On Modeling Multiagent Task Scheduling as a Distributed Constraint Optimization Problem

Evan A. Sultanik and **Pragnesh Jay Modi** and **William C. Regli**
{eas28,pmodi,wregli}@cs.drexel.edu
Department of Computer Science, Drexel University, Philadelphia, USA

Abstract

This paper investigates how to represent and solve multiagent task scheduling as a Distributed Constraint Optimization Problem (DCOP). Recently multiagent researchers have adopted the C_TÆMS language as a standard for multiagent task scheduling. We contribute an automated mapping that transforms C_TÆMS into a DCOP. Further, we propose a set of representational compromises for C_TÆMS that allow existing distributed algorithms for DCOP to be immediately brought to bear on C_TÆMS problems. Next, we demonstrate a key advantage of a constraint based representation is the ability to leverage the representation to do efficient solving. We contribute a set of pre-processing algorithms that leverage existing constraint propagation techniques to do variable domain pruning on the DCOP. We show that these algorithms can result in 96% reduction in state space size for a given set of C_TÆMS problems. Finally, we demonstrate up to a 60% increase in the ability to optimally solve C_TÆMS problems in a reasonable amount of time and in a distributed manner as a result of applying our mapping and domain pruning algorithms.

1 Introduction

The coordinated management of inter-dependent plans or schedules belonging to different agents is an important, complex, real-world problem. Diverse application domains such as disaster rescue, small-team reconnaissance, security patrolling and others require human teams to schedule their tasks and activities in coordination with one another. It is envisioned that automated planning and scheduling processes in the form of agents operating on portable computing hardware can assist human teams in such domains. The problem is inherently a distributed one; an agent that manages the schedule of a human user must effectively manage the interdependencies between its schedule and the schedules of other users by exchanging information and coordinating with other agents. No single agent has a global view of the problem; instead, each must make local planning and scheduling decisions through collaboration with other agents to ensure a high quality global schedule.

A key step towards addressing the distributed scheduling problem is to devise appropriate representations. In this paper, we have three goals in mind. We desire a representation that (a) captures the problem's distributed multiagent aspects, (b) gives formal guarantees on solution quality, and (c) is amenable to distributed solving by existing methods.

In regard to goal (a), the Coordinators Task Analysis Environmental Modeling and Simulation (C_TÆMS) representation [Boddy *et al.*, 2006] is a general language (based on the original TÆMS language [Decker, 1996]) that was jointly designed by several multiagent systems researchers explicitly for multiagent task scheduling problems. C_TÆMS is a Hierarchical Task Network (HTN) style representation where the task nodes in the network have probabilistic utility and duration. C_TÆMS is an extremely challenging class of scheduling problem and has been adopted as a common challenge problem domain representation for distributed multiagent task scheduling research.

We propose a *distributed constraint reasoning* approach for solving problems expressed in the C_TÆMS language. Our methodology is one of problem transformation in which we create a mapping that converts a C_TÆMS instance into an instance of the Distributed Constraint Optimization Problem (DCOP) [Modi *et al.*, 2005]. DCOP was devised to model constraint reasoning problems where several agents must communicate to ensure that solutions are globally optimal. Several distributed algorithms for DCOP currently exist in the literature including distributed dynamic programming (DPOP [Petcu and Faltings, 2005]), cooperative mediation (OptAPO [Mailler and Lesser, 2004]) and asynchronous backtracking (Adopt [Modi *et al.*, 2005]).

The expressivity of C_TÆMS comes at a cost; it is arguable that the primary concern in the design of the C_TÆMS language was to comprehensively express the complexities of the distributed scheduling problem, while less concern was placed on goals (b) and (c) enumerated above. Indeed, current distributed solution techniques cannot be applied straightforwardly or are unable to provide solution quality guarantees. Thus, to make progress on this difficult problem, we contribute a set of representational compromises in the form of a subset of C_TÆMS, called DSC_TÆMS, that is soluble by current DCOP algorithms.

