Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling

Rajiv T. Maheswaran, Milind Tambe, Emma Bowring,
Jonathan P. Pearce, and Pradeep Varakantham
University of Southern California
{*maheswar, tambe, bowring, jppearce, varakant*}*@usc.edu*

Abstract

Distributed Constraint Optimization (DCOP) is an elegant formalism relevant to many areas in multiagent systems, yet complete algorithms have not been pursued for real world applications due to perceived complexity. To capably capture a rich class of complex problem domains, we introduce the Distributed Multi-Event Scheduling (DiMES) framework and design congruent DCOP formulations with binary constraints which are proven to yield the optimal solution. To approach real-world efficiency requirements, we obtain immense speedups by improving communication structure and precomputing best case bounds. Heuristics for generating better communication structures and calculating bound in a distributed manner are provided and tested on systematically developed domains for meeting scheduling and sensor networks, exemplifying the viability of complete algorithms.

1. Introduction

In many large-scale multiagent applications, including sensor nets, distributed spacecraft, disaster rescue simulations, and software personal assistants, agents must attempt to optimize their joint performance. For example, sensor agents must optimally schedule sensor resources to maximize targets tracked, and personal assistant agents must optimize their users' time while scheduling multiple meetings. Distributed constraint optimization (DCOP) [8] has emerged as a key formalism for such settings where distributed agents, each with control of some variables, try to optimize a global objective function, which is an aggregation of utility functions, each constrained by the values of a subset of variables. DCOP presents itself as a useful tool in domains such as meeting scheduling, where an organization wants to maximize the value of their employees' time while maintaining the privacy of information such as the relative importance of meetings or users' schedules. It is also appropriate in environments such as large-scale distributed sensor networks where there may not be a node with sufficient computing capability to centralize the task of finding optimal viewing decisions.

Despite its promise, two challenges must be addressed for DCOP to advance as a viable approach for real-world problems. First, while researchers have mapped specific problems to DCOP [7], a systematic reusable framework with congruent mappings to DCOP formulations has not been developed. Without automated mappings, the tedious process of modeling an environment, choosing variable sets, and designing constraint utility functions that yield the appropriate optimal solution would have to be repeated for each problem domain. Second, it is unclear if DCOPs obtained from concrete problems will fall within a space where complete algorithms for problems with NP complexity are fast enough to be utilized.

This paper takes some key steps in addressing the two challenges to DCOP raised above. We present the DiMES (Distributed Multi-Event Scheduling) framework which captures a rich class of real-world problems where multiple agents must generate a coordinated schedule for execution of joint activities or resource usage in service ofmultiple events. We then design three formulations that map DiMES into DCOP and prove the congruency and optimality of our formulations. When all constraints involve only two agents, we can model DCOP as a graph where the nodes represent variables and constraint utility functions are distributed as weights on edges between appropriate variables. To address the efficiency of complete algorithms, we present two key heuristics to improve convergence: (i) While organizing distributed constraint graphs as a tree is useful to eliminate the restrictive requirement of forcing a linear ordering over all agents [13, 12], the precise impact of tree structure in DCOP remained uninvestigated. We present

a new technique to provide shallower trees and experimentally illustrate the speedups that result. (ii) We also developed a new heuristic where a variable can *a priori* compute best case bounds for the subtree for which it is the root. These bounds, obtained in a *distributed* manner for all nodes in the tree, expedite the evolution of the search by allowing for better messaging to children and shifting a threshold at the root that determines termination. Our algorithmic improvements are demonstrated on ADOPT [8], experimentally demonstrated to be the most efficient *complete* DCOP algorithm in a range of settings. We show the effectiveness of our formulations in both meeting scheduling and sensor network settings where our two *convergence catalysts* combine to provide orders of magnitude improvement in performance.

2. Distributed Multi-Event Scheduling

We present a framework called DiMES (Distributed Multi-Event Scheduling) for capturing fundamental characteristics of problems occurring in real-world domains involving *joint activities*. We begin with a resource set $\mathcal{R} := \{R_1, \ldots, R_N\}$ of cardinality N where R_n refers to the n-th resource and an event set $\mathcal{E} := \{E^1, \ldots, E^K\}$ of cardinality K where E^k refers to the k-th event. Let us consider the minimal expression for the time interval $[T_{earliest}, T_{latest}]$ over which all events are to be scheduled. Let $T \in \mathbb{N}$ be a natural number and Δ be a length such that $T \cdot \Delta = T_{latest} - T_{earliest}$. We can then characterize the time domain by the set $\mathcal{T} := \{1, \ldots, T\}$ of cardinality T where the element $t \in \mathcal{T}$ refers to the time interval $[T_{earliest} + (t-1)\Delta, T_{earliest} + t\Delta]$. Thus, a business day from 8:00 AM to 6:00 PM partitioned into half hour time slots would be represented by $\mathcal{T} = \{1, \ldots, 20\}$ where time slot eight would represent the interval [11:30 AM, 12:00 PM]. Here, we implicitly assume equal-length time slots, though this can be relaxed easily.

