Coordination for Uncertain Outcomes using Distributed Neighbor Exchange

James Atlas
Computer and Information Sciences
University of Delaware
Newark, DE 19716
atlas@cis.udel.edu

Keith Decker
Computer and Information Sciences
University of Delaware
Newark, DE 19716
decker@cis.udel.edu

ABSTRACT

Coordination of agent activites in non-deterministic, distributed environments is computationally difficult. Distributed Constraint Optimization (DCOP) provides a rich framework for modeling such multi-agent coordination problems, but existing representations, problem domains, and techniques for DCOP focus on small (<100 variables), deterministic solutions. We present a novel approach to DCOP for large-scale applications that contain uncertain outcomes.

These types of real-time domains require distributed, scalable algorithms to meet difficult bounds on computation and communication time. To achieve this goal, we develop a new distributed neighbor exchange algorithm for DCOPs that scales to problems involving hundreds of variables and constraints and offers faster convergence to high quality solutions than existing DCOP algorithms. In addition, our complete solution includes new techniques for dynamic distributed constraint optimization and uncertainty in constraint processing. We validate our approach using test scenarios from the DARPA Coordinators program and show that our solution is very competitive with existing approaches.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms

Keywords

Distributed Constraint Optimization, Multi-agent Coordination, Distributed Problem Solving, Task Scheduling

1. INTRODUCTION

Distributed Constraint Optimization (DCOP) is a general problem representation for multi-agent systems. Recent advances in DCOP algorithm development have led to an increasing number of application domains and focus on DCOP techniques. Recent applications of DCOP to real-world problems include sensor networks[7], traffic flow cooperation [4], and event scheduling [6]. These existing problem domains for DCOP focus on small (<100 variables), deterministic domains. We present a novel approach to DCOP for large-scale applications that contain uncertain outcomes.

Cite as: Coordination for Uncertain Outcomes using Distributed Neighbor Exchange, James Atlas, Keith Decker, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 1047-1054

Scalability is one of the primary difficulties in the adoption of distributed constraint optimization techniques in real-world scale applications. In our results we present DCOP instances with over 1,000 variables, 5,000 constraints and 30 agents. These are not feasible problem sizes for most DCOP algorithms. We contribute a new local algorithm for DCOP that converges to high quality solutions in many fewer message passing cycles than existing algorithms.

Existing DCOP formalizations require deterministic utility outcomes for constraint assignments. This representation precludes reasoning about uncertainties within a DCOP algorithm, limiting the current scope of problem domains for DCOP. We introduce a new DCOP formalization that allows for non-deterministic constraint functions that evaluate to discrete utility distributions. These discrete utility distributions take the place of integer values in DCOP algorithms and allow the processing of non-linear functions over combined distributions.

Our approach makes four main contributions: a new DCOP algorithm for large-scale problems, an extension to the existing DCOP formalism for problems with uncertainty, a mapping from a general hierarchical planning and execution language (C-TAEMS) to this formalism, and an implementation of this mapping in a real-world dynamic environment.

First, we begin with an introduction of the existing DCOP formalism. We then introduce a new neighborhood exchange algorithm for DCOP that scales to problems involving thousands of variables and constraints. Next, we extend the existing DCOP formalization to include constraint functions with discrete utility distributions. We detail the mapping between problem instances in a hierarchical planning and execution language, CTÆMS, and the extended DCOP formalism. Finally, we show that our new DCOP algorithm can be used in conjunction with our extended DCOP formalism and our CTÆMS problem mapping to produce a full, real-world scale solution that is very competitive with the general approaches used in the DARPA Coordinators program.

2. EXISTING DCOP FORMALIZATION

DCOP has been formalized in slightly different ways in recent literature, so we will adopt the definition as presented in [9]. A Distributed Constraint Optimization Problem with n nodes and m constraints consists of the tuple $< X, D, U >$ where:

- $X = \{x_1,..,x_n\}$ is a set of variables, each one assigned to a unique agent
- $D = \{d_1,..,d_n\}$ is a set of finite domains for each variable

- $U = \{u_1,..,u_m\}$ is a set of utility functions such that each function involves a subset of variables in X and defines a utility for each combination of values among these variables

An optimal solution to a DCOP instance consists of an assignment of values in D to X such that the sum of utilities in U is maximal. Problem domains that require minimum cost instead of maximum utility can map costs into negative utilities. The utility functions represent soft constraints but can also represent hard constraints by using arbitrarily large negative values.

3. DISTRIBUTED NEIGHBOR EXCHANGE ALGORITHM

We now introduce a new algorithm to solve large-scale distributed constraint optimization problems, the Distributed Neighbor Exchange Algorithm (DNEA). We developed this algorithm after finding that existing DCOP algorithms that can actually run on larg-scale problems took many message passing cycles to converge to solutions. DNEA is a local neighborhood search algorithm that exchanges potential gains for multiple assignments to all neighbors in each cycle.

We begin with a brief motivation from the problem domain we will present later that highlighs the need for scalability and dynamic variable assignment. We continue with the description of the algorithm and its implementation, and present comparisons for different algorithms on static versions of the smallest problem sets in our testing.