Finally, we demonstrate three key advantages of our constraint based approach. First, we provide a mapping that au-

tomatically transforms a C_TÆMS problem into an *equivalent* DCOP. That is, the optimal solution to a given C_TÆMS problem is also the optimal solution to the DCOP obtained by applying our problem transformation mapping, *i.e.*, our mapping preserves the problem exactly. Second, arguably one of the primary advantages of adopting a constraint based approach is the ability to apply existing constraint propagation techniques from the constraint programming literature. Thus, we leverage our problem transformation by applying constraint propagation pre-processing techniques to perform domain pruning to significantly reduce the search space. We demonstrate a 96% reduction in state space size for a set of benchmark problems. Finally, another advantage of our approach is that by mapping the C_TÆMS problem into a DCOP, we are able to immediately apply existing solution techniques for DCOP to the C_TÆMS problem domain.

2 Related Work

Numerous techniques have been employed to address the problem of multiagent coordination and scheduling. Musliner, *et al.*, map C_TÆMS to a Markov decision process which is then used to generate a policy dictating when and how agents execute methods [Musliner *et al.*, 2006]. Musliner, *et al.*, also propose distributed constraint satisfaction as a method for negotiating a feasible schedule by maximizing previously-calculated local expected qualities [Musliner *et al.*, 2006]. Smith, *et al.*, address the problem of dynamic schedule revision (due to changes in the model) using Simple Temporal Networks [Smith *et al.*, 2006]. Phelps and Rye address this same problem through a domain-independent implementation of Generalized Partial Global Planning, in which *proxy methods* allow for distributed approximation of the performance characteristics of potentially complex tasks [Phelps and Rye, 2006].

Constraint propagation is a general family of techniques that check a constraint based representation for consistency to reduce problem search space, often in a pre-processing phase. Barták provides a survey of the general topic in [Barták, 2001]. One of the primary advantages of adopting a constraint based approach is the ability to apply constraint propagation techniques such as node, arc and path consistency. As such, constraint propagation is well-studied with several algorithms available that vary in complexity and pre-processing resource requirements. The first arc consistency algorithms were formalized in [Mackworth, 1977] including the widely used AC-3 algorithm. Bessière, *et al.*, improve on average-case efficiency with the AC-6 and AC-7 algorithms [Bessière *et al.*, 1999]. More recently, researchers have introduced distributed methods for arc consistency processing including the DisAC-9 algorithm [Hamadi, 2002] and the DMAC protocol [Silaghi *et al.*, 2001].

3 Formalization

This section formalizes the notion of a DCOP and specifies the subset of C_TÆMS on which we focus.

3.1 Distributed Constraint Optimization

A DCOP is a subclass of distributed decision processes in which a set of agents are responsible for assigning the values of their respective variables subject to a set of constraints. A DCOP can be defined as a tuple $\langle A, V, \mathscr{D}, F, \alpha, \sigma \rangle$, where

- A is a set of agents;
- V is a set of variables, $\{v_1, v_2, \ldots, v_{|V|}\}$;
- $\mathscr{D}$ is a set of domains, $\{D_1, D_2, \ldots, D_{|V|}\}$, where each $D \in \mathscr{D}$ is a finite set containing the feasible values for its associated variable;
- F is a set of $|V|^2$ cost functions, one for each pair of variables, such that $f_{ij} : D_i \times D_j \rightarrow \mathbb{N}_0 \cup \{\infty\}$. Each cost function maps every possible variable assignment to its associated cost, for all pairs of variables and for all possible assignments. These functions can also be thought of as constraints;
- α is function $\alpha : V \rightarrow A$ mapping variables to their associated agent. $\alpha(v_i) \mapsto a_j$ means that it is a_j's responsibility to assign the value of v_i; and
- σ is a function $\sigma : F \rightarrow \mathbb{N}_0$ that aggregates the individual costs[1].

The objective of a DCOP is to have each agent assign values to its associated variables in order to minimize $\sigma(F)$ for a given assignment of the variables. This definition was adapted from [Davin and Modi, 2005] and has been modified for the sake of brevity, clarity, and applicability.

