DJAO: A Communication-Constrained DCOP Algorithm that Combines Features of ADOPT and Action-GDL

Yoonheui Kim and **Victor Lesser**
University of Massachusetts at Amherst
MA 01003, USA
{ykim,lesser}@cs.umass.edu

Abstract

In this paper we propose a novel DCOP algorithm, called DJAO, that is able to efficiently find a solution with low communication overhead; this algorithm can be used for optimal and bounded approximate solutions by appropriately setting the error bounds. Our approach builds on distributed junction trees used in Action-GDL to represent independence relations among variables. We construct an AND/OR search space based on these junction trees. This new type of search space results in higher degrees for each OR node, consequently yielding a more efficient search graph in the distributed settings. DJAO uses a branch-and-bound search algorithm to distributedly find solutions within this search graph. We introduce heuristics to compute the upper and lower bound estimates that the search starts with, which is integral to our approach for reducing communication overhead. We empirically evaluate our approach in various settings.

Introduction

In this paper, we formulate a new algorithm for distributed constraint optimization problems (DCOPs), called DJAO, which works on a distributed junction tree (Paskin, Guestrin, and McFadden 2005). DJAO operates in two phases. In the first phase, heuristic upper and lower bounds for variable value configurations are created using a bottom-up propagation scheme similar in character to Action-GDL (Vinyals, Rodriguez-Aguilar, and Cerquides 2011). Except that instead of transmitting values for all configurations, we transmit only the filtered upper and lower bounds of configuration values. The next phase using these heuristics conducts an ADOPT-like (Modi et al. 2005; Yeoh, Felner, and Koenig 2008) search on AND/OR search graph based on the junction tree, which we call AND/OR search junction graph, to find a solution with desired precision. This two-phase strategy reduces overall communication significantly.

AND/OR search tree and context-minimal AND/OR search graph

An AND/OR search space (Marinescu and Dechter 2005) is introduced to exploit independencies encoded by the graphical model upon which DCOP algorithms such as Action-GDL and Max-Sum (Farinelli et al. 2008) are constructed. An AND/OR search tree is a search space with additive AND nodes whose subtrees denote disjoint search spaces under different variables in addition to OR nodes in traditional search trees whose subtrees denote disjoint search spaces under values of variables. AND nodes decompose the search space in their subtrees under Generalized Distributive Law framework (Aji and McEliece 2000). It reduces the size of DCOP search space from $O(\exp(n))$ to $O(n \cdot \exp(m))$, where m is the depth of the pseudo-tree (Koller and Friedman 2009) and n is the number of variables. DCOP algorithms such as ADOPT (Modi et al. 2005) and BnB-ADOPT (Yeoh, Felner, and Koenig 2008) can be viewed as distributed search algorithms on this AND/OR search space.

Definition 1 (AND/OR search tree) *Given a COP instance P, its primal graph G and a pseudo-tree T of G, the associated AND/OR search tree $S_T(P)$ has alternating levels of OR nodes and AND nodes. The OR nodes are labeled X_i and correspond to variables. The AND nodes are labeled $\langle X_i, a\rangle$ and correspond to value assignments in the domains of variables. The root of the AND/OR search tree is an OR node, labeled with the root of T. The children of an OR node X_i are AND nodes labeled with assignments $\langle X_i, a\rangle$, consistent along the path from the root. The children of an AND node $\langle X_i, a\rangle$ are OR nodes labeled with the children of variable X_i in T. The path of a node $n \in S_T$, denoted $Path_{S_T}(n)$, is the path from the root of S_T to n, and corresponds to a partial value assignment to all variables along the path.*

An example of AND/OR search tree is given in Fig. 1a. Because AND nodes decompose the problem into separate subproblems, variables in different subtrees of an AND node n are considered independently given the value assignment along the path to n. The arcs in S_T are annotated by appropriate *labels* of the cost functions.

Definition 2 (label) *The label $l(X_i, \langle X_i, a\rangle)$ is defined as the sum of all the cost function values for which variable $\mathbf{X}_i$ is contained in their scope and whose scope is fully assigned along the path from root to n.*

Definition 3 (value) *The value $v(n)$ of a node $n \in S_T$, is defined recursively as follows: (i) if $n = \langle X_i, a\rangle$ is a terminal AND node then $v(n) = l(X_i, \langle X_i, a\rangle)$; (ii) if $n =$*

$\langle X_i, a\rangle$ is an internal AND node then $v(n)=(X_i, \langle X_i, a\rangle) + \sum_{n' \in succ(n)} v(n')$; (iii) if $n = X_i$ is an internal OR node then $v(n) = \max_{n' \in succ(n)} v(n')$, where $succ(n)$ are the children of n in S_T.

In (Dechter and Mateescu 2007), the AND/OR search graph shown in Fig. 1c was introduced to reduce the size of the search tree in Fig 1a by merging two nodes that root identical subtrees. A **Context-based merge** operation is defined as either i) merging two OR nodes that share the same variable assignments on the ancestors of these nodes, which have connections in G to these nodes or their descendants, or ii) merging two AND nodes that share assignments on these nodes and ancestors of nodes, which have connections in G to these nodes' descendants.