3.1 Scalability and Dynamism

Scalability is one of the primary difficulties in the adoption of distributed constraint optimization techniques in real-world scale problems. Using our mapping, the smallest problem sets we present in our results involve 35 variables with 75 constraints over 6 agents. The values for variables/constraints/agents rapidly increase for other problems in our result set to 120/210/25 and up to 1400/5000/32. These are not feasible problem sizes for most DCOP algorithms. Any algorithm that scales exponential with respect to any graph characteristic is infeasible for this problem domain for anything larger than the smallest problems. Thus, we must employ local algorithms for distributed constraint optimization.

Another very large hurdle to cross for real-world scale problems is dynamic changes to the problem during computation. Updates to the set of variables, the values for different domains, and even to the constraint evaluations require algorithms to propagate solution changes very quickly. Current complete algorithms including ADOPT[7] and DPOP[9] do not adequately handle these cases, and in the worst case incur the same exponential performance cost as in the static case. Local algorithms such as DBA, DSA, and MGM are able to respond very quickly to underlying problem changes, but often do not arrive at high utility assignments or may take too many cycles to converge to a solution.

3.2 Algorithm Phases

Our algorithm is similar in phases to other local value exchange based algorithms, including MGM, SCA, DBA, and max-sum [1, 2]. These algorithms exchange current variable assignments with neighbors, compute a maximization function based on neighboring assignments, and then choose to update the local variable assignment. There are two phases to our algorithm, value exchange and neighborhood utility exchange. An example of a simple execution path is shown in Figure 1.

3.2.1 Value Exchange

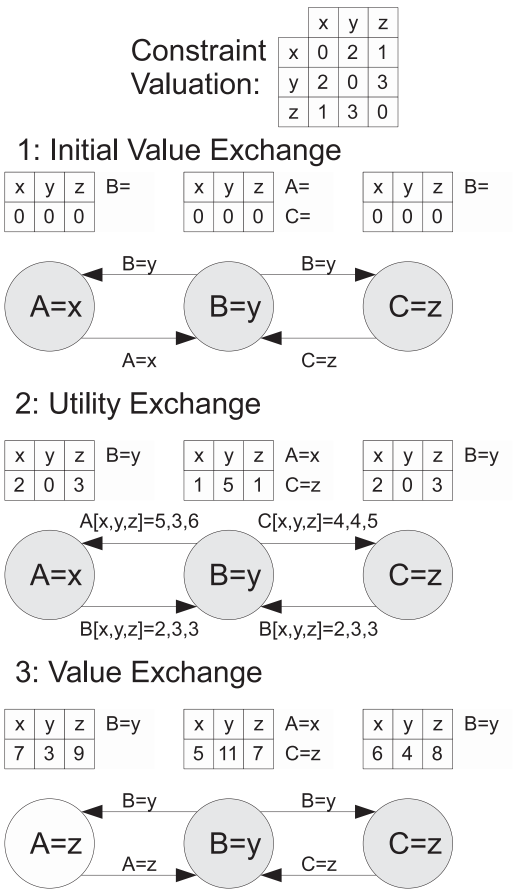

Figure 1: DNEA in action. Given constraint valuations shown at top and a random starting assignment of A=x, B=y, and C=z, DNEA finds the optimal assignment in one round of exchanges. Note how the utility exchange message from B to A in step 2 contains aggregated utility from C.

For each variable X with domain D_X in the DCOP instance, an associated agent chooses a value to assign the variable from a set of neighborhood utility valuations. At the beginning these valuations are zero, so the agent randomly assigns a value to X. On subsequent iterations, the agent will have received a utility exchange message for each variable that is a neighbor in the constraint graph. Each utility exchange message includes a utility value for each possible assignment in D_X for X. Each of these values represents the best local utility the neighboring variable can achieve for each possible value of X. The agent then sums all of the neighbor valuations together and adds the sum of local valuations for the vector of possible assignments in X:

$$U_X = localSum_X + exchangeSum_X \quad (1)$$

Note that each term in equation 1 is a vector of corresponding size to the possible assignments in X. The agent then chooses with probability p to change the value of X to the maximum assignment in U_X.

3.2.2 Utility Exchange

When the agent for variable X receives all new values for neighboring variables of X, it calculates a set of utility exchanges. First, the agent calculates the local utility at X for each possible assignment in D_X to X given the current neighboring variable assignments $\beta \in N_X$ (producing the vector $localSum_X$ in equation 1):

$$localSum_X = \forall\alpha \in D_X \sum_{\beta \in N_X} U(\alpha, \beta) \tag{2}$$

Then, for each neighbor Y, the agent calculates its optimal assignment to X for each value $\gamma \in D_Y$ given the current neighboring variable assignments minus the current assignment β for Y:

$$\begin{aligned} exchangeSum_X(Y) = \forall\gamma \in D_Y \arg\max_{\alpha \in D_X} \Big[\\ localSum_X(\alpha) - U(\alpha, \beta) + U(\alpha, \gamma)\Big] \end{aligned} \tag{3}$$

This maximum local utility at X for each value in D_Y is sent to the agent for variable Y (term $exchangeSum_X$ in equation 1). After the agent has calculated and sent all of these utility exchange messages, it waits for updated value exchange messages from its neighbors.

