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 distributely 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 $(X_{i}, a)$ 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 $(X_{i}, a)$, consistent along the path from the root. The children of an AND node $(X_{i}, a)$ are OR nodes labeled with the children of variable $X_{i}$ in $T$. The path of a node $n \in S_{T}$, denoted $\text{Path}_{S_{T}}(n)$, 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 $l(X_{i}, (X_{i}, a))$ is defined as the sum of all the cost function values for which variable $X_{i}$ 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 $n = (X_{i}, a)$ is a terminal AND node then $v(n) = l(X_{i}, (X_{i}, a))$; (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 \( ST \).

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.

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.

Junction Tree and DCOP

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

- A set of variables \( X = \{ X_1, \ldots, X_r \} \), where each variable has a finite domain \( D \) (maximum size \( N \)) of possible values that it can be assigned.

- A set of constraint functions \( F = \{ F_1, \ldots, F_k \} \), where each constraint function, \( F_i : X_i \rightarrow \mathbb{R} \), takes as input any setting of the variables \( X_j \subseteq X \) and provides a real valued utility.

In DCOP, we assume that each variable \( x_i \) is owned by an agent \( a_i \in A \) 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 \( F_i \) that connects all variables that are involved in the function \( F_i \).

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

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 \( (X, C, S, F) \), where \( X \) is a set of variables, \( C \) is a set of cliques, where each clique \( C_i \) is a subset of variables \( C_i \subseteq X \), \( S \) is a set of separators, where each separator is an arc between two adjacent cliques containing their intersection; and \( F \) is a set of potentials, where each potential in \( F \) is assigned to each clique in \( C \).

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 \( C_i \in C \) can be solved independently given the local constraint functions \( f_i \in F \) and the values from neighbors on separator \( s_i \in S \). Separators \( S \) 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 \(|S_{NF}| + 2 \). 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 node 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](image)

**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 \( \langle c_i; (a, N_i), (S_{ij}, b) \rangle \) where \( a \) are variables assignments in the domains of variables in separators \( S_i \), whose value are propagated from ancestors and newly appeared variables \( N_i \) in clique \( C_i \). These OR nodes correspond to the cliques with partial assignment. The AND nodes are labeled \( \langle S_{ij}, b \rangle \) 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 \( \langle S_{ij}, b \rangle \) are OR nodes which are labeled with \( C_j; \langle S_{ij}, b, N_j \rangle \) with the same assignment on variables in separators \( S_{ij} \) and \( S_j \).

**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. 1c, the search is unguided until the third expansion. 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. 3.

**Definition 6** label: The label \( l(C_i; (S_i, a, N_i), (S_{ij}, b)) \) of the arc from the OR node to the AND node \( (S_{ij}, b) \) 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 its child AND node.

The value \( v(n) \) 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 $S_i$ and $S_{ij}$ 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 $C_i$ : $(S_i, a, N_i)$ 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 $C_3$ : $(B, 0, D), C_3$ : $(B, 1, D)$. 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 $C_i$ : $(S_i, a, N_i)$, $A_j$ transmits partial assignments 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 $UB(n)$, 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 $PSG$ 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) $n = (S_{ij}, b)$ is an AND node, having OR children $m_1, \ldots, m_k$, and $label = l(C_i : \langle S_i, a, N_i \rangle, \langle S_{ij}, b \rangle)$, then $UB(n) = \min(h_{UB}(n), label + \sum_{i=1}^{k} UB(m_i))$ $LB(n) = \max(h_{LB}(n), label + \sum_{i=1}^{k} LB(m_i))$; (iii) if $n = C_i : \langle P_i, a, N_i \rangle$ where $n$ is an OR node, having an AND child $m$, then $UB(n) = \min(h(n), UB(m))$ and $LB(n) = \max(h(n), LB(m))$.

**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 $UB(n) \leq h_{UB}(n)$.

**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 heuristic function.

**Induction step:** At any AND node having OR children $m_1, \ldots, m_k$, $v(n) = label + \sum_i v(m_i) \leq label + \sum_i UB(m_i)$, where $v(m_i) \leq UB(m_i)$. $v(n) \leq \min(h_{UB}(n), label + \sum_i UB(m_i)) = UB(n)$, where $v(n) \leq h_{UB}(n)$.