3.2 C_TÆMS

C_TÆMS is a derivative of the TÆMS modeling language [Decker, 1996] and can be used for representing instances of task domains of the distributed multiagent scheduling problem. Unlike other HTN representations, C_TÆMS emphasizes modeling of the interrelationships between tasks more so than those between agents and their environment [Musliner *et al.*, 2006]. For the sake of exposition and formalism, we represent C_TÆMS instances using set theory, whereas the actual specification is a grammar. We represent a C_TÆMS instance as a tuple $\langle N, \mathscr{M}, \mathscr{T}, G, \mu, \phi, \omega \rangle$, where

- N is a set of agents;
- $\mathscr{M}$ is a set of methods;
- $\mathscr{T}$ is a set of tasks;
- $\mathscr{E}$ is a set of "Non-Local Effects";
- G is a special task known as the "Task Group;"
- μ is a function $\mu : \mathscr{M} \rightarrow A$;
- ϕ is a function $\phi : \mathscr{M} \cup \mathscr{T} \rightarrow \mathscr{T} \cup \{\emptyset\}$ that maps methods and tasks to their parent task; and
- ω is a quality accumulation function $\omega : \mathscr{M} \cup \mathscr{T} \rightarrow \mathbb{R}^+ \cup \{0\}$ that returns the quality of a method or task. For $T \in \mathscr{T}$, $\omega(T)$ is usually defined as a function of the associated qualities of all $X \in \phi^{-1}(T)$.

Each $M \in \mathscr{M}$ is itself a tuple, $\langle l, u, d \rangle$, where

- l is the earliest start time of the method;
- u is a deadline for the method; and
- d is the expected duration of the method.

Note that $l \leq u$, but $l + d$ might not necessarily be less than or equal to u. Each $T \in \mathscr{T}$ is defined as a similar tuple, except tasks (as a type of virtual method aggregator) do not

[1] We use the notation "$\mathbb{N}_0$" to denote the non-negative integers.

have explicit durations. Also, these bounds on execution time are inherited from parents. In other words, $l_{\phi(X)} \leq l_X$ and $u_{\phi(X)} \geq u_X$. It is assumed that ϕ creates parent-child relationships such that the resulting hierarchy is a tree rooted at G. The range of ϕ ensures that all methods are leaves.

C_TÆMS prescribes four primary types of quality accumulation function (QAF): SUM, MIN, MAX, and SYNCSUM (the qualities of the children are summed if and only if all of the children are scheduled to start execution at the same time).

$\mathscr{E}$ is a tuple containing sets of functions, called "Non-Local Effects" (NLEs) in C_TÆMS, that define temporal precedence constraints between methods and tasks: $\langle E, D, F, H \rangle$. Each function in each of the sets maps pairs of methods and tasks to a boolean; if the function maps to TRUE, then the associated NLE holds for that pair of methods/tasks. E contains a set of hard "enables" NLEs (if X enables Y, then Y cannot execute until X has accumulated positive quality). D is a set of "disables" NLEs, which is the opposite of an *enables* NLE. Finally, F and H are respectively "facilitates" and "hinders," both being soft NLEs that either increase or decrease the quality of targeted methods/tasks depending on the quality of the source method/task.

A "schedule" is a grammar within the C_TÆMS specification for defining the chosen start times of methods; it can be formalized as a function $s : \mathscr{M} \rightarrow \mathbb{N}_0 \cup \{\emptyset\}$. A start time of $\emptyset$ means that the method will not be executed. A *feasible* schedule obeys both mutex and precedence constraints (*i.e.* an agent may not execute more than one method at a time and all NLEs are obeyed). The objective is to create a feasible schedule that maximizes $\omega(G)$.

4 Mapping C_TÆMS to a DCOP

The basis of our approach is to map a given C_TÆMS representation of a distributed scheduling problem to an equivalent DCOP whose solution leads to an optimal schedule. The technical challenge lies in ensuring that the resulting DCOP's solution leads to an optimal schedule. The following sections formalize the approach.