Example For the primal graph G in Fig. 1b and a search tree in Fig. 1a for G, OR nodes for D that shares assignments for B can be merged as B is connected to D in G. In contrast, OR nodes for C cannot be merged as the assignments for A and B should match. AND nodes at the lowest level can be merged if these nodes share the assignments as these nodes do not have any descendant.

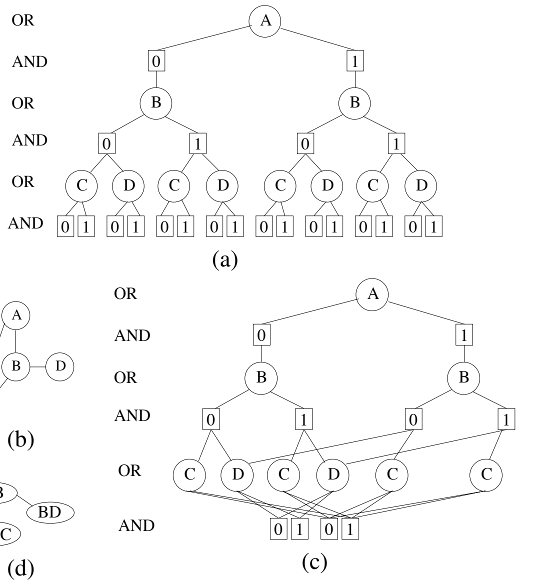


Figure 1: an AND/OR search graph

Definition 4 (context minimal AND/OR graph) *The AND/OR search graph of G that is closed under context-based merge operator is called a context minimal AND/OR search graph.*

Distributed Constraint Optimization and Junction-Tree

A distributed constraint optimization problem (DCOP) instance $P = \langle \mathbf{A}, \mathbf{X}, D, \mathbf{F}\rangle$ is formally defined by the following parameters:

- A set of variables $\mathbf{X} = \{X_1, \ldots, X_r\}$, where each variable has a finite domain D (maximum size N) of possible values that it can be assigned.
- A set of constraint functions $\mathbf{F} = (F_1, \ldots, F_k)$, where each constraint function, $F_j : \mathbf{X}_j \rightarrow \Re$, takes as input any setting of the variables $\mathbf{X}_j \subseteq \mathbf{X}$ and provides a real valued utility.

In DCOP, we assume that each variable x_i is owned by an agent $a_i \in \mathbf{A}$ and that an agent only knows about the constraint functions in which it is involved. The DCOP can be represented using a *constraint network*, where there is a node corresponding to each variable x_i and where there is an edge (hyper-edge) for each constraint F_j that connects all variables that are involved in the function F_j.

The objective in the DCOP is to find the complete variable configuration **x** that maximizes $\sum_{F_j \in \mathbf{F}} F_j(\mathbf{x}_j)$.

The *dual constraint graph* (Koller and Friedman 2009) is a transformation of a non-binary network into a special type of binary network. It contains cliques (or c-variables) domains of which ranges over all possible value combinations permitted by the corresponding constraint functions, and shared variables in any two adjacent cliques have same values.

A junction tree (or join tree) (Cowell et al. 2007) T is a subgraph of the dual graph which is a tree and satisfies the condition that cliques associated with a variable x form a connected subset of T. A *Junction tree* is represented as a tuple $\langle \mathbf{X}, \mathbf{C}, \mathbf{S}, \mathbf{F}\rangle$. where $\mathbf{X}$ is a set of variables, $\mathbf{C}$ is a set of cliques, where each clique C_i is a subset of variables $C_i \subseteq \mathbf{X}$; $\mathbf{S}$ is a set of separators, where each separator is an arc between two adjacent cliques containing their intersection; and $\mathbf{F}$ is a set of potentials, where each potential in $\mathbf{F}$ is assigned to each clique in $\mathbf{C}$.

A distributed junction tree (Paskin, Guestrin, and McFadden 2005) decomposes a DCOP into a series of subproblems, some of which can be solved in parallel. A subproblem represented as a clique $c_i \in \mathbf{C}$ can be solved independently given the local constraint functions $f_i \in \mathbf{F}$ and the values from neighbors on separator $s_i \in \mathbf{S}$. Separators $\mathbf{S}$ specify which values will be used in the neighboring cliques in order to compute the solution for its local subproblem.

DJAO(k)

First Phase: Heuristics Generation

Preprocessing techniques to supply the search with heuristic values has successfully been used to enhance both centralized and decentralized search methods. (Ali, Koenig, and Tambe 2005; Wallace 1996). In this section we describe a scheme for generating initial heuristic estimates h_{UB} and h_{LB} used in $DJAO(k)$, based on a new function filtering technique, which we call Soft Filtering, described here. (Pujol-Gonzalez et al. 2011; Brito and Meseguer 2010) used the Function Filtering technique (Brito and Meseguer 2010) on DCOPs to prune variable configurations of local nodes, that do not yield the optimal solution. We use the soft function filtering technique to generate heuristics that maintains the tuples that potentially yield the optimal solution while summarizing the rest with upper and lower bounds. Unlike the heuristics in (Ali, Koenig, and Tambe 2005; Wallace 1996) which are generated by solving lower complexity problems than the original, DJAO solves the original problem and focuses on reducing communication by filtering tuples that are unlikely to be part of the optimal solution.