Let us characterize the k-th event with the tuple $E^k := (A^k, L^k; V^k)$ where $A^k \subset \mathcal{R}$ is the subset of resources that are required by the event. The length of the event, $L^k \in \mathcal{T}$, is the number of contiguous time slots for which the resources A^k are needed. The heterogeneous importance of an event to the resources it requires is described in a value vector V^k whose length is the cardinality of A^k. If $R_n \in A^k$, then V^k_n will be an element of V^k which denotes the value per time slot to the n-th resource for scheduling event k. Each resource also has a value for each time slot which characterizes its preference for keeping that resource unassigned during that time slot. Let $V^0_n(t) : \mathcal{T} \to \mathbb{R}^+$ denote the n-th resource's valuation for keeping time slot t free. The relative values of various time slots for a particular resource reflects an ordering of slots to be used for assignments of events in $\mathcal{E}$. These valuations allow agents to compare the relative importance of events to other events and also to compare the importance of the event to the value of the resource's time. We implicitly assume that a resource cannot schedule two events simultaneously and the value of scheduling an event is independent of the time the event is assigned. Though extensions to time varying rewards are straightforward, our current framework reflects the idea that the importance of events tend to be stationary and temporal preferences generally emerge due to factors in the resource's schedule. Let $\overline{V}_n := \max_{k \in \{1,\ldots,K\}} V^k_n$ be the maximum value to the n-th resource for scheduling any event. A resource can eliminate a time slot from being considered by setting the value for keeping the time slot unassigned sufficiently high, i.e. $V^0_n(t) > \overline{V}_n$.

Given the framework discussed above, we now present the scheduling problem we are considering. Let us define a schedule S as a mapping from the event set to the time domain where $S(E^k) \subset \mathcal{T}$ denotes the time slots committed for event k. We assume that the event is not disjoint, i.e., event E^k must be scheduled in L^k contiguous slots. This implies that all resources in A^k must agree to assign the time slots $S(E^k)$ to event E^k in order for the event to be considered *scheduled*, consequently allowing the resources to obtain the utility for completing it. A scheduling conflict occurs if two events with at least one common resource are scheduled in a manner such that assigned time slots overlap: $S(E^{k_1}) \cap S(E^{k_2}) \neq \emptyset$, for any $k_1, k_2 \in \{1, \ldots, K\}$, $k_1 \neq k_2$, $A^{k_1} \cap A^{k_2} \neq \emptyset$. An assignment of $S(E^k) = \emptyset$ implies that event E^k is not scheduled. This can occur either because the required resources cannot agree on a common time due to scheduling conflicts with higher reward events or that the rewards for the event is too low with respect to the value of their unassigned time. To completely specify DiMES, we need a metric to choose among the possible schedules that have no conflicts.

Let us define the utility of a resource as the difference between the sum of the values from scheduled events and the aggregate values of the time slots if they were kept free. This measures the net gain between the opportunity benefit and opportunity cost of scheduling various events. The organization wants to maximize the sum of utilities of all its resources as it represents the best use of all assets within the team. Incorporating this naturally emerging global metric, we characterize the fundamental problem in this general frame-

work as: $\max_S \left\{\sum_{k=1}^{K} \sum_{n\in A^k} \sum_{t\in S(E^k)} \left(V_n^k - V_n^0(t)\right)\right\}$ such that $S(E^{k_1}) \cap S(E^{k_2}) = \emptyset \;\; \forall k_1, k_2 \in \{1,\ldots,K\},\; k_1 \neq k_2,\; A^{k_1} \cap A^{k_2} \neq \emptyset$. The DiMES problem is *NP*-Hard as can be seen by mapping it from graph *K*-coloring.

3. DCOP Formulations for DiMES

Given a problem from a real domain captured by the DiMES framework, we need an approach to obtain the optimal solution. As we are optimizing a global objective with local restrictions (eliminating conflicts in resource assignment), DCOP [8] presents itself as a useful and appropriate approach. A DCOP consists of a variables set $X = \{x_1, \ldots x_N\}$ distributed among agents where the variable x_i takes a value from the finite discrete domain D_i. The goal is to choose values for variables to optimize a global objective function, which is an aggregation of utility functions, each of which depend on the values of a particular subset of variables in X. If all the utility functions depend on exactly two variables, it can be modeled with a graph, where nodes represent variables and every utility function can be captured as an edge whose weight is determined by the values chosen by the nodes determining the edge. For each edge $(i, j) \in E$, (where E denotes a set of edges whose endpoints belong to a set homeomorphic to X), we have a function $f_{ij}(x_i, x_j) : D_i \times D_j \rightarrow \mathbb{R}$. Our goal is to choose an assignment $a^* \in A := D_1 \times \cdots \times D_N$, such that $a^* = \arg\max_{a\in A} \sum_{(i,j)\in E} f_{ij}\left(x_i = a_i, x_j = a_j\right)$.