4.1 Challenges

DCOPs are a promising solution technique, but there exist several challenges that must first be addressed:

Representation. There are different semantics in which the problem can be represented: one could have a variable for each method in the C_TÆMS instance that is assigned the value of its start time, or one could have a variable for each instance in time that will be assigned the method that will be executed at that time.

Determinism. C_TÆMS prescribes probability distributions over certain parameters, such as method duration, which is a problem since DCOPs are deterministic.

Aggregation Functions. DCOP solution algorithms do not allow for arbitrary aggregation functions (σ). For example, ADOPT requires associativity, commutativity, and monotonicity of the aggregation function in order to ensure optimality [Modi *et al.*, 2005].

n-ary Constraints. Although DCOPs allow for constraints of arbitrary arity, discovery of efficient algorithms for solving problems containing greater-than-binary constraints is still an open problem [Modi *et al.*, 2005].

Finite Domains. Regardless of how the problem is represented, time must be discretized since both the number of variables and the cardinality of variables' domains must be finite.

4.2 Method

The representational challenge is met by creating one DCOP variable for each method; the variables will be assigned the start times of their associated methods. The variables' domains will therefore contain the feasible start times of the associated method. Time is discretized into quanta for the domains. Likewise, the probability distributions in C_TÆMS are discretized using expected values.

For each agent in the C_TÆMS instance, $n \in N$, create an associated DCOP agent, $a \in A$. For each method, $M \in \mathscr{M}$, create an associated variable $v_M \in V$. Therefore, the α function of the DCOP can be defined as an analog of the μ function of C_TÆMS. The domains of the variables will contain all possible start times of the method (including an option for the method to forgo execution). In order to encode the mutex constraints,

$$(\forall n \in N : (\forall (M_i, M_j) \in \mu^{-1}(n) \times \mu^{-1}(n) : f_{ij}(x \in D_i, y \in D_j) \mapsto \infty \text{ when } (x < y < x + d_{M_i}) \vee (y < x < y + d_{M_j}))).$$

In other words, for all agents $n \in N$ find all pairs of methods M_i and M_j that share agent n and create a hard DCOP constraint (*i.e.* of infinite cost) for all pairs of equal domain values for the associated variables. This will ensure that an agent may not execute multiple methods at once. NLEs (*i.e.* precedence constraints) are encoded similarly. For example, the *enables* NLEs are encoded as follows. Each $e \in E$ is a function $e : (\mathscr{M} \cup \mathscr{T}) \times (\mathscr{M} \cup \mathscr{T}) \rightarrow \mathbb{B}$ mapping pairs of methods and tasks to a boolean. Let $\varphi : \mathscr{T} \times (\mathscr{M} \cup \mathscr{T}) \rightarrow \mathbb{B}$ be a function such that $\varphi(X, Y) \mapsto \text{TRUE} \iff \phi(X) \mapsto Y$, and let φ^+ be the transitive closure of φ. $\varphi^+(X, Y)$ implies that X is in the subtree rooted at Y. Therefore,

$$e(X, Y) \implies (\forall X', Y' \mid \varphi^+(X', X) \wedge X' \text{ is a method} \wedge \varphi^+(Y', Y) \wedge Y' \text{ is a method} : X' \text{ must have finished executing before } Y' \text{ can start}).$$

In other words, $e(X, Y)$ means that all methods in the subtree rooted at X must have finished executing *before* any of the methods in the subtree rooted at Y may be executed[2]. For each $e \in E$, if the transitive closure $e^+(X, Y)$ maps to TRUE, X is said to precede Y in an "NLE chain." Then,

$$\begin{array}{rll} (\forall e \in E : (\forall X, Y & \mid & e(X, Y) : \\ (\forall M_i, M_j & \mid & \varphi^+(M_i, X) \wedge M_i \in \mathscr{M} \wedge \\ & & \varphi^+(M_j, Y) \wedge M_j \in \mathscr{M} \\ & : & f_{ij}(x \in D_i, y \in D_j) \mapsto \infty \\ & & \text{when } y < x + d_{M_i}))). \end{array}$$

[2]Note that this definition is slightly more restrictive than that of [Boddy *et al.*, 2006], in which Y cannot execute until X has accumulated positive quality: $\omega(X) > 0$.