3.3 Complexity Analysis

The Distributed Neighborhood Exchange Algorithm has a similar flow of execution to other local value exchange algorithms, but calculates a merge of utility valuations for all neighbors in a single cycle; this drastically decreases the number of message passing cycles required for convergence. Maximum computation per variable per phase for DNEA is $O(|N_X| \cdot |D_X|)$ for value exchange and $O(\sum_{Y \in N_X} |D_Y| \cdot |D_X|)$ for utility exchange. Because it is based on local search techniques, it is amenable to dynamically changing problem domains.

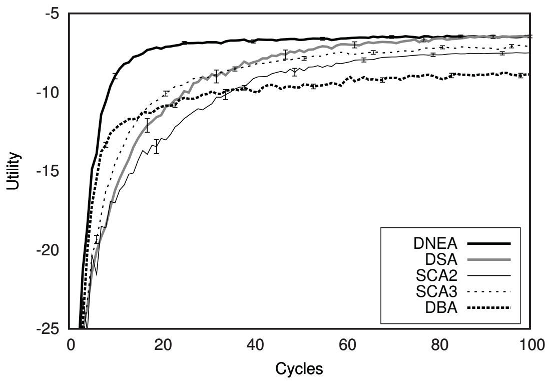

Figure 2: Graph Coloring Problems: 40 variables. Utility over message passing cycles shown for DNEA verses other local search algorithms.

3.4 Comparison with other DCOP algorithms

We compare messaging metrics between algorithms because it is assumed that the processing done at each local DCOP agent is sufficiently small in comparison to message transit time (a factor of both latency and bandwidth). The overall run-time of each algorithm is dominated by message passing cycles.

We tested DNEA against four local search algorithms, DSA and DBA from [13], and SCA2 and SCA3 from [1]. We made one optimization for these algorithms that significantly improved their performance: all variables were allowed to change (or offer a group change) values if the gain was greater than or equal to zero (instead of only greater than). We chose these local search algorithms because they are able to scale to problem sizes containing 1,000+ variables. We performed comparisons for standard graph coloring problems that can be found in the DCOP repository at USC [8]. There are 25 problems that contain 40 variables each, with 120 constraints, and 3 possible colors for each variable, with constraint violations for connected variables of the same color worth -1 utility. Each problem was run 10 times for each algorithm with the same set of 10 random number seeds and results are shown as the average global utility at each message passing cycle in Figure 2.

3.4.1 Performance: Message Passing Cycles

We observe that DNEA reaches a high utility by cycle 20 and the highest utility by cycle 40. This was the goal in the creation of DNEA, to be able to achieve high utility with very few message passing cycles while keeping polynomial time calculation for each cycle. We do not show it here, but eventually SCA2 and SCA3 catch up; however, it takes well over 500 cycles to do so. Since fast convergence is a necessity for dynamically changing problem domains, DNEA offers a compelling option for such domains.

3.5 Extension for Exchange Depth

In the description of DNEA in Section 3.2.2 we note that the Utility Exchange phase passes information only between immediate neighbors. We can extend this exchange of information to additional nodes while maintaining our $O(\sum_{Y \in N_X} |D_Y| \cdot |D_X|)$ runtime per utility exchange phase by performing d utility exchange phases between value assignment phases. We define exchange depth, d, as the number of constraint hops between the original sender of a utility exchange message and the farthest receiver to which the message propagated. Propagation of utility exchange for $d > 1$ is handled exactly as in Phase 2 of the original algorithm.

3.5.1 Exchange Depth Results

In our tests, increasing exchange depth does not help on the problems tested in Figure 2, mainly because they do not have a high constraint density, averaging 3 constraints per node. To test highly constrained graphs, we tested depths 1 to 6 in Figure 3 on large graph coloring problems with 500 variables and an average of 10 constraints per node. We immediately see that there is a benefit for exchanging information to additional depths. At depth 2 we see substantial gains over depth 1. We note the stair-stepping behaviour of DNEA when using exchange depths greater than 1; since DNEA does not assign new values while it is waiting for exchanged utility to propagate, we see the width of the steps increases as the exchange depth increases. We also observe that the larger the depth value, the more cycles are required to converge. For our future tests we will use a depth setting of 2 since it represents a profitable tradeoff between convergence speed and high utility.

4. DCOP WITH UTILITY DISTRIBUTIONS

Existing DCOP formalizations require deterministic utility outcomes for constraint assignments. This representation precludes reasoning about uncertainties within a DCOP algorithm, limiting the current scope of problem domains for DCOP. This section introduce a new DCOP formalization that allows for non-deterministic constraint functions that evaluate to discrete utility distributions.

We can extend the DCOP problem formalization to include uncertainty by allowing constraint evaluation functions to return a distribution instead of a single value. A global optimum is now an optimal distribution instead of a maximum (or minimum) sum. To evaluate the optimality of a distribution, evaluation criteria must be formalized. The optimal evaluation function may not be the same

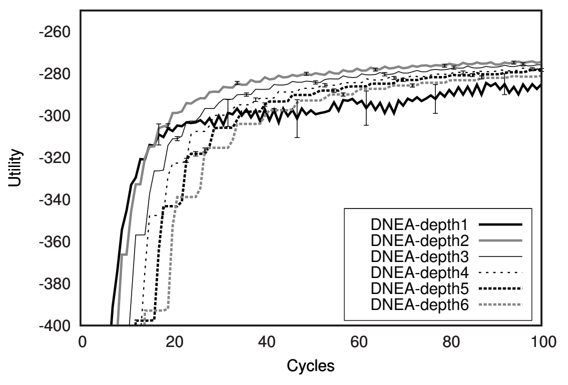

Figure 3: Large Graph Coloring Problems: 500 variables. Utility over message passing cycles shown for DNEA using exchange depths from 1 to 6.