Our challenge is to convert a given DiMES problem into a DCOP with binary constraints. We may then apply existing (or improved) algorithms developed for DCOP to obtain a solution. A DiMES problem is modeled by events and valuations. A DCOP is composed of a variable set and constraint utility functions. We developed three DCOP formulations based on three unique concepts for creating variable sets: time slots as variables (TSAV), events as variables (EAV), and private events as variables (PEAV). For each variable set, we constructed constraint utility functions such that the optimal solution of the resulting DCOP can be proven to be identical to the optimal solution to the underlying DiMES problem. Thus, given a quantification of events and valuations for a problem rooted in the real world, there exist at least three methods to directly obtain an optimal schedule. Due to space limitations, we provide only the characterization of variable sets for the TSAV and EAV formulations. For PEAV, we provide a description of the variable set, the explicit form of the constraint utility functions, and a proof of congruency.

TSAV (Time Slots as Variables): This method reflects a natural first step when considering scheduling issues. Let us define a DCOP where a variable $x_n(t)$ represents the *n*-th resource's *t*-th time slot. Thus, we have $N \cdot T$ variables. Each variable can a take on a value of the index of an event for which it is a required resource, or the value "0" to indicate that no event will be assigned for that particular time slot: $x_n(t) \in \{0\} \cup \{k \in \{1,\ldots,K\} : R_n \in A^k\}$. It is natural to distribute the variables in a manner such that $\{x_n(1), \ldots, x_n(T)\}$ belong to an agent representing the schedule of the *n*-th resource.

EAV (Events as Variables): We note that the graph structure of TSAV grows as the time range considered increases or the size of the time quantization interval decreases, leading to a denser graph. An alternate approach is to consider the events as the decision variables. Let us define a DCOP where the variable x^k represents the starting time for event E^k. Each of the K variables can take on a value from the time slot range that is sufficiently early to allow for the required length of the event or "0" which indicates that an event is not scheduled: $x^k \in \{0, 1, \ldots, T - L^k + 1\}, \quad k = 1, \ldots, K$. If a variable x^k takes on a value $t \neq 0$, then it is assumed that for all required resources of that event ($\forall n \in A^k$), the time slots $\{t, \ldots, t + L^k - 1\}$ must be assigned to the event E^k. It would be logical to assign each variable/event to the agent of one of the required resources for the event.

PEAV (Private Events as Variables): We note that in EAV, if an agent is to make a decision for an event as a variable, it must be endowed with both the authority to make assignments for multiple resources as well as have valuation information for all required resources. There are settings where resources, though part of a team, are unwilling or unable to cede this authority or information. To address this, we consider a modification of EAV that protects these interests. Let us define a set of variables $X^k := \{x_n^k : n \in A^k\}$ where $x_n^k \in \{0, 1, \ldots, T - L^k + 1\}$ denotes the starting time for event E^k in the schedule of R_n which is a required resource for the event. If $x_n^k = 0$, then R_n is choosing not to schedule E^k. We then construct a DCOP with the variable set $X := \cup_{k=1}^{K} X^k$. Let us now define a set $\tilde{X}_n := \{x_m^k \in X : m = n\} \subset X$ which is the collection of variables pertaining to the *n*-th resource. Clearly, $|\tilde{X}_n| > 0 \;\forall n$, otherwise the resource is not required in any event. If $|\tilde{X}_n| = 1$, let $X_n := \tilde{X}_n \cup \{x_n^0\}$ where $x_i^0 \equiv 0$ is a dummy variable. Otherwise, $X_n := \tilde{X}_n$. The DCOP partitions the variables in X_n to an agent representing the *n*-th resource's interests. Let all the variables within X_n (intra-agent links) be fully connected. The addition of the dummy variable to sets X_n with cardinality one is to ensure that intra-agent links exist for all agents. This allows us to design constraint utility functions where all valuation information is on internal links,

thus maintaining privacy. Inter-agent links exist between the variables for all participants of a given event, i.e., all the variables in X^k are fully connected.

Given a particular variable set, our challenge is to construct constraint utility functions such that when the resulting DCOP is solved, we obtain a solution which is congruent to the original DiMES problem. We have created such functions and proved their equivalence for all formulations. The resulting DCOP constraint graphs from TSAV, EAV, and PEAV for a scenario with resources $\{A, B, C, D, E, F\}$ in a four-time-slot window, where five events $\{E^1, \cdots, E^5\}$ of duration one time slot require the resources $\{AB, ACD, ADE, BC, EF\}$, respectively are shown in Figure 1. Due to space limitations, we only present the solution for PEAV.