The Soft Filtering technique used in DJAO summarizes constraint functions to reduce communication required for transmitting such function. A simple difference from Function Filtering is that the Soft Filtering technique provides summarized lower and upper bounds on filtered configurations. Let the variable configuration S in message m be divided into two sets filtered configurations S_F, and non-filtered configurations S_{NF}. Let UB be the upper bounds of values on variable configurations and LB the lower bounds. The values UB_m and LB_m in the messages are filtered as follows.

$$UB_m(v) = \begin{cases} UB(v) & \text{if } v \in S_{NF} \\ \max_{S_F} UB(v) & \text{if } v \in S_F \end{cases}$$

LB_m is similarly defined with $\max$ replaced with $\min$.

Filtered configurations are summarized as a *filtered* tuple with a single upper and lower bounds, therefore reducing the number of items in each message from $\|2S\|$ to $2\|S_{NF}\|+2$. The optimal strategy is guaranteed to remain in the search space as no solution is completely dropped. This summarization builds a basis for the next phase where an ADOPT-like search finds a solution within a desired accuracy. Among many ways to select which items to filter, we select items in the bottom $(100 - d)\%$ of the function range.

Second Phase: Search on AND/OR junction graph

On the pseudo-tree based search graph, functions are evaluated only when their scope is fully assigned along the path. The search backtracks to evaluate different variable assignment which occurs among the domain of a single variable at each level. This complete decentralization in value selection in the distributed setting results in the exponential number of messages in ADOPT. Instead, we introduce AND/OR search graph on a junction tree where each level is associated with each clique in the junction tree in a DCOP (See Fig. 2), consequently yielding a more compact search graph with a lower number of nodes. This search graph is a context-minimal AND/OR search graph upon construction.

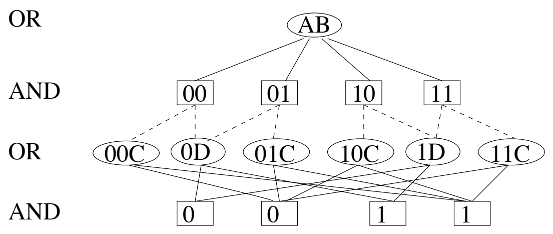


Figure 2: an AND/OR search graph based on a junction tree

Definition 5 (AND/OR search junction graph) *Given a DCOP instance P and its junction tree T, the associated AND/OR search junction graph $S_T(P)$ has alternating levels of OR nodes and AND nodes. The OR nodes are labeled $C_i : \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle$ where $\mathbf{a}$ are variables assignments in the domains of variables in separators $\mathbf{S}_i$ whose value are propagated from ancestors and newly appeared variables $\mathbf{N}_i$ in clique C_i. These OR nodes correspond to the cliques with partial assignment. The AND nodes are labeled $\langle \mathbf{S}_{ij}, \mathbf{b} \rangle$ and correspond to value assignments in the domains of the separator between clique C_i and its child C_j. The root of the AND/OR search graph is an OR node, labeled with the root of T. The children of an AND node $\langle \mathbf{S}_{ij}, \mathbf{b} \rangle$ are OR nodes who are labeled with C_j:$\langle \mathbf{S}_j, \mathbf{b}, \mathbf{N}_j \rangle$ with the same assignment on variables in separators $\mathbf{S}_{ij}$ and $\mathbf{S}_j$.*

Example Consider the graphical model in Fig. 1b describing a graph coloring problem over domains {0,1}. An AND/OR search graph based on a possible pseudo-tree is given in Fig 1c and an AND/OR search junction graph in Fig 1d is given in Fig. 2. Observe that the function evaluation on $l(\{A, B\}, a)$ occurs at the expansion of nodes at level 3 in Fig. 1c instead of at level 1 in Fig. 2. On Fig. **??**fig:aograph], the search is unguided until the third expansion and it also elongates the backtrack path leading to an increase in the number of visited nodes to evaluate a single function.

Theorem 1 *Given a DCOP instance P and a junction tree T, its AND/OR search junction graph is sound and complete. It contains all and only solutions.*

The solution space in AND/OR search junction graph is identical to the junction tree, thus it contains all and only solutions. Consequently, any search algorithm that traverses the AND/OR search junction graph in a depth-first manner is guaranteed to have a time complexity equal to the time complexity of Action-GDL (Vinyals, Rodriguez-Aguilar, and Cerquides 2011) on the same junction tree which is exponential in the tree width.

Theorem 2 *The size of search junction graph has exactly same size as the total complexity of junction tree as no subtree is redundant. The depth of the graph does not exceed the number of agents.*

The search result for its subtree is stored at each node, therefore no identical subtree is explored twice and a value assignment on a cost function is never repeated. The arcs in S_T are annotated by appropriate *labels* of the cost functions. The nodes in S_T are associated with a *value*, accumulating the result of the computation resulted from the subtree below. $labels$ on the arcs are determined by values from neighboring nodes unlike the one defined in Def. **??**def:value].