for all problems for all agents. Thus we must include the evaluation function as part of the extended DCOP problem. We extend our previous DCOP formalization for this:

- $U = \{u_1,..,u_m\}$ is a set of utility functions such that each function involves a subset of variables in X and defines a utility distribution for each combination of values among these variables
- $u = \{(u_p^1,\ u_v^1),..,(u_p^t,\ u_v^t)\ \}$ is a distribution of probabilities and values such that $\sum_{r=1}^{t} u_p^r = 1$
- $E = \{e_1,..,e_n\}$ is a set of evaluation functions for each variable that reduce a utility distribution to a single utility value; $e(u) = v$ where v is a single utility value

This extension requires extra computation and memory proportional to the maximum allowed size of a distribution. However, it allows processing of non-linear functions over combinations of utility distributions, which proves very useful to implementing risk averse behavior in our application domain as shown in Section 5.6.

5. C-TÆMS COORDINATION PROBLEM

Multi-agent task planning and scheduling problems require a rich language for domain representation. In this section, we map problems in one such language, C-TÆMS, to our formalization of DCOP with uncertainty.

5.1 C-TÆMS

The original TÆMS (Task Analysis, Environment Modeling, and Simulation) language was developed to provide a domain independent, quantitative representation of the complex coordination problem [3]. A C-TÆMS problem instance, sample shown in Figure 4, contains a set of agents and a hierarchically decomposed task structure. Nodes in the graph are either complex tasks (internal nodes) or primitive methods (leaf nodes). Each node may have temporal constraints on the earliest start time and the deadline. Nodes may also have non-local effect (NLE) constraints that represent hard (enables and disables) and soft (facilitates and hinders) node relationships. Methods have probabilistic outcomes for duration, quality, and cost. Tasks have a quality accumulation function (QAF) that describes how quality accrues at the task based on the quality of its subtasks and methods.

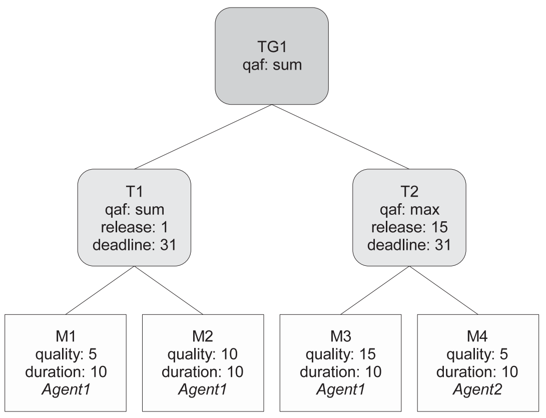

Figure 4: An example C-TÆMS problem instance.

5.2 Existing C-TÆMS mapping

A mapping for a subset of C-TÆMS to DCOP is proposed in [11]. The mapping using our formalization is:

- X = Each method is assigned to a unique variable.
- D = Unique domains for each variable containing all possible start times for the method assigned to the variable.
- U = Three types of utility functions:
  - Mutex constraints on all pairs of methods that share the same agent
  - For an NLE between two nodes, N_1 and N_2, all methods in the subtree of N_1 have a precedent constraint with all methods in the subtree of N_2
  - Unary soft constraints on each method that apply a cost if the method is not scheduled

While this mapping is a good start, it is severely limited. It allows only sum, min, and max QAFs, and all QAFs in the same problem must be of the same type (no mixing sum with max QAFs, or taking the max over a set of sums). It also only allows enables NLEs and requires deterministic task outcomes, so it cannot handle NLEs that are contingent upon the outcome of a method or method's with non-deterministic quality or duration. Because it requires deterministic task outcomes, this also precludes advanced uncertainty reasoning which could avoid risky inter-agent schedules.

We propose a new mapping that includes the full set of QAFs, NLE functions, and uncertainties used in the DARPA Coordinators project.

5.3 Proposed mapping

Our proposed mapping for C-TÆMS to DCOP can be broken into two distinct parts: variable and constraint mappings. In the variable mappings we describe the domain values for the variable. For the constraints we present the utility function rules for involved variable values. All constraints are binary constraints in this mapping and produce discrete utility distributions as presented in Section 4.

5.4 Variables

Variables are created for each method and task in the C-TÆMS problem. In addition, a special end-time variable is created for each task with an outgoing NLE at or above it in the structure.

5.4.1 Methods

Method variables are created with all possible start times as values and an additional value for *not scheduled*. Additional values are created for each permutation of a modifier that affects the method, including a synchronization point and all incoming NLEs.

5.4.2 Tasks

Task variables can have several different sets of values depending on the type of QAF assigned to the task. These values describe how quality will accumulate from subtasks and methods. All task variables contain values for *no execution* and *execution allowed no quality* that force children not to execute or to not accumulate quality (respectively). Max, min, and exactlyone QAFs contain values representing the children that will accumulate quality. Sum QAFs and sumand QAFs have a single sum quality value. Sync sum QAFs contain values for all possible synchronized start times for descendant methods. All task domains with a sumand QAF ancestor also have a modifier flag to force execution if set to a quality accumulating value.