Proposition 1 *The DCOP formulation with private events as variables, where the constraint between the variables $x_{n_1}^{k_1}$ and $x_{n_2}^{k_2}$ when $x_{n_1}^{k_1} = t_1$ and $x_{n_2}^{k_2} = t_2$ takes on the utility*

$$f(n_1, k_1, t_1; n_2, k_2, t_2) = -MI_{\{n_1 \neq n_2\}} I_{\{k_1 = k_2\}} I_{\{t_1 \neq t_2\}} + I_{\{n_1 = n_2\}} I_{\{k_1 \neq k_2\}} f_{intra}(n_1; k_1, t_1; k_2, t_2). \quad (1)$$

where $f_{intra}(n; k_1, t_1; k_2, t_2) =$

$$\begin{cases} -M & t_1 \neq 0, t_2 \neq 0, t_1 \leq t_2 \leq t_1 + L^{k_1} - 1, \\ -M & t_1 \neq 0, t_2 \neq 0, t_2 \leq t_1 \leq t_2 + L^{k_2} - 1, \\ g(n; k_1, t_1; k_2, t_2) & \textit{otherwise} \end{cases}$$

and

$$g(n; k_1, t_1; k_2, t_2) = \frac{1}{|X_n| - 1} \left(Z_n^{k_1}(t_1) + Z_n^{k_2}(t_2) \right)$$

$$\textit{where } Z_n^{k_i}(t_i) = \sum_{l=1}^{L^{k_i}} \left(V_n^{k_i} - V_n^0(t_i + l - 1) \right) I_{\{t_i \neq 0\}}$$

with $M > NTV_{\max}$ where N is the number of participants, T is the number of time slots and $V_{\max} = \max_{k,n} V_n^k$, yields the optimal solution to the Distributed Multi-Event Scheduling (DiMES) problem.

Proof. The first term in (1) characterizes that a penalty of $-M$ is assessed on an inter-agent link ($n_1 \neq n_2$) for a common event ($k_1 = k_2$) for which the same starting time is not selected ($t_1 \neq t_2$) by the connected resources. The latter term in (1) addresses intra-agent constraints ($n_1 = n_2$) between different events ($k_1 \neq k_2$) where the link utility $f_{intra}(\cdot)$ ensures that a penalty is incurred on an intra-agent constraint if a scheduling conflict is created. Otherwise, the utility gain for a resource assigning a viable time for an event is uniformly distributed among the outgoing intra-agent links as denoted in $g(\cdot)$.

Let us assume that a penalty is incurred on an inter-agent constraint. This implies that the required resources for a particular event could not agree on a common time to start. Since the total utility gain (excluding penalties) for holding an event E^k cannot exceed $\sum_{n \in A^k} \sum_{t=1}^{L^k} V_{\max} \leq NTV_{\max} < M$, there exists a solution for the DCOP where the event is not scheduled which is at least as good as that with the event scheduled. Let us now assume that a penalty is incurred on an intra-agent link. This implies that an agent has chosen starting times for two events that causes the same time slot to be assigned to two events. By similar logic, the penalty M is sufficiently large such that by choosing not to schedule one of the events and allowing all other agents to choose not to schedule that event (thereby avoiding incurring an inter-agent penalty), we obtain a higher quality solution. The above analysis implies that the optimal DCOP solution is void of assignments that would activate a penalty. Thus, $x_n^k = x_m^k, \ \forall m, n \in A^k$. Given E^k and some $n \in A^k$, let us define $S(E^k) = \emptyset$ if $x_n^k = 0$, and $S(E^k) = \{x_n^k, \ldots, x_n^k + L^k - 1\}$ if $x_n^k \neq 0$. Then, we have

$$S(E^{k_1}) \cap S(E^{k_2}) = \emptyset \ \ \forall k_1, k_2 \in \{1, \ldots, K\}, \quad k_1 \neq k_2, A^{k_1} \cap A^{k_2} \neq \emptyset. \quad (2)$$

Otherwise, a penalty would have been assessed. The global utility is then the sum of all intra-agent links devoid of penalties ($\sum g(\cdot)$), which can be represented as

$$\begin{aligned} &\sum_{n=1}^{N} \sum_{k=1}^{K} I_{\{n \in A^k\}} \left(\frac{1}{|X_n| - 1} Z_n^k(x_n^k) \left[\left(\sum_{k=1}^{K} I_{\{n \in A^k\}} \right) - 1 \right] \right) \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} I_{\{n \in A^k\}} Z_n^k(x_n^k) \\ &= \sum_{k=1}^{K} \sum_{n \in A^k} \sum_{l=1}^{L^k} \left(V_n^k - V_n^0(x_n^k + l - 1) \right) I_{\{x_n^k \neq 0\}} \\ &= \sum_{k=1}^{K} \sum_{n \in A^k} \sum_{t \in S(E^k)} \left(V_n^k - V_n^0(t) \right). \end{aligned}$$

