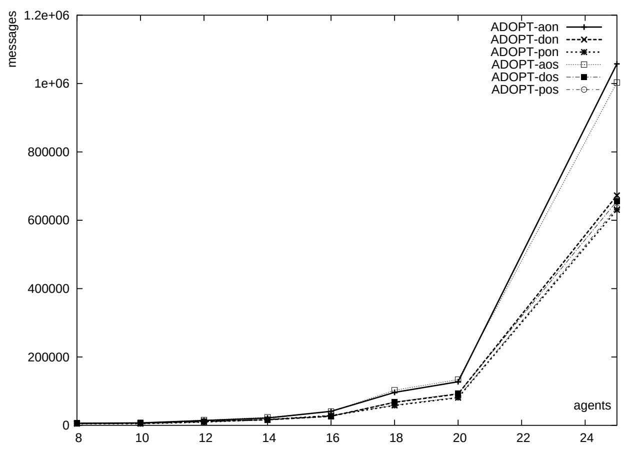

Figure 4. Total number of messages used by versions of ADOPT-ng using CAs to solve problems with density .3.

according to experimental results, the improvements brought by threshold nogoods are minor in ADOPT-ng (approx 1probably because the stored nogoods of the agent already maintain much of that information. Experiments were also run comparing ADOPT-ng with Valued Dynamic Backtracking [4], and ADOPT-ng is shown to be several times better. Due to lack of space, detailed graphs with all results will be given only in an extended version.

8 Conclusions

We propose a generalization of the ADOPT algorithm, denoted ADOPT-ng, based on valued nogoods. References to culprit constraints (SRCs) allow detection and exploitation of dynamically created independences between subproblems (that are due to some assignments).

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. The compatibility of the current order on variables with a good (short) DFS tree is shown important and this provides significant hints towards promising ABTR-compliant dynamic heuristics to be explored in the extension of ADOPT-ng with the reordering of ABTR.

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

[5] P. Dago and G. Verfaillie. Nogood recording for valued constraint satisfaction problems. In *ICTAI*, pages 132–139, 1996.

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

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

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

[9] 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.

[10] 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.

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

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

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

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

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

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

[17] M.-C. Silaghi. Generalized dynamic ordering for asynchronous backracking on discsp. In *submitted to CP*, 2006.

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

[19] 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.

[20] 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.

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

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

[23] 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.

[24] M.-C. Silaghi and M. Yokoo. Nogood-based asynchronous distributed optimization (ADOPT-ng). In *AAMAS (to appear)*, 2005.

[25] 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.

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

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

[28] 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.

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

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

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

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