Definition 6 *label: The label $l(C_i$:$\langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle, \langle \mathbf{S}_{ij}, \mathbf{b} \rangle)$ of the arc from the OR node to the AND node $\langle \mathbf{S}_{ij}, \mathbf{b} \rangle$ is defined as the cost function values contained in the clique C_i whose scope is fully assigned with values from the parent OR node and and child AND node.*

The **value** of $v(n)$ of a node $n \in S_T(P)$ is computed in the same way as in Def. 3. Likewise, the value of each node can be recursively computed from leaves to root. We can show that:

Proposition 1 *Given an AND/OR search graph $S_T(P)$ of a DCOP instance, the value function $v(n)$ is the maximum cost solution to the subproblem rooted at n, subject to the current variable instantiation along the path from root to n. If n is the root of S_T, then v(n) is the maximum cost solution to P.*

We prove optimality by verifying the value of nodes are identical to the values produced during the execution of Action-GDL. The $value$ of an AND node is identical to the value of corresponding assignments in the messages from the corresponding clique of Action-GDL given the context along the search path to these nodes. Valuation of OR nodes is the combination of local utility functions and values of its child nodes and corresponds to the maximum achievable value of variable assignments given the variable assignments along the search path.

Proposition 2 *AND/OR search junction graph $S_T(P)$ is context-minimal upon construction*

The separators $\mathbf{S}_i$ and $\mathbf{S}_{ij}$ contain all and only variables that build a context for each node and a single node is created for each value assignment in the separator, thus it is context-minimal.

Search in distributed settings Each agent in the system distributedly conducts its share of search for the nodes on $S_T(P)$ it owns. Agents are responsible for valuation of owned nodes and path determination.

Definition 7 *Agent ownership: Each node in $S_T(P)$ is owned by an agent. Agent A_i owns all nodes associated with its own clique C_i: OR nodes $C_i : \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle$ and child AND nodes of these are assigned to A_i.*

For example, suppose clique AB, ABC and BD in Fig 1d are owned by agent A_1, A_2, and A_3 respectively. OR nodes of clique BD are $C_3 : \langle B, 0, D \rangle, C_3 : \langle B, 1, D \rangle$. These OR nodes and child AND nodes of these are assigned to A_3.

Search procedure between nodes belonging to different agents incurs communication. When an agent A_j chooses to expand a child OR node $C_i : \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle$, A_j transmits partial assignments $\mathbf{a}$ to an agent A_i who owns the child nodes. Updated function values are sent back to A_j when the search backs up. Search paths that incur communication are displayed as dotted lines in Fig. 2.

DJAO on AND/OR search junction graph

If each node $n \in S_T(P)$ is assigned a *heuristic lower-bound estimate LB(n)* and *heuristic upper-bound estimate UP(n)*, then we can calculate the lower and upper bound estimates of assignments and dominated search space can be pruned.

Bounds on Partial Solution Similarly to (Marinescu and Dechter 2005), a partially expanded search graph, denoted as *PSG*, contains the root node, will have a *frontier* containing all the nodes that were generated but not expanded. Each expansion of a leaf node on the search tree updates the lower and upper bound estimates on AND/OR search graph. An active partial subtree $APT(n)$ rooted at a node n ending at a tip node t contains the path between n and t, and all OR children of AND nodes on the path. A dynamic heuristic function of a node n relative to the current *PSG* given the initial heuristic functions h_{UB} and h_{LB} can be computed.

Definition 8 (Dynamic Lower and Upper Bound) *Given an active partial tree $APT(n)$, the dynamic heuristic estimate of upper and lower bound function, UB(n) and LB(n), is defined recursively as follows: (i) if there is a single node n in $APT(n)$ and is evaluated, then $UB(n) = v(n) = LB(n)$ else if n is a single node in APT $UB(n) = h_{UB}(n)$ and $LB(n) = h_{LB}(n)$; (ii) $n = \langle \mathbf{S}_{ij}, \mathbf{b} \rangle$ is an AND node, having OR children $m_1, \ldots, m_k$, and $label = l(C_i : \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle, \langle \mathbf{S}_{ij}, \mathbf{b} \rangle)$, then*

$$UB(n) = \min(h_{UB}(n), label + \textstyle\sum_{i=1}^{k} UB(m_i))$$
$$LB(n) = \max(h_{LB}(n), label + \textstyle\sum_{i=1}^{k} LB(m)) \;;$$

(iii) if $n = C_i : \langle P_i, \mathbf{a}, N_i \rangle$ where n is an OR node, having an AND child m, then $UB(n) = \min(h(n), UB(m))$ and $LB(n) = \max(h(n), LB(m_i))$.

Theorem 3 *$LB(n)$ is a lower bound on the optimal solution to the subproblem rooted at n, namely $LB(n) \leq v(n)$, and also by definition $LB(n) \geq h_{LB}(n)$. Also, $UB(n) \geq v(n)$ and $UB(n) \leq h_{LB}(n)$.*

Proof: We will prove by induction assuming the correctness of heuristics that $v(n) \leq h_{UB}(n), v(n) \geq h_{LB}(n)$. **Basis:** At leaf nodes of AND/OR junction search graph, it is trivial that $v(n) = UB(n) = LB(n)$ as $v(n)$ is computed using local constraints and does not involve any heuristics.
Induction step: At any AND node having OR children $m_1, \ldots, m_k$,

$$v(n) = label + \textstyle\sum_i v(m_i) \leq label + \sum_i UB(m_i),$$

where $v(m_i) \leq UB(m_i)$.

$$v(n) \leq \min(h_{UB}(n), label + \textstyle\sum_i UB(m_i)) = UB(n),$$

where $v(n) \leq h_{UB}(n)$.