The solution to the DCOP maximizes the previous expression, which when coupled with the no conflict condition in (2) is identical to the DiMES problem. ■

The constraint utility functions and proofs of congruency for TSAV and EAV follow similar reasoning and can be found at *http://pollux.usc.edu/~maheswar/aamas04proofs.pdf*.
We note that in practice instead of explicitly calculating $V_{\max}$ which may not be knowable due to privacy, we would use an upper bound on V_n^k given by the system. Given events and values, we are able to construct graphs and assign constraint link utilities from which we can apply a DCOP algorithm and directly obtain an optimal solution to the DiMES problem.

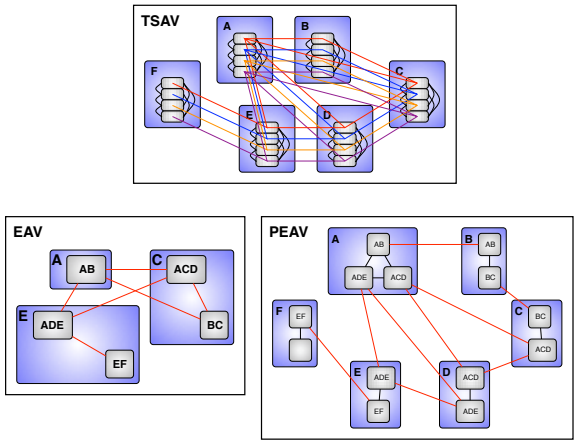

Figure 1. DCOP Constraint Graphs

4. Convergence Catalysts

To test the efficiency of our formulations, we used ADOPT [8] as a base as it has been shown to be the best available complete DCOP algorithm. Initial results when applying ADOPT "out of the box" to the EAV, PEAV, and TSAV encodings illustrated the critical need to address the speed of convergence. To ameliorate these complexity issues, we developed two key heuristics which produced significant speedups.

Communication Structure: The first heuristic involved the communication aspect of the DCOP algorithm. The ADOPT algorithm converts the constraint graph into a DFS tree which is used as a hierarchy to communicate value and cost messages. Though this broke away from the commonly used chain communication structure with linear ordering [12, 13], the best method to take advantage of the parallelism introduced by trees remained an open problem . The current method used to construct the communication tree is an extension of a heuristic used in linear ordering where the most constrained node (MCN) is used as the metric to choose the root from a subgraph.

Initial experiments have shown that the depth of the tree has a great effect on the rate of convergence to the optimal solution, and we hypothesize that the depth of the tree is a key factor to be minimized. The rationale for this can be seen when analyzing an exhaustive search for which an additional level of depth increases the number of solutions to be tested by a factor of the number of values available to the added variable. The MCN method does not yield the minimum depth tree in many cases. Since finding a minimum depth DFS tree is an NP-Complete problem [2] and the benefits of tree depth are unknown *a priori*, we propose a practical polynomial-time heuristic to find shallower DFS trees. This new heuristic is based on a method to find a minimum-depth spanning tree, where the node that is closest to the midpoint of the longest shortest path is used as a the root from any given subgraph, hereby denoted as the MLSP tree. The MLSP tree algorithm attempts to create the greatest branching while guaranteeing that there are no links between subtrees when all the links of the original graph are mapped onto the tree. We propose that MLSP is a superior metric for root selection as it attempts to address tree depth while MCN does not. The key algorithm in the recursive process to generate the MLSP tree is outlined in Algorithm 1. The MLSP tree generation heuristic is a polynomial-time algorithm as `MLSPTree` is called at most once per node and each process within it takes polynomial time. We note that we utilized centralized algorithms to generate both trees as we were interested in investigating the effect of the communication structure on performance. Designing an algorithm that efficiently implements MLSP in a distributed manner is still an open problem, but as polynomial-time algorithms for distributed all-pairs shortest path identification exist [4], we believe this is achievable.

Algorithm 1 `MLSPTree` (*Parent*, *Graph*, *Tree*)

1: *MidNode* = midpoint of longest shortest path in *Graph*
2: *PossibleChildren* = nodes in *Graph* connected to *Parent*
3: *ClosestNode* = node in *PossibleChildren* that is closest to *MidNode*
4: *ClosestNode* is set to be child of *Parent* in *Tree*
5: *RemaningGraphs* = Collection of connected graphs when *ClosestNode* and its links are removed from *Graph*
6: **for all** *SubGraph* ∈ *RemainingGraphs* **do**
7: *Tree* = `MLSPTree` (*ClosestNode*, *SubGraph*, *Tree*)
8: **end for**
9: Return *Tree*