5.4.3 NLEs

Non-local effects do not have their own variables. However, if a non-local effect originates at a task, then a special task end time variable is created for the task. This task end time variable contains all possible ending times for descendant methods and a value for *not scheduled*.

5.5 Constraints

Constraints are created between each related node in the problem structure. There are four types of relationships we create constraints for: task-subtask, task-method, method-method at the same agent, and to and from nodes in a NLE. In addition special constraints are created for synchronization points. The actual implementation values for constraints will be problem specific, with the following defined values: Q_{max} is an upper bound for the maximum quality of the entire problem structure and Q_{M1} is the quality distribution for method $M1$. All hard constraints are enforced using $-Q_{max}$ as the utility for a constraint violation.

5.5.1 Task-Subtask

A task-subtask constraint enforces a correct accumulation of quality up from the methods through each of the QAFs to the root. Generally, the constraint enforces that a subtask may only accumulate quality if the task allows it. Additionally, this constraint propagates information about the forced execution modifier flags so that a subtask knows if the task is relying on it to produce some quality. If a task is assigned to accumulate quality, the implementation depends on the type of QAF.

5.5.2 Task-Method

A task-method constraint accumulates the quality for properly scheduled methods. Thus for a valid scheduled method, this constraint returns a value of Q_M. The value for Q_M includes any modification indicated by incoming NLE flags (for example it will be twice the original method quality if a facilitation flag is set with a factor 2). A constraint violation occurs if any method is not scheduled and is in the set of methods selected for scheduling by the task value. Again, the implementation for accumulating quality depends on the type of QAF.

5.5.3 Method-Method

A method-method constraint ensures that an agent is not scheduled to execute two methods at the same time. A mutex constraint is created between all pairs of methods at an agent. We prune all mutex constraints that have no possible overlap. For assignments that would cause overlap, the utility returned is equal to $\max_{k \in C_M} -QM_k$ where C_M is the set of methods involved in the constraint.

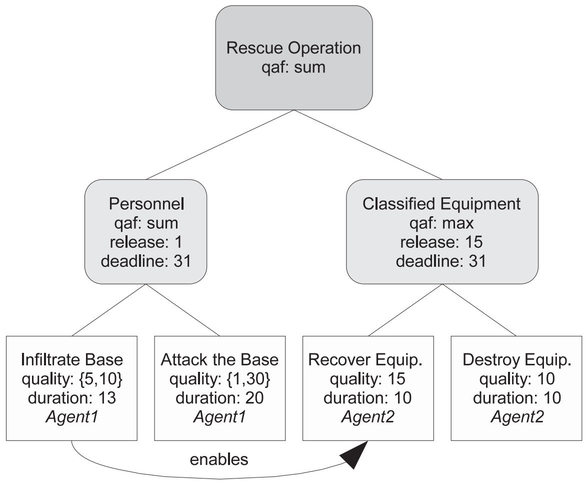

Figure 5: A real-world C-TÆMS problem instance with uncertainty.

5.5.4 Non-Local Effects

A non-local effect (NLE) constraint may occur between tasks, methods, or both. In keeping with the CTÆMS specification, we decompose non-local effect constraints into only task-method and method-method relationships. A task-task constraint between T_1 and T_2 by definition behaves the same way as if the constraint existed from T_1 to all descendant methods of T_2. For task-method NLEs we use the special end time variable to calculate the time at which the NLE is active. For method-method NLEs we use the start time plus the duration (which is a distribution) to determine when the NLE is active. Active NLEs can help produce positive quality or may produce constraint violations depending on the type of the NLE.

5.6 Uncertainty in the C-TÆMS scheduling problem

Uncertainties for task characteristics in C-TÆMS are represented as discrete distributions of values, primarily over duration and outcome quality of methods. To illustrate the usefulness of our model of DCOP with Utility Distributions for this domain, lets update the simple CTÆMS structure in Figure 4 with a real scenario with uncertainty. The new scenario is shown in Figure 5. In this example, Agent1 does not know what the quality outcomes of *Infiltrate the Base* (M1) and *Attack the Base* (M2) will be a priori. Agent2 knows the outcomes for *Recover Equip.* (M3) and *Destroy Equip.* (M4), but is dependant on Agent1's choice of execution because of the enables relationship. Based on the deadlines involved, the agents can either execute M1+M3 or M2+M4 (M1+M4 is possible but clearly inferior to M1+M3). How does Agent1 choose whether to execute M1 or M2? Assuming uniform probability distributions, let us consider three evaluation functions:

- *Average Value (Exponential Value=1)* - For M1+M3 this is $(0.5 * 5 + 0.5 * 10) + 15 = 22.5$ and for M2+M4 this is $(0.5 * 1 + 0.5 * 30) + 10 = 25.5$. M2+M4 is the optimal choice here.

- *Interval Average (0 to 0.5)* - For M1+M3 this is $5 + 15 = 20$ and M2+M4 as $1 + 10 = 11$. Thus M1+M3 is the optimum choice.
- *Parametric Integration (Risk Averse, $2 * (1 - x)^2$)* - For M1+M3 it evaluates to $(0.875{*}5{+}0.125{*}10){+}15 = 20.625$ and for M2+M4 it is $(0.875{*}1{+}0.125{*}30){+}10 = 14.625$. M1+M3 is the best choice for this function as well.