At any OR node having AND child $m = \mathrm{argmax}_i\, v(m_i)$,

$$v(n) = v(m) \leq UB(m)$$

$$v(n) \leq min(h_{UB}(n), UB(m)) = UB(n).$$

$LB(n) \leq v(n)$ can be proved similarly. Therefore,

$$LB(n) \leq v(n) \leq UB(n). \qquad \square$$

Also, $UB(n)$ and $LB(n)$ provides tighter bounds than the initial heuristic functions.

Proposition 3 (Pruning rule) *For any AND node n and its sibling m, if $UB(n) < LB(m)$ or $UB(n) = LB(n)$ then subtree below n can be pruned.*

DJAO(k) We now set up a DJAO search on $S_T(P)$ whose nodes are assigned to agents. Starting from the root agent given the initial heuristic upper and lower bound functions h_{UB} and h_{LB}, the objective is to search one of the solution that satisfies the termination condition while pruning dominated candidate solutions.

DJAO agents use three types of messages: VALUE, COST, and TERMINATE. At the start, the root agent expands the OR nodes from its AND node and selects the best branch in the subtree and sends VALUE messages containing variable values on the chosen branch to its child nodes.

Upon receipt of a VALUE message, an agent evaluates whether the back-up condition is satisfied for the given value assignments $\mathbf{b}$ in the message. If the back-up condition is satisfied, the agent backs up with updated values by sending a COST message to its parent. Otherwise, it expands the OR nodes compatible with $\mathbf{b}$ and selects the best branch and sends VALUE message to its children.

Algorithm 1: DJAO(k)(1)

```
procedure Init()
$wait \leftarrow 0$ ; // number of waited messages
$k_i \leftarrow 0, k_c \leftarrow 0$ ; // own and child's k value
$m_b \leftarrow nil$ ; // OR node in $par(a_i)$ to backtrack to
$m^*, m^{**}$; // OR node with max, second max UB
$n_c$; // AND node context-compatible with $m_b$
procedure RootRun()
Init();
UpdateM();
if (CheckTermination()) then
    Send(TERMINATE) to $\forall c \in succ(a_i)$;
    terminate;
end
$k_i \leftarrow UB(m^*) - \max(UB(m^{**}) - k, LB(m^*))$;
$wait \leftarrow \|succ(a_i)\|$;
Send(VALUE, $m^*, k_i$);
loop forever
while (message queue is not empty) do
    pop msg off message queue;
    When Received(msg);
    if (CheckTermination()) then
        Send(TERMINATE) to $\forall c \in succ(a_i)$;
        terminate;
    else if ( wait==0) then
        UpdateM();
        $k_i = UB(m^*) - \max(UB(m^{**}) - k, LB(m^*))$;
        Send(VALUE, $m^*, k_i$) to $\forall c \in succ(a_i)$;
end
procedure Run()
Init();
loop forever
while (message queue is not empty) do
    pop msg off message queue;
    When Received(msg);
    if (Decide BackUp() && wait==0) then
        Send(COST, $m_b, UB(m_b), LB(m_b)$) to $par(a_i)$
    else if (wait==0) then
        UpdateM();
        Send(VALUE, $m^*, k_c$), to $\forall c \in succ(a_i)$
end
```

Algorithm 2: DJAO(k)(2)

```
procedure UpdateM()
$m^* = \text{argmax}\, UB(m)$, for $m \in succ(n_r)$;
$m^{**} = \text{argmax}\, UB(m)$, for $m \in succ(n_r) \setminus m^*$;
procedure When Received(COST, $m, v_{UB}, v_{LB}$)
$wait \leftarrow wait - 1, UB(m) \leftarrow v_{UB}, LB(m) \leftarrow v_{LB}$;
$UB'(n) \leftarrow UB(n)$, where $n = par(m)$ ;
$UB(n) \leftarrow max(UB(n), UB(m))$ ;
$LB(n) \leftarrow max(LB(n), LB(m))$;
procedure When Received(VALUE, $m, k$)
$k_i \leftarrow k, m_b \leftarrow m, wait \leftarrow \|succ(a_i)\|$;
$n_c \leftarrow \mathbf{context-compatible(m)}$,;
procedure Decide BackUp()
if $(UB(n_c) - UB'(n_c) \geq k_i)$ then
    return true;
else
    $k_c \leftarrow (k_i - (UB(n) - UB'(n)))/\|succ(a_i)\|$;
    return false;
end
procedure When Received(TERMINATE)
Send(TERMINATE) to $\forall c \in succ(a_i)$;
terminate;
procedure Check Termination()
if ( $UB(n_r) == LB(n_r)$) then
    return true;
else if for $\forall m \in succ(n_r) \backslash m^*, UB(m) \leq LB(m^*)$ then
    return true;
return false;
```

Upon receipt of COST message containing the updated lower and upper bounds on the chosen expanded OR nodes from all child nodes, it recalculates the lower and upper bounds of its AND node. It then re-evaluates the back-up condition for the received VALUE message. Unless it satisfies the back-up condition, then the question of which branch to select is re-examined and the agents sends another VALUE message to its children. These steps are repeated until a termination condition in Prop 4 holds for the root agent. It then sends a TERMINATE message to each of its children and terminate. Upon receipt of a TERMINATE message, each agent does the same.