Best Case Bounds ADOPT and other DCOP algorithms [8, 13] maintain best/worst-case bounds on solutions at each node in order to limit their search and determine termination at the root node (e.g., the best-case bound is an upper bound when maximizing utilities, or a lower bound when minimizing costs). The initial tightness of these bounds greatly affects convergence when applying ADOPT to real-world scenarios. This phenomenon does not reveal itself in domains such as graph coloring where the initial bounds are serendipitously as tight as possible due to the structure of the problem. Our key idea to expedite the DCOP search was to devise a method to endow each node with *a priori* information regarding best-case bounds that are automatically pre-computed in a *distributed* manner. This induces speedup due to the fact that the time spent during the evolution of the search discovering a pre-computable level for bounds is eliminated. In effect, a limited amount of preprocessing (one message per node sent up the DFS tree) can significantly

cut the actual DCOP computation at run time.

To this end, we introduce the *passup* heuristic to determine best-case bounds. If x is a node in a tree T, let A_x be the set of all ancestors of x, D_x be the set of all descendants of x, and $C_x \subseteq D_x$ be the set of all children of x (where a child is a descendant who is one level below x). Let f_{xy} denote the constraint utility function between nodes x and y. The descendant node is responsible for the utility of a link and thus, the total utility from a subtree with root x is

$$\begin{aligned} U_x &:= \sum_{y \in A_x} f_{xy} + \sum_{z \in D_x} \sum_{y \in A_z} f_{zy} \\ &= \sum_{y \in A_x} f_{xy} + \sum_{z \in C_x} \sum_{y \in A_z} f_{zy} + \sum_{z \in C_x} \sum_{q \in D_z} \sum_{y \in A_q} f_{qy} \\ &= \sum_{y \in A_x} f_{xy} + \sum_{z \in C_x} U_z \end{aligned}$$

Defining $T_x = \max U_x$, we have $T_x \leq \max \sum_{y \in A_x} f_{xy} + \sum_{z \in C_x} T_z$, by the concavity of the max function. Thus, best-case bounds (T_x) can be obtained throughout the tree if each node calculates a bound on its own contribution ($\max \sum_{y \in A_x} f_{xy}$) adds it to bound messages from its children ($\sum_{z \in C_x} T_z$) and passes it up to its parent. In our formulations, $f_{xy} = \alpha_x U_x^{node} + \alpha_y U_y^{node}$, where $0 < \alpha_x, \alpha_y \leq 1$, and the maximum node contribution to the utility, U_x^{node} can be bounded as follows:

$$\begin{aligned} \textbf{TSAV} &: U_x^{node} \leq \max_{k : n \in A^k} \left(V_n^k - V_n^0(t) \right) \\ \textbf{EAV} &: U_x^{node} \leq \max_{t \in \mathcal{T}} \sum_{n \in A^k} \left(V_n^k - V_n^0(t) \right) \\ \textbf{PEAV} &: U_x^{node} \leq \max_{t \in \mathcal{T}} \left(V_n^k - V_n^0(t) \right). \end{aligned}$$

Thus, by performing local maximizations, local aggregations and *passups*, best-case bounds can be obtained *a priori* for all nodes in the tree. This helps convergence in two ways: (i) the more obvious advantage is that with better best-case bounds, we effectively begin with a smaller search space as we eliminate all assignments with solution quality between the old and new bounds. (ii) the more subtle advantage is that when ADOPT calls for each parent to partition its best-case bound among its children during the dynamics of the search, the children are able to respond more quickly to bad partitioning assignments due to better knowledge of their best-case bounds which results in more intelligent partitioning. These effects combine to produce dramatic speedups as shown in Section 5.

5. Experimental Results

Initial experiments were conducted on random graphs with a handful of variables. Though they verified that our heuristics yield great speedups, we endeavored to test our ideas on two complex real-world domains encoded in DiMES. Our work on the CALO project [1] for developing a state-of-the-art personal assistant agent and earlier work on sensor networks [7] gave us a solid foundation upon which to create concrete scenarios. To that end, we systematically developed formal models to generate test cases in both domains.

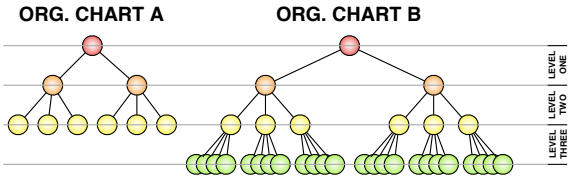

Figure 2. Organizational Hierarchies

SCENARIO	ORG CHART	GRP	LEVEL 1	LEVEL 2	LEVEL 3	MEETINGS
1	A	3 GRP	1 PTC	4 PTC	N/A	8
2	A	3 GRP	1 SIB	6 PTC	N/A	10
3	B	9 GRP	1 PTC	2 PTC	4 PTC	12
4	B	9 GRP	1 SIB	2 SIB	4 SIB	12