We see that the average value function does not take any risk into account. The second function is mildly risk averse but doesn't really take into account the whole distribution. The third function is strongly risk averse and does evaluate over the whole distribution. For all of our presented results we use the third function.

5.7 Static C-TÆMS Internal Comparison

To determine what algorithms and configurations to use for the full DARPA simulation platform, we apply our C-TÆMS mapping in a static scenario (the agents are searching for the schedule with best expected utility but no methods are executed and scenario time stays at 0). This scenario was run in two parts, one randomly generating an initial schedule for the agents, and one using an initial schedule generated by an offline deterministic solver. Each scenario was run 10 times using different random seeds. Details of the scenarios ("OptOP5PMix" and "OptBigReMix") are provided in Section 6. Results are shown in Figures 6, 7, 8, and 9.

It is clear that for all scenarios, including those with and without initial schedules, DNEA at exchange depth 2 converges quickly to the schedule with the best estimated utility. We see this beginning at cycle 45 in Figure 7, cycle 50 in Figure 6, and cycle 25 in Figure 8. This result led us to use DNEA at exchange depth 2 as the base algorithm in our Coordinators simulation.

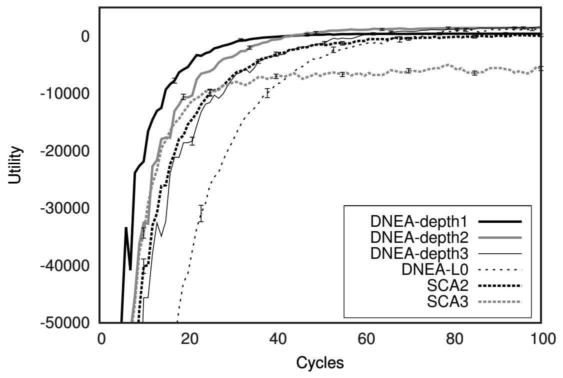

Figure 6: OptOP5PMix: ∼100 variables, random initial schedule. DSA not shown as it never exceeds -50000.

5.8 Existing Non-DCOP Approaches

Three teams participated in the second phase testing of DARPA's Coordinators project, which used C-TÆMS scenarios for their test suite, and used very different approaches. Detailed descriptions of the approaches can be found in [5, 12, 10]. The best performing team used a risk avoidance strategy based on predictability and criticality metrics (PCM). These metrics did not aim to optimize an approximate global utility function, but instead minimized chances for conflicts while opportunistically inserting changes that did not negatively impact the current schedule. A second approach used simple temporal networks to create a flexible time schedule of methods that changes over time using constraint propagation (FTS). Agents choose to modify the schedule when speculation indicates local utility gain is greater than neighboring utility loss. The third approach used distributed MDPs to approximate optimal execution policies (MDP).

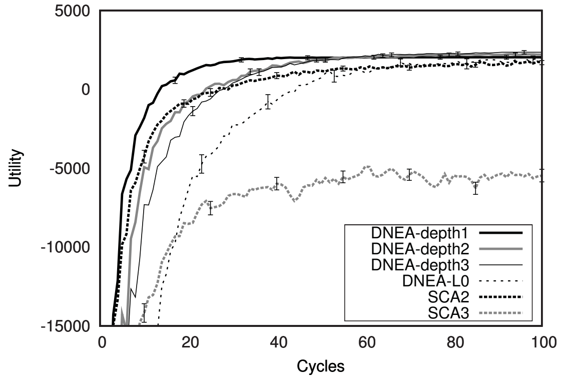

Figure 7: OptOP5PMix: ∼100 variables, given initial schedule. DSA not shown as it never exceeds -15000.

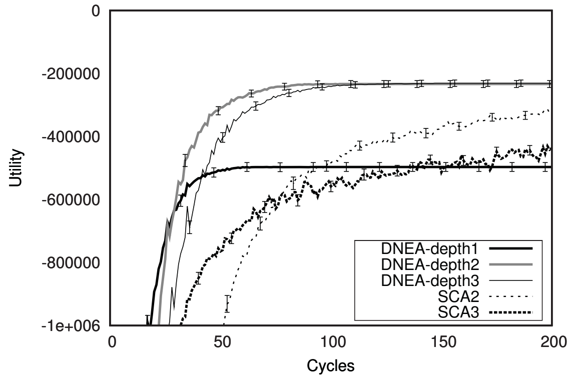

Figure 8: OptBigReMix: ∼1000 variables, random initial schedule

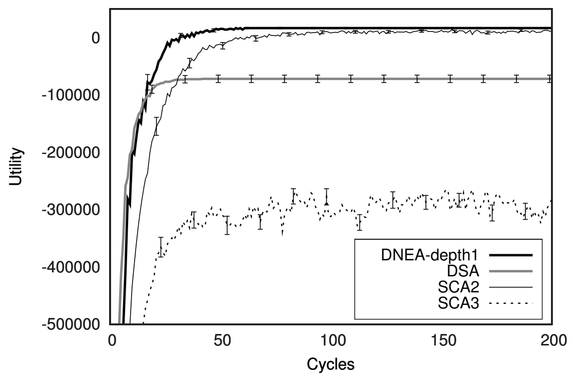

Figure 9: OptBigReMix: ∼1000 variables, given initial schedule. DNEA depths 2 and 3 not shown because they are equivalent to depth 1 for this test.