Proposition 4 *Given an OR node n and AND nodes $m_1, \dots, m_k$ at the root agent, DJAO(k) is terminated if UB and LB satisfies the condition $UB(n) = LB(n)$ or $\exists i, UB(m_j) \leq LB(m_i)$ for $\forall j$, $i \neq j$.*

Each agent stores the lower and upper bounds of expanded nodes and updates these values upon each COST message arrival. The memory requirement for each agent does not exceed $O(n^d)$ where n is the size of variable domain and d is the induced width of the junction tree.

Among many different search strategies which determines the back-up condition for solving COP and DCOP, best-first search and depth-first branch-and-bound search have been primarily studied (Marinescu and Dechter 2005; 2007; Modi et al. 2005; Yeoh, Felner, and Koenig 2008). Best-first search always follows the best item found and in the distributed setting whenever there is an update, agents propagate it to all ancestors whose best items may change. On the other hand, depth-first search retains the current path until it is certainly dominated or the true value of node $v(n)$ is found. In (Gutierrez, Meseguer, and Yeoh 2011), $ADOPT(k)$ provides a trade-off between these two extremes, where the search keeps the current path until the distance between the best solution on the current path and the best solution found so far becomes greater than a given constant k.

Similarly, we developed DJAO(k), which subsumes both depth-first and best-first search strategy on AND/OR search junction graph. It performs depth-first when $k = \infty$, best-first when $k = \epsilon$, and a hybrid when $\epsilon < k < \infty$, where

k is the distance between the best found solution $UB(m^*)$ and the next best solution $UB(m^{**})$ found so far. Unlike ADOPT(k) in which each agent determines k using the best solutions based on its subproblems provided by the node's subtree, each agent in DJAO instead uses a measurement that considers a global perspective of the entire problem on the current best solution. The search backtracks when $UB(m^*) \leq \max(UB(m^{**}) - k, LB(m^*))$, which occurs as soon as the best solution is dominated by the second best with the best-first strategy with $k = \epsilon$, and when the true value for m^* is found (Thus, $UB(m^*) = LB(m^*)$) with the depth-first strategy with $k = \infty$.

Algorithm 1 and 2 shows the pseudocode of DJAO(k), where a_i is a generic agent, $par(a_i)$ its parent agent, $succ(a_i)$ its set of child agents, $par(n)$ the parent node of the node n in the search graph, $succ(n)$ the set of node n's child nodes, and n_r the AND node of the root agent. The root agent runs RootRun() which contains search initiation whereas all other agents runs Run(). The pseudocode uses a predicate $\textbf{context–compatible}(m)$ to select a node whose variable value assignment matches that of the node m.

Example of DJAO On the junction tree in Fig. 1d, let there be three agents, A_1, A_2 and A_3 for the cliques AB, ABC and BD respectively. The constraint function $f(A, B)$ are assigned to clique AB, $f(A, C)$ and $f(B, C)$ to clique ABC, $f(B, D)$ to BD.

$f(A, B)$:

A	B	
0	0	0
0	1	4
1	0	5
1	1	1

$f(B, D)$:

B	D	
0	0	0
0	1	1
1	0	4
1	1	2

$f(A, C)$:

A	C	
0	0	0
0	1	2
1	0	3
1	1	0

$f(B, C)$:

B	C	
0	0	1
0	1	4
1	0	5
1	1	2

Phase 1: The first phase starts by the agent A_2 computing the local potential b by merging $f(A, C)$ and $f(B, C)$ in the clique ABC as well as A_3.

$b(A, B, C)$:

A	B	C	
0	0	0	1
0	0	1	6
0	1	0	5
0	1	1	4
1	0	0	4
1	0	1	4
1	1	0	8
1	1	1	2

Each agent who does not own the root node generates a filtered message once they have received from all the child agents (agents who own the child OR nodes). The message from A_2 and A_3 to A_1 in Action-GDL would be as follows.

$M_{A_3 \to A_1}$:

B	
0	1
1	4

$M_{A_2 \to A_1}$:

A	B	
0	0	6
0	1	5
1	0	4
1	1	8

Filtered messages are created and sent with the filtering rate $l = 80$. FS denotes the filtered set of variable configurations. For the message $M_{A_2 \to A_1}$, the function range is $(8 - 4)$, thus items with upper bound equal or less than (4+4*0.8) are filtered except $(A{=}1, B{=}1)$. Messages in DJAO are:

$M_{A_3 \to A_1}$:

B	h_{LB}	h_{UB}
1	4	4
FS	1	1

$M_{A_2 \to A_1}$:

A	B	h_{LB}	h_{UB}
1	1	8	8
FS		4	6

Once messages received, A_1 calculates potential b as the total sum of received messages and local functions.