Table 1. Meeting Scheduling Scenarios

In the CALO team setting, we considered a multiple meeting scheduling problem. CALO's domain consists of office settings with organizational hierarchies such as the ones shown in Figure 2, where three types of meetings need to be scheduled: *group* (GRP) meetings consisting of a node in the org. chart and all its children, *parent to child* (PTC) meetings, and *sibling* (SIB) meetings. We investigated archetypical scenarios described in Table 1. With all meetings taking one time slot, we considered an eight-time-slot schedule, and randomly generated valuations for each scenario. Our metric for performance was the number of cycles [8], where one cycle is defined as all agents receiving incoming messages, executing local processing, and sending outgoing messages. The cost of preprocessing *passup* is equal to the depth of the tree in cycles as listed in Table 2. The average number of cycles after preprocessing to termination for twenty five EAV tests per scenario and three PEAV tests per scenario with various heuristics applied are shown in Figures 3 and 4, respectively. We note that tests with *passup* terminated in less than 1000 cycles for EAV creating miniscule bars in Figure 3.

A second domain that we considered was that of sensor networks, for scenarios where we are given corridors composed of squares which indicate areas that need to be observed. Sensors are located at each vertex of a square. Sensors at all vertices of a particular

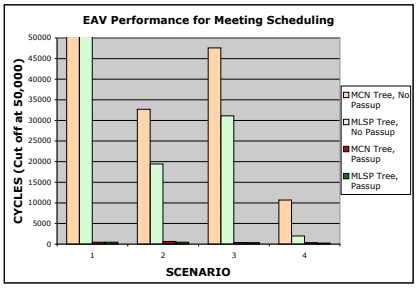

Figure 3. EAV for Meeting Scheduling

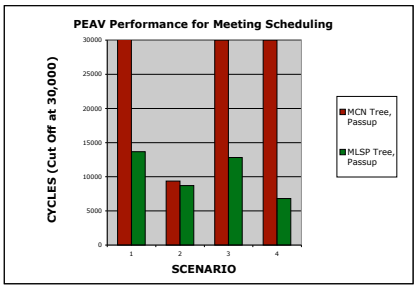

Figure 4. PEAV for Meeting Scheduling

square must be focused on that square for it to be considered observed. Given a set of events (squares to be observed) over some time horizon, the sensors (which can observe at most one square) must choose which square they are to observe in order to coordinate observation of events with the highest rewards. The layouts considered are shown in Figure 5. In each scenario, the event set was an observation of every square in the layout for one time slot. Given an eight-time-slot calendar and randomly generated valuations for twenty five runs of each scenario, the convergence data under EAV is shown in Figure 6. Table 2 shows the graph complexity and tree-depth differences for scenarios with meeting scheduling under EAV (MSE), meeting scheduling under PEAV (MSP) and sensor networks under EAV (SNE). We note that under EAV and PEAV, each variable can choose among eight values (time slots).

	MSE-1	MSE-2	MSE-3	MSE-4	MSP-1	MSP-2	MSP-3	MSP-4	SNE-1	SNE-2	SNE-3	SNE-4
VARIABLES	8	10	12	12	21	25	47	47	16	16	10	16
CONSTRAINTS	16	17	17	17	43	47	122	123	16	17	11	19
MCN TREE DEPTH	5	7	5	6	14	18	22	22	11	10	5	6
MLSP TREE DEPTH	5	6	4	5	10	13	12	14	8	8	5	5

Table 2. Graph Complexity and Tree Depths

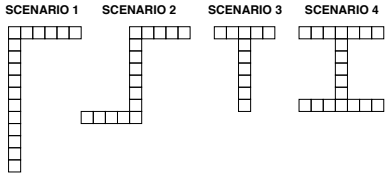

Figure 5. Scenarios for Sensor Nets Domain

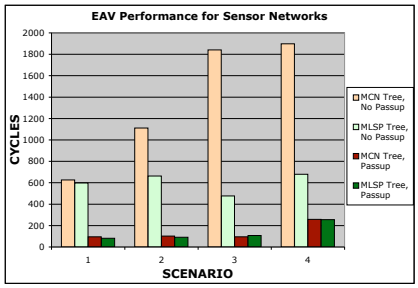

Figure 6. EAV for Sensor Nets, Linear Scale

We encountered two surprises by taking DCOP to real-world settings: (i) Our expectation that ADOPT "out of the box" would solve EAV problems within one hundred cycles as it had done for graph coloring problems with similar numbers of variables and constraints was shattered when convergence times were on the order of tens of thousands. This illustrated the existence of fundamental differences between abstract (graph coloring) and concrete (meeting scheduling, sensor network) problems. (ii) Our heuristics induced dramatic speedups bringing intractable problems (MSP) into a tractable space and tractable problems (MSE, SNE) into an expeditious space of hundreds of cycles. This enabled us to solve the largest known experiment (47 variables, 123 constraints) for complete DCOP.