Finally, create one soft $|\mathscr{M}|$-ary constraint between all of the variables that aggregates the QAFs in the HTN.

4.3 DCOP-Solvable C_TÆMS

The fundamental shortcoming of the mapping proposed in the previous section is the use of n-ary constraints, which are extremely inefficient (and often not supported) by current solution techniques. DCOP-Solvable C_TÆMS (DSC_TÆMS) is our proposed a subset of C_TÆMS that allows for immediate application of current DCOP algorithms.

DSC_TÆMS consists of only *enables* and *disables* NLEs, and may only contain either: (1) all SUM and SYNCSUM QAFs; (2) all MAX QAFs; or (3) all MIN QAFs. In each case the mapping is exactly the same as that previously introduced, however, the |M|-ary soft constraint is decomposed into |M| unary constraints. Add one unary soft constraint for all methods' variables as follows: $(\forall M_i \in \mathscr{M} : f_i(\emptyset) \mapsto \omega(M_i))$.

If a method is *not* scheduled to execute, its unary constraint will have a cost equal to the quality that the method *would have* contributed to G had it been executed. This mapping will produce a DCOP with $|M|$ variables and worst-case $O(|M|^2)$ constraints. In case 1, when all of the QAFs are summation, use summation for DCOP aggregation function σ. Likewise, in case 2, use maximization for σ. Finally, in the case of minimization, one can create the dual of the minimization problem and apply the MAX aggregation function.

It is theoretically possible to extend DSC_TÆMS to allow for cases in addition to the three listed above, assuming n-ary constraints are feasible for small n. For example, one could encode a DCOP containing all four types of QAF by adding an n-ary soft constraint for each maximal subtree of the HTN that is rooted by either a MAX or MIN QAF, adding a unary soft constraint (as usual for DSC_TÆMS) for each method *not* a member of one such maximal subtree.

$$(\forall T_i \in \mathscr{T} \,|\, \omega(T_i) \text{ is MAX or MIN} \wedge$$
$$(\forall T_j \in \varphi^+(T_i, T_j) \,:\, \omega(T_j) \text{ is SUM or SYNCSUM}) :$$
$$\text{Create an } n\text{-ary soft constraint encoding the QAFs}$$
$$\text{in the subtree rooted at } T_i).$$

A detailed analysis of the correctness and complexity of our mapping is available in [Sultanik, 2006].

5 Efficiency Optimizations

Nothing thus far in the mapping precludes domains from being infinite, which is one of the challenges (and limitations) of using DCOP. From a practical standpoint, this is also a problem because most DCOP solution techniques have exponential computational complexity with respect to both the domain size and number of variables. Since the number of variables is generally inflexible, not only do we need to make the domains finite but we ideally need to make them as small as possible while ensuring that the solution space of the DCOP still contains the optimal solution.

5.1 Naïve Domain Bounding

It is possible to create a finite (although not necessarily tight) upper bound on the start times of the methods. Let us consider a C_TÆMS instance in which all methods have an earliest start time of zero. In this case, assuming all of the methods will be executed, the optimal start time of a method cannot be greater than the sum of the expected durations of all of the other methods. In the general case of heterogeneous earliest start times, we can define an upper bound on the start time of a method M as the maximum finite earliest start time in $\mathscr{M}$ plus the duration of all other methods.

5.2 Bound Propagation

Although the nature of the distributed scheduling problem implies that a child's bounds are inherited from (and therefore cannot be looser than) its parent's, C_TÆMS neither requires nor enforces this. Bounds can be propagated down the tree from the root to improve upon the naïve bounding. A distributed method for implementing this procedure (requiring only local knowledge) is given in Algorithm 1. To calculate the bounds for the methods of agent a, the algorithm would be invoked as RECURSE-EXEC-BOUNDS$(a, G, C, 0, \infty)$.