$b(AB)$:

A	B	LB	UB
0	0	5	7
0	1	12	14
1	0	10	12
1	1	13	13

Phase 2: A_1 checks the termination condition on the possible solution with the highest upper bound $(A{=}0, B{=}1)$. Since $LB(A{=}0, B{=}1)$ does not dominate $UB(A{=}1, B{=}1)$, the search starts. A_1 computes the distance k_1 between maximum and the second maximum upper bounds. The distance $k_1 = 14{-}13 = 1$. The search backtracks when the upper bound decreases by equal or more than $\min(k, k_1)$. The upper and lower bound gap originates only from $M_{A_2 \to A_1}$. Therefore, A_1 sends a VALUE message with a variables configuration $(A{=}0, B{=}1)$ and $\min(k, k_1)$ to A_2. A_2 receives this VALUE message. It then sends a COST message $LB(0, 1){=}5, UB(0, 1){=}5$ as it is a leaf node. Upon receipt of the COST message, the root node updates its potential.

$b(A, B)$:

A	B	LB	UB
0	0	6	7
0	1	12	12
1	0	11	12
1	1	13	13

Since the lower bound of $(A{=}1, B{=}1)$ dominates upper bounds of all other configurations, the termination condition is satisfied and the search terminates.

Approximate DJAO(k) An approximate version of the algorithm can be obtained by relaxing the constraint on upper and lower bound gap similar to search-based DCOP algorithms (Modi et al. 2005; Yeoh, Sun, and Koenig 2009). Approximate DJAO with an error bound e terminates when the solution contains no more than error e such that that value of the found solution n is no worse than $v(n^*) - e$, where n^* is the optimal solution. The corresponding termination condition is $LB(n) \geq UB(n) - e$ or $\exists i, UB(m_j) \leq LB(m_i) + e$ for $\forall j, i \neq j$. Also, the search backtracks when $UB(m^*) \leq \max(UB(m^{**}) - k, LB(m^*) + e)$.

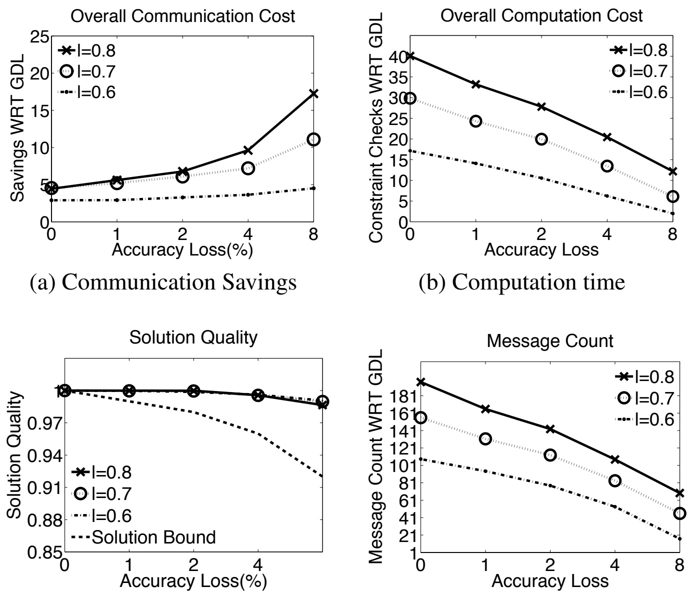
Empirical Evaluation

In this section we evaluate the performance of DJAO search. For each experiment, we report the communication costs, NCCCs(non-concurrent constraint check), and solution quality for approximate solutions with respect to optimal solutions. We used a DJAO that sends VALUE messages to at most 25 nodes when there are ties. We evaluate and compare our approach with Action-GDL and ADOPT(k) with k which was reasonable among 400, 4000 and 40000. Communication costs are measured as the number of bytes sent during execution[1] and the message count of both UTIL

[1]A single variable value is 4byte, and cost 8byte.

c	Algorithm	Total Bytes	NCCCs	Msgs
20	DJAO($k = \epsilon$)	**227744**	25627367	667
	DJAO($k = 10$)	260708	25991283	615
	DJAO($k = 100$)	229503	20898049	664
	DJAO($k = 500$)	271962	34144648	1485
	Action-GDL	394690	2741859	18
	ADOPT(K=4000)	1217757	4341532	206082
25	DJAO($k = \epsilon$)	1540523	794393531	2888
	DJAO($k = 10$)	1509601	835729595	2804
	DJAO($k = 100$)	**1508705**	570976502	2478
	DJAO($k = 500$)	2516606	1792901180	9263
	Action-GDL	3679104	25942425	18
	ADOPT(K=4000)	54556373	417172726	8992463
30	DJAO($k = \epsilon$)	**1513317**	734471121	3203
	DJAO($k = 10$)	2204580	1117448830	3505
	DJAO($k = 100$)	1513698	553233251	2910
	DJAO($k = 500$)	4084228	2759254975	17741
	Action-GDL	4568592	33038068	18
	ADOPT(K=4000)	37285466	219714985	6334245

Table 1: Performance of Optimal DJAO(k) on Random Binary DCOP Instances



(a) Communication Savings

(b) Computation time

(c) Solution Quality

(d) Message Count

Figure 3: Performance of Approximate DJAO(k=10)