Our approach is most similar to the FTS approach because we use DCOP to optimize over a flexible execution schedule. However, we design the constraints with discrete value distributions and reason over them using a risk aversion function. This function provides a similar role to the metrics used in the PCM approach as it is not a direct approximation of the global utility function.

6. RESULTS

We now present results using our DCOP-based multi-agent solution to challenging, real-world coordination problems that were part of DARPA's Coordinators program. Our solution required many additional lower level components to interact between the test simulation and our underlying DCOP mapping and algorithm. These components are not presented here, but are available upon request from the authors.

6.1 Timed Simulation

The actual Coordinators project uses a simulation framework based on simulated time ticks. A master simulation agent tracks a schedule of agent requests for method executions. The simulation agent sends a pulse message to each agent for each simulated time tick. Actual simulation runs set the simulated time per tick to one second of real time, with the first methods to execute typically available beginning at tick 30. Agent communication is bounded by a message passing infrastructure that allows only around 20-50 messages to be sent from any agent per second. These computational and communication bounds place tight restrictions on the agent reasoning capabilities.

We ran tests using the Coordinators simulation for four sets of problems. Each scenario within a problem set had similar characteristics to other scenarios in the same set:

1. *OptOP5PMix* - 50 medium sized scenarios (average of 26 agents, 104 tasks/methods, 16 NLEs, 112 ticks of execution, and no C-TÆMS meta-events); contains initial schedules.

2. *OptBigReMix* - 8 large sized scenarios (average of 15 agents, 1018 tasks/methods, 137 NLEs, 1680 ticks of execution, and no C-TÆMS meta-events); contains initial schedules.

3. *BakeA25Mix* - 32 large sized scenarios (average of 25 agents, 1283 tasks/methods, 192 NLEs, 589 ticks of execution, and 16.5% of the task structure updated with C-TÆMS meta-events); contains initial schedules.

4. *RealWorld-HighDyn* - 32 large sized scenarios (average of 33 agents, 1257 tasks/methods, 148 NLEs, 471 ticks of execution, and 27.5% of the task structure updated with C-TÆMS meta-events); does not contain initial schedules.

6.2 Coordinators Simulation

Result comparisons for the first two problem sets are shown in Figure 10 and Figure 11 as the percent of quality achieved by each team with respect to the known optimal quality as determined by an offline, centralized MDP solver (such an MDP solver does not work in the real-time scenario because it takes much more time to construct this policy than allowed in the scenario; additionally for the second two problem sets the offline MDP solver simply cannot solve problems of that magnitude due to exponential time complexity). Result comparisons for the second two problem sets are shown in Figure 12 and Figure 13 as the percent of quality achieved by each team with respect to the best quality achieved by any team for each scenario.

For comparison, we show a baseline strategy that used no coordination and simply executed the initial schedule given in the problem, and opportunistically inserted unscheduled methods when they would not conflict with future scheduled methods (labeled Naive). We do not show the Naive approach on the meta-event scenarios as it achieved 0 quality because it does not process meta-events. We also show results for the three teams that took part in the Coordinators second phase testing (labeled using abbreviations from Section 5.8). Our DCOP approach is shown using the Distributed Neighbor Exchange Algorithm (DNEA) with exchange depth equal to 1 and 2.

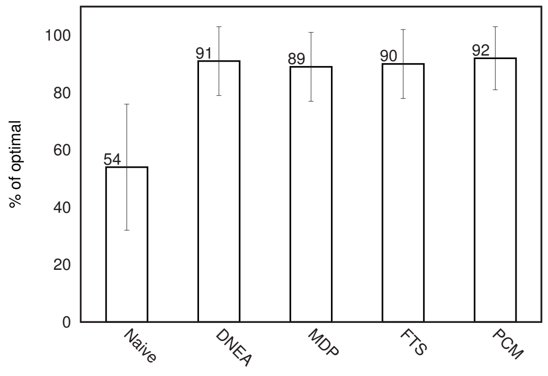

Figure 10: OptOP5PMix: Solution Quality as % of optimal. Error bars show standard deviation.

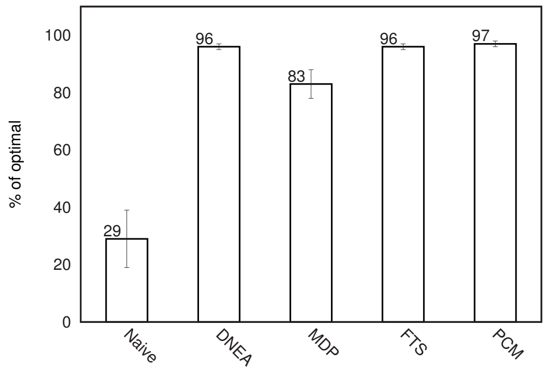

Figure 11: OptBigReMix: Solution Quality as % of optimal. Error bars show standard deviation.

7. ANALYSIS

We see clearly that our DCOP approach can achieve much higher performance than the naive baseline in Figures 10 and 11. Table 1 shows the total number of best quality scores achieved by each team for the problem sets (ties are possible) where we used normalized results because the optimal value is unknown.