Algorithm 1 RECURSE-EXEC-BOUNDS$(a, T, C, \ell, \upsilon)$

Require: a is the agent from whose perspective we will create the bounds, T is the task rooting the tree whose bounds we will create, C is a C_TÆMS problem instance, ℓ is a lower bound on the bounds of T, and υ is an upper bound on the bounds of T.
Ensure: $\beta_a : \mathscr{T} \rightarrow (\mathbb{N}_0 \cup \{\infty\}) \times (\mathbb{N}_0 \cup \{\infty\})$ is a function mapping all tasks in the subtree rooted at T to lower and upper bounds on their start times. The subscript a exists to emphasize the point that each agent has its own β function; the mapping of each β function is contingent upon the extent of knowledge the agent has been given within the problem instance.

```
1: l ← ℓ
2: u ← υ
3: if VISIBLE-TO?(C, T, a) then
4:    e ←EARLIEST-START-TIME(C, T)
5:    d ←DEADLINE(C, T)
6:    if e ≠ ∞ ∧ e > l then
7:       l ← e
8:    end if
9:    if d ≠ −∞ ∧ d < u then
10:      u ← d
11:   end if
12: end if
13: β_a(T) ↦ (l, u)
14: for all S ∈SUBTASKS(C, T) do
15:   if IS-METHOD?(C, S) then
16:      This means S is a method (i.e. a special type of task)
17:      if VISIBLE-TO?(C, S, a) then
18:         l_m ← l
19:         u_m ← u
20:         e ←EARLIEST-START-TIME(C, S)
21:         d ←DEADLINE(C, S)
22:         a ←EXPECTED-DURATION(C, S)
23:         if e ≠ ∞ ∧ e > l_m then
24:            l_m ← e
25:         end if
26:         if d ≠ −∞ ∧ d − a < u_m then
27:            u_m ← d
28:         end if
29:         β_a(S) ↦ (l_m, u_m)
30:      end if
31:   else
32:      This means S is a regular task.
33:      RECURSE-EXEC-BOUNDS(a, S, C, l, u)
34:   end if
35: end for
```

5.3 Constraint Propagation

A binary constraint, f_{ij}, is *arc consistent* if $(\forall d_i \in D_i : (\exists d_j \in D_j : f_{ij}(d_i, d_j) \neq \infty))$. A DCOP

is said to be arc consistent if all $f \in F$ are arc consistent [Barták, 2001]. We use forward constraint propagation down the NLE chains to prune the domains, ensuring arc consistency of the DCOP. A distributed method for constraint propagation is given in Algorithm 2. Note that this algorithm makes use of a function BROADCAST-BOUNDS(C, β_a, a), which has the following postcondition: agent a's bounds will be broadcast to all other agents that share an NLE with the method/task associated with the given bound. Algorithm 2 works by having agents continually broadcast their current start time bounds for their methods; if they receive a bound that violates arc consistency, they increase the lower bound on their method's start time until the constraint is arc consistent and re-broadcast the new bounds. Since the lower bounds monotonically increase and are bounded above, the algorithm must terminate. An analysis of the messaging overhead of this algorithm is presented in §6.

Algorithm 2 DIST-CONSTRAINT-PROP(C, β_a, a, Q)

Require: C is a C_TÆMS problem instance and β_a is the associated bound function for the agent, a, that is running this instance of the algorithm. Q is a queue that is continuously updated with incoming broadcasts.