	Algorithm	Total Bytes	NCCCs	Msgs
A	DJAO($k = \epsilon$)	163,360	22,217,301	315
	DJAO($k = 100,000,000$)	**155,021**	25,413,935	326
	Action-GDL	3,624,186	19,905,921	126
	ADOPT(K=30,000,000)	11,121,364	24,068,428	2,005,732
B	DJAO($DJAOk = \epsilon$)	278,168	30,441,957	325
	DJAO($k = 100,000,000$)	**238,696**	21,998,565	342
	Action-GDL	4,274,606	24553995	126
	ADOPT(K=30,000,000)	54,735,040	166,542,715	9,869,280
C	DJAO($k = \epsilon$)	207,801	9,379,233	194
	DJAO($k = 100,000,000$)	**120,516**	7,509,783	191
	Action-GDL	1,294,382	6,419,231	78
	ADOPT(K=30,000,000)	1,009,124	2,625,136	178,301
D	DJAO($k = \epsilon$)	718,528	31,980,359	405
	DJAO($k = 100,000,000$)	**482,654**	43,316,277	474
	Action-GDL	10,321,229	58,754,948	126
	ADOPT(K=30,000,000)	21,573,856	66,347,439	3,812,541

Table 2: Sensor Network Instances

and VALUE messages. For approximate results, we show the true utility of found solution which is often much higher than the estimated lower bound.

We experimented on random binary DCOP instances with 10 variables of domain size 10. The function cost are randomly generated over the range $\langle 0, \ldots, 200 \rangle$. We varied the number of constraint functions c over 20, 25 and 30 and averaged our results over 20 instances for each value of c. The total communication amount is largely determined by the structure of junction tree. Thus, five junction trees were constructed for each problem instance. Initial heuristics were generated using soft filtering where the bottom 70% is filtered. The DJAO with $k = \epsilon$(i.e., conducts best-first search) consistently performed well in these experiments.

Tab. 2 shows the results on sensor network instances from a public repository (Yin 2008) with a heuristic which filters 90% ($l = 0.9$). Tab. 1 and 2 show DJAO requires less communication than both ADOPT[2] and Action-GDL. Compared to the random DCOP instances, function ranges are wider and many clearly dominated variable configurations results in significant communication savings with DJAO. DJAO with high k values performed better in terms of communication on these lower connectivity graphs compared to the random DCOP instances.

Lastly, in an experiment on the random binary DCOP instances, we measured the methods' trend as the solution quality guarantee changes. We evaluated 20 instances for each quality loss ranging from $loss = 0\%$ to $loss = 8\%$ using a heuristic which filters 80%(l=0.8) and 70%(l=0.7). Results in Fig. 3 show that DJAO gains significant savings in communication as the error bound increases. With 80% heuristics, it transmits about 18 times less information than that of Action-GDL when $loss = 8\%$ while it uses 13 times more computation (NCCCs) and about 130 messages per agent.

Conclusions

We addressed the problem of solving DCOP exactly and with precise approximation bounds by developing a new distributed algorithm called DJAO. There are three novel ideas in DJAO. Firstly, it uses an AND/OR junction graph representation, which builds a basis for efficient search in the distributed settings. The second is a two phase search strategy that combines characteristics of ADOPT and Action-GDL. The third is a soft filtering technique to significantly reduce communication without losing any accuracy.

[2]Results from (Gutierrez, Meseguer, and Yeoh 2011)

References

Aji, S., and McEliece, R. 2000. The generalized distributive law. *Information Theory, IEEE Transactions on* 46(2):325–343.

Ali, S.; Koenig, S.; and Tambe, M. 2005. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, 1041–1048. New York, NY, USA: ACM.

Brito, I., and Meseguer, P. 2010. Improving DPOP with function filtering. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, 141–148. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Cowell, R. G.; Dawid, A. P.; Lauritzen, S. L.; and Spiegelhalter, D. J. 2007. *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks*. Springer Publishing Company, Incorporated, 1st edition.

Dechter, R., and Mateescu, R. 2007. AND/OR search spaces for graphical models. *Artif. Intell.* 171(2-3):73–106.

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, 639–646. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Gutierrez, P.; Meseguer, P.; and Yeoh, W. 2011. Generalizing ADOPT and BnB-ADOPT. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, 554–559. AAAI Press.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

Marinescu, R., and Dechter, R. 2005. AND/OR branch-and-bound for graphical models. In *Proceedings of the 19th international joint conference on Artificial intelligence*, IJCAI'05, 224–229. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Marinescu, R., and Dechter, R. 2007. Best-first AND/OR search for graphical models. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, 1171–1176. AAAI Press.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.* 161(1-2):149–180.

Paskin, M.; Guestrin, C.; and McFadden, J. 2005. A robust architecture for distributed inference in sensor networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05. Piscataway, NJ, USA: IEEE Press.

Pujol-Gonzalez, M.; Cerquides, J.; Meseguer, P.; and Rodriguez-Aguilar, J. A. 2011. Communication-constrained DCOPs: message approximation in GDL with function filtering. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, 379–386. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Vinyals, M.; Rodriguez-Aguilar, J. A.; and Cerquides, J. 2011. Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems* 22(3):439–464.

Wallace, R. J. 1996. Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*, AAAI'96, 188–195. AAAI Press.

Yeoh, W.; Felner, A.; and Koenig, S. 2008. BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, 591–598. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Yeoh, W.; Sun, X.; and Koenig, S. 2009. Trading off solution quality for faster computation in dcop search algorithms. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, 354–360. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Yin, Z. 2008. USC DCOP repository.