In comparing formulations, we note that EAV outperformed PEAV by approximately one order of magnitude when using *passup* in meeting scheduling scenarios. When choosing formulations, a system designer would weigh the qualitative benefits of PEAV (greater distribution in variable control, less sharing of valuation information) against its convergence cost w.r.t. EAV. The inefficiency of the TSAV formulation, which never terminated for the simplest scenario under both heuristics, is illustrated by an example where two agents attempt to schedule two meetings in an eight-time-slot calendar. For three runs with randomized valuations, EAV (14 cycles) outperformed PEAV (97 cycles) which outperformed TSAV (8450 cycles). This dramatic scale-up in

cycles for TSAV serves to illustrate that choosing a formulation has a great impact on convergence and furthermore, creating an efficient congruent encoding is not a trivial problem.

In verifying our heuristics, we note depth differences between MCN and MLSP trees in all but two scenarios where depths were equal. This verifies the effectiveness of our heuristic for that purpose. Furthermore, we see that shallower depths generally lead to faster convergence. However, the fact that this is not a dominant characteristic justifies a polynomial-time preprocessing cost for likely speedup as opposed to investing in a non-polynomial-time algorithm to find the absolute shallowest tree. Applying the *passup* heuristic led to dramatic improvements with both trees. No PEAV test terminated within 72000 cycles without *passup*. In combination, our heuristics led to speedups of one to two orders of magnitude. Full details of the experimental results can be found at *http://teamcore.usc.edu/dcop*.

6. Summary and Related Work

This paper addresses DCOP for real-world problems, specifically two concrete settings: scheduling for teams of personal software assistant agents [1, 10] and scheduling teams of sensor agents [7, 5]. Our key contributions were (i) designing three formulations that automatically map the DiMES framework into DCOP that are proven to be optimal, (ii) introducing two novel heuristics to speedup DCOP algorithms, based on a new tree ordering technique and a distributed precomputation of best-case bounds, and (iii) experimental investigation of the impact of our techniques on systematically developed real-world domains illustrating speedups of one to two orders of magnitude. The main conclusion is that complete algorithms can indeed be viable options for real-world problems.

Complete algorithms outside ADOPT include SynchBB [13] and SynchID [8]. Several incomplete algorithms have been developed which sacrifice optimality for efficiency [14]. These algorithms could also be applied to the formulations presented in this paper. Frameworks have been developed for job shop scheduling [6, 9] which incorporate the idea of precedence constraints. Frameworks for meeting scheduling have been developed and studied where negotiation [11] and satisfaction [3] were the primary metrics.

Acknowledgments. This research is supported by a subcontract from SRI International.

References

[1] *CALO: Cognitive Agent that Learns and Organizes*, 2003. http://www.ai.sri.com/project/CALO, http://calo.sri.com.

[2] R. Dechter. *Encyclopedia of Artificial Intelligence*, chapter Constraint Networks, pages 276–285. John Wiley and Sons, second edition, 1992.

[3] M. S. Franzin, E. C. Freuder, F. Rossi, and R. Wallace. Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. In *Proceedings of the AAAI Workshop on Preference in AI and CP*, Edmonton, Canada, July 2002.

[4] S. Haldar. An 'all pairs shortest paths' distributed algorithm using $2n^2$ messages. In *Proc. 19th Int. Workshop on Graph-Theoretic Concepts in Comp. Sci.*, pages 350–363, 1993.

[5] V. Lesser, C. Ortiz, and M. Tambe. *Distributed Sensor Nets: A Multiagent Perspective*. Kluwer, 2003.

[6] J. Liu and K. P. Sycara. Multiagent coordination in tightly coupled task scheduling. In *Proceedings of the First International Conference on Multiagent Systems*, pages 181–188, December 1996.

[7] P. J. Modi, H. Jung, M. Tambe, W. Shen, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Proceedings of the Seventh International Conference on Principles and Practices of Constraint Programming*, Paphos, Cyprus, 2001.

[8] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems*, Sydney, Australia 2003.

[9] N. Sadeh and M. S. Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86:1–41, 1996.

[10] P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real-world. *Journal of Artificial Intelligence Research*, 17:171–228, 2002.

[11] S. Sen and E. Durfee. A formal study of distributed meeting scheduling: Preliminary results. In *Proceedings of the Conference on Organizational Computing Systems*, pages 55–68, November 1991.

[12] M. C. Silaghi, D. Sam-Haroud, and B. Faltings. Abt with asynchronous reordering. In *Second Asia-Pacific Conf. on Intelligent Agent Technology*, Maebashi, Japan, 2001.

[13] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.

[14] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.