1: BROADCAST-BOUNDS(C, β, a)
2: **while** $Q \neq \emptyset$ **do**
3: $\langle f, r, s, t, (l, u)\rangle \leftarrow$POP$(Q)$
4: **if** $r =$SOURCE **then**
5: $\beta_a(s) \mapsto (l, u)$
6: $l_s \leftarrow l$
7: $u_s \leftarrow u$
8: $(l_t, u_t) \leftarrow \beta_a(t)$
9: **else**
10: $\beta_a(t) \mapsto (l, u)$
11: $l_t \leftarrow l$
12: $u_t \leftarrow u$
13: $(l_s, u_s) \leftarrow \beta_a(s)$
14: **end if**
15: $\delta \leftarrow l_s +$EXPECTED-DURATION$(C, s) - l_t$
16: **if** $\delta > 0 \wedge l_t + \delta \leq u_t$ **then**
17: $\beta_a(t) \mapsto (l_t + \delta, u_t)$
18: $\beta_a(t) \mapsto (l_t + \delta, u_t)$
19: BROADCAST-BOUNDS(C, β_a, a)
20: **end if**
21: **end while**

6 Results

Using the C_TÆMS scenario generator created for DARPA's COORDINATORs project, we randomly-generated a set of 100 C_TÆMS instances, each with four agents, three-to-four windows[3], one-to-three agents per window, and one-to-three NLE chains. The scenario generator does not ensure the existence of a feasible schedule for its resulting C_TÆMS instances. Even with the small simulation parameters and using all of our domain reduction techniques, the average state space size of these problems was astronomical: on the order of 10^{77}. Therefore, some of the problems inevitably require inordinate amounts of computation time. There seems to be a phase transition in the problems, such that some are soluble within the first couple thousand cycles of the DCOP algorithm, while the rest keep searching for an optimal solution into the millions of cycles. In terms of computation time, this equates to several orders of magnitude difference: seconds versus days. This necessitated a threshold—that we've set to 10,000 DCOP cycles—above which a C_TÆMS instance is simply declared "insoluble." We have not found a single C_TÆMS instance that has been soluble in greater than 5000 cycles (given a reasonable amount of computation time).

We used the DCOP algorithm Adopt [Modi *et al.*, 2005] to solve the resulting DCOPs. Of the 100 random problems, none were soluble by the naïve domain bounding approach. Applying bound propagation (Algorithm 1) to the naïve bounding resulted in 2% of the problems becoming soluble. Applying all of our methods resulted in 26% solubility. Using an upper one-sided paired t-test, we can say with 95% certainty that Algorithm 2 made an average domain size reduction of 7.62% over the domains produced from Algorithm 1. If we look at this reduction in terms of state space size, however, it becomes much more significant: an average 94% decrease in state space size. Table 1 presents the state space reduction efficiency of our constraint propagation technique in terms of problem solubility. Since constraint propagation was fairly consistent in the percentage of state space reduced between those problems that were soluble and those that were insoluble, this suggests that the 74% of the problems that remained insoluble were due to the large state space size inherent in their structure. For example, the insoluble problems' state spaces were, on average, *one million times* as large as those that were soluble.

We also conducted tests over C_TÆMS problems of differing complexity (by varying the number of windows and NLE chains). The number of windows is correlated to the number of variables in the resulting DCOP, while the number of NLEs is correlated to the number of constraints. These data are presented in Table 2. Notice that problems bounded naïvely were never soluble. Over the most complex problems with 6 windows and 3 NLEs chains, Algorithm 2 required an average of 144.94 messages (with a standard deviation of 16.13). This was negligible in comparison to the number of messages required to arrive at an optimal solution.

7 Discussion

We have presented a mapping from the C_TÆMS modeling language to an equivalent DCOP. We have shown that the resulting DCOP of a subset of this language, DSC_TÆMS, is soluble using existing techniques, and whose solution is guaranteed to lead to an optimal schedule. We have empirically validated our approach, using various existing techniques from the constraint processing literature, indicating that these problems are in fact soluble using our method.

We are optimistic in extending our mapping to subsume a larger subset of C_TÆMS. There are also various heuristic techniques in the literature, such as variable ordering [Chechetka and Sycara, 2005], that can be applied to the mapping while retaining formal guarantees on solution quality. If the resulting schedule need not be optimal (*i.e.* feasibility is sufficient), approximation techniques for DCOPs exist.

With the groundwork laid in solving distributed multiagent coordination problems with distributed constraint optimization, we have many extensions in which to investigate.

[3] "Windows" are tasks whose parent is the task group (*i.e.*, they are tasks at the second layer from the root in the HTN).