The "BakeA25Mix" and "RealWorld-HighDyn" scenario subsets were among the most difficult sets and showed the largest disparities between the top performing DARPA team (the PCM team) and the rest of the teams.

Our DCOP approach achieved a very high level of performance for this application domain and was very competitive with the

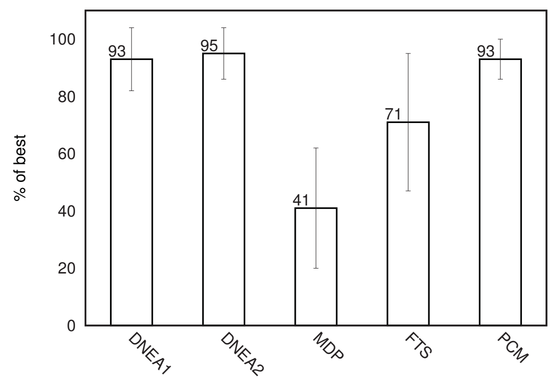

Figure 12: BakeA25Mix: Solution Quality as % of best achieved by any solution. Error bars show standard deviation.

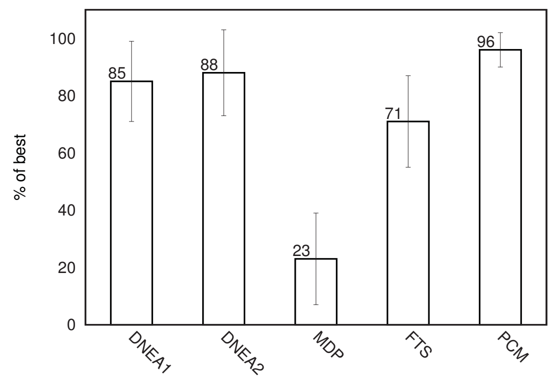

Figure 13: RealWorld-HighDyn: Solution Quality as % of best achieved by any solution. Error bars show standard deviation.

PCM team. We were able to match their performance on the "BakeA25Mix" scenarios and managed to outperform their solution on roughly half of the problems in this set. Our DCOP approach also performed very well on the "RealWorld-HighDyn" scenarios, but fell just short of the PCM team's performance. However, the DCOP approach was able to achieve higher performance on about a third of the scenarios within this set. The major difference between these two scenario sets is that the "RealWorld-HighDyn" scenarios do not provide the agents with any initial schedule. If our DCOP agent does not receive an initial schedule from the scenario it randomly makes variable assignments to the methods such that no local method overlaps execution. However, it does not consider NLEs, task tree composition, or uncertainties about the method when making this initial assignment. This is the main reason why our solution does not do better on the "RealWorld-HighDyn" scenarios, as we were able to see by running the scenarios multiple times that our variance in quality was largest on these problems.

8. CONCLUSION

We have presented four major contributions to DCOP based solutions for complex, real-world multi-agent systems domains. We introduced a new scalable DCOP algorithm, the Distributed Neighbor Exchange Algorithm, that achieved high quality solutions with extremely fast convergence. We developed a new DCOP formalization for domains with uncertainty and showed how a complex multi-agent planning and execution language can be mapped to this formalism. Finally, we combined our contributions into a DCOP-based solution to challenging scenarios that were a part of DARPA's Coordinators program. We achieved a high level of performance for this domain, comparable to the best performing Coordinators team for many problems. These results show that DCOP techniques can be used to solve large-scale, real-world problems, and that continued work in this direction is very promising.

Approach	BakeA25Mix	RealWorld-HighDyn
DNEA1	7	5
DNEA2	12	15
MDP	0	0
FTS	0	0
PCM	13	12
Total Possible	32	32

Table 1: Number of problems in each scenario which an approach achieved the best performance in the comparison group (ties possible).

9. REFERENCES

[1] E. Bowring, J. P. Pearce, C. Portway, M. Jain, and M. Tambe. On k-optimal distributed constraint optimization algorithms: new bounds and algorithms. In *AAMAS '08*, pages 607–614, 2008.

[2] A. Farinelli, A. Rogers, A. Petcu, and N. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS '08*, pages 639–646, 2008.

[3] B. Horling, et al. The TAEMS White Paper, January 1999.

[4] R. Junges and A. L. C. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *AAMAS '08*, pages 599–606, 2008.

[5] R. T. Maheswaran, et al. Predictability & criticality metrics for coordination in complex environments. In *AAMAS '08*, pages 647–654, 2008.

[6] R. T. Maheswaran, et al. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS '04*, pages 310–317, 2004.

[7] P. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. In *AIJ*, pages 149–180, 2005.

[8] J. P. Pearce. USC DCOP repository, 2005.

[9] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *IJCAI '05*, pages 266–271, 2005.

[10] S. F. Smith, A. Gallagher, and T. Zimmerman. Distributed management of flexible times schedules. In *AAMAS '07*, pages 1–8, 2007.

[11] E. Sultanik, P. J. Modi, and W. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI '07*, 2007.

[12] S. Witwicki and E. Durfee. Commitment-driven distributed joint policy search. In *AAMAS '07*, pages 1–8, 2007.

[13] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS '03*, pages 185–192, 2003.