At any OR node having AND child $m = \arg\max_i v(m_i)$, $v(n) = v(m) \leq UB(m)$ $v(n) \leq \min(h_{UB}(n), UB(m)) = UB(n)$.

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

**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 $b$ 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 $b$ and selects the best branch and sends a VALUE message to that child node.
Upon receipt of a 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.
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, \( \text{par}(a_i) \) its parent agent, \( \text{suc}(a_i) \) its set of child agents, \( \text{par}(n) \) the parent node of the node \( n \) in the search graph, \( \text{suc}(n) \) the set of node \( n \)’s child nodes, and \( n_r \) the AND node of the root agent. The root agent runs \( \text{RootRun}() \) which contains search initiation whereas all other agents run \( \text{Run}() \). The pseudo-code uses a predicate context-compatible\( (m) \) to select a node whose variable 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 \( f(A, B) \) is assigned to clique \( AB \), \( f(A, C) \) and \( f(B, C) \) to clique \( ABC, f(B, D) \) to \( BD \).

\[
\begin{array}{c|c|c|c}
A & B & C & D \\
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
f(A, B) : & 0 & 1 & 4 \\
f(B, D) : & 0 & 1 & 1 \\
f(A, C) : & 0 & 1 & 2 \\
f(B, C) : & 0 & 1 & 2 \\
\end{array}
\]

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 \( A_3 \).

\[
\begin{array}{c|c|c|c}
A & B & C \\
0 & 0 & 0 \\
0 & 1 & 1 \\
0 & 1 & 6 \\
1 & 0 & 5 \\
1 & 1 & 8 \\
\end{array}
\]

\[
b(A, B, C) : 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \\
\]

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

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
M_{A_3 \rightarrow A_1} & B & 0 & 1 & 4 & A & 0 & 1 & 5 \\
M_{A_2 \rightarrow A_1} & 0 & 0 & 0 & 1 & 0 & 4 & 1 & 1 \\
\end{array}
\]

Filtered messages are created and sent with the filtering rate \( l = 80 \). FS denotes the filtered set of variable configurations. For the message \( M_{A_2 \rightarrow A_1} \), 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:

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
M_{A_3 \rightarrow A_1} & B & 0 & 1 & 4 & M_{A_2 \rightarrow A_1} & A & 0 & 1 & 5 \\
FS & 4 & 4 & 1 & 1 & 8 & 8 & 4 & 6 \\
\end{array}
\]

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

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
A & B & LB & UB \\
0 & 0 & 1 & 5 & 7 \\
0 & 1 & 12 & 14 \\
1 & 0 & 10 & 12 \\
1 & 1 & 13 & 13 \\
\end{array}
\]

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 \( M_{A_2 \rightarrow A_1} \). Therefore, \( A_1 \) sends a \( \text{VALUE} \) message with a variables configuration \((A=0,B=1)\) and \( \min(k,k_1) \) to \( A_2 \). \( A_2 \) receives this \( \text{VALUE} \) message. It then sends a \( \text{COST} \) message \((LB(0,1)=5, UB(0,1)=5) \) as it is a leaf node. Upon receipt of the \( \text{COST} \) message, the root node updates its potential.

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
A & B & LB & UB \\
0 & 0 & 6 & 7 \\
0 & 1 & 12 & 14 \\
1 & 0 & 11 & 12 \\
1 & 1 & 13 & 13 \\
\end{array}
\]

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 \( \exists i, UB(m_i) \leq LB(m_i) + e \) 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 \( \text{VALUE} \) messages to at most 25 nodes when there are ties. We evaluate and compare our approach with Action-GDL and ADOPT(k) with \( k = 4000 \) 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

\(^1\)A single variable value is 4bytes, and cost 8bytes.
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.

## Conclusions

3Results from (Gutierrez, Meseguer, and Yeoh 2011)
References


Yin, Z. 2008. USC DCOP repository.