Solubility	Avg. Domain Size Reduction	Avg. Final Domain Size	State Space Reduction	Final State Space Size
Solved	8.02%	35.29	97%	2.34×10^{71}
Unsolved	7.61%	35.24	94%	1.47×10^{77}

Table 1: Efficiency of Algorithm 2 at reducing average domain size and state space size, in terms of solubility.

			% Soluble		Avg. # Cycles		Avg. # Messages	
	# Windows	# NLE Chains	Naïve	CP	Naïve	CP	Naïve	CP
	3	0	0	40.00	-	143.87	-	2569.25
	3	2	0	36.67	-	143.40	-	3114.14
	3	4	0	46.67	-	220.73	-	3854.2
	4	0	0	33.33	-	83.89	-	1758.5
	4	2	0	11.76	-	66.18	-	2783
	4	4	0	33.33	-	108.72	-	3887
	5	0	0	60.00	-	122.1	-	1364.5
	5	2	0	60.00	-	242.4	-	2634
	5	4	0	50.00	-	248.8	-	5748.67
*	6	3	0	41.67	-	196.42	-	4025.08

Table 2: Solubility statistics for different complexities of C_TÆMS instances. All simulations were conducted with four agents. None of the problems bounded using the naïve method were soluble. * This is the default configuration for the C_TÆMS scenario generator.

References

[Barták, 2001] Roman Barták. Theory and practice of constraint propagation. In J. Figwer, editor, *Proceedings of the 3rd Workshop on Constraint Programming in Decision Control*, Poland, June 2001.

[Bessière *et al.*, 1999] Christian Bessière, Eugene C. Freuder, and Jean-Charles Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.

[Boddy *et al.*, 2006] Mark S. Boddy, Bryan C. Horling, John Phelps, Robert P. Goldman, Regis Vincent, A. Chris Long, Robert C. Kohout, and Rajiv T. Maheswaran. C-TAEMS language specification v. 2.02, 2006.

[Chechetka and Sycara, 2005] Anton Chechetka and Katia Sycara. A decentralized variable ordering method for distributed constraint optimization. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1307–1308, New York, NY, USA, 2005. ACM Press.

[Davin and Modi, 2005] John Davin and Pragnesh Jay Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of Forth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1057–1063, July 2005.

[Decker, 1996] Keith Decker. TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.

[Hamadi, 2002] Youssef Hamadi. Optimal distributed arc-consistency. *Constraints*, 7:367–385, 2002.

[Mackworth, 1977] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[Mailler and Lesser, 2004] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, Washington, DC, USA, 2004. IEEE Computer Society.

[Modi *et al.*, 2005] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 2005.

[Musliner *et al.*, 2006] David J. Musliner, Edmund H. Durfee, Jianhui Wu, Dmitri A. Dolgov, Robert P. Goldman, and Mark S. Boddy. Coordinated plan management using multiagent MDPs. In *Proceedings of the AAAI Spring Symposium on Distributed Plan and Schedule Management*. AAAI Press, March 2006.

[Petcu and Faltings, 2005] Adrian Petcu and Boi V. Faltings. A scalable method for multiagent constraint optimization. In *Proc of International Joint Conference on Artificial Intelligence*, 2005.

[Phelps and Rye, 2006] John Phelps and Jeff Rye. GPGP—a domain-independent implementation. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*. AAAI Press, March 2006.

[Silaghi *et al.*, 2001] Marius-Calin Silaghi, Djamila Sam-Haroud, and Boi V. Faltings. Asynchronous consistency maintenance. In *Intelligent Agent Technologies*, 2001.

[Smith *et al.*, 2006] Stephen Smith, Anthony T. Gallagher, Terry Lyle Zimmerman, Laura Barbulescu, and Zack Rubinstein. Multi-agent management of joint schedules. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*. AAAI Press, March 2006.

[Sultanik, 2006] Evan A. Sultanik. Enabling multi-agent coordination in stochastic peer-to-peer environments. Master's thesis, Drexel University, April 2006.