Criticality Metrics for Distributed Plan and Schedule Management

Rajiv T. Maheswaran and **Pedro Szekely**
Information Sciences Institute, University of Southern California
4676 Admiralty Way #1001, Marina Del Rey, 90292
{maheswar, pszekely} @ isi.edu

Abstract

We address the problem of coordinating the plans and schedules for a team of agents in an uncertain and dynamic environment. Bounded rationality, bounded communication, subjectivity and distribution make it extremely challenging to find effective strategies. The Criticality-Sensitive Coordination (CSC) system uses multiple policy modification managers making predictable policy changes based on criticality metrics derived from simple computations on a graph-representation of the reward function with nearest neighbor communication. In the context of the DARPA Coordinators program, under an extensive and independent evaluation, the CSC system significantly outperformed competing approaches based on Temporal Networks and Markov Decision Processes.

Introduction

Coordinating the execution of activities for a multi-agent team in dynamic and uncertain environments is critical in domains such as large-scale disaster rescue, joint military operations and project management, among others. There are many characteristics of these domains that make effective coordination extremely challenging. A team begins with an initial plan of activities with *uncertain* duration and outcome. As uncertainties are resolved during execution, agents may need to modify their plans, e.g., reschedule or perform alternate activities. As the *scale* increases, it becomes infeasible to calculate and store an optimal set of policies that prescribe appropriate plan changes for all contingencies. This introduces one form of *dynamism*, where agents must modify their policies over time. A second form of *dynamism* occurs when agents' models of the world or the team reward function changes during execution. An agent might discover that an activity will take much longer or is more likely to fail than originally anticipated. The team might find unexpected and significant new tasks to perform in the middle of execution. The initial plan then becomes invalid and agents must adapt their plans and schedules during execution to get a high quality solution.

Other domain characteristics include *distribution* and *subjectivity*. As execution evolves, each agent can observe the outcomes of only some activities. As agents modify their policies, each agent is only aware of the changes to its policy. Agents may only have partial knowledge of the team reward function. Thus, agents must share information to choose the appropriate policy modifications.

Any coordination technology must work within the pace of execution, and the *bounded communication* and *bounded rationality* limitations of the infrastructure and equipment available. When coupled with dynamism, centralization is not a viable strategy, especially as the problem scales. Also, having a single point of failure is risky and undesirable. As the number of agents increases, it becomes infeasible to share all information among all agents without incurring delays that exceed the times at which decisions must be made. The shared information must be processed and subsequent policy modifications must be analyzed using the reasoning cycles available between decision points. Thus, time-critical domains restrict the number of contingencies that can be considered. The impact of these difficulties can be greatly magnified by *nonlinearities* that arise from conjunction, disjunction or synchronization in the team reward function. Researchers have been tackling problems with many of the characteristics mentioned above, however, conceiving techniques that handle *all* aspects of these settings is a nascent area of research.

The DARPA Coordinators program has required solution concepts that address all the aforementioned characteristics through its goal of creating "distributed intelligent software systems that will help fielded units adapt their mission plans as the situation around them changes and impacts their plans." [1] Three teams of researchers pursued three different solution concepts. One approach was based on Simple Temporal Networks (STNs). Another was based on Markov Decision Processes (MDPs). After two years of development and under independent and extensive testing, both were significantly outperformed by a novel approach: the Criticality-Sensitive Coordination (CSC) system used a set of simple policy modification managers that were tightly integrated with criticality metrics that propagated over a graph representation of the reward function.

In this paper, we describe the Coordinators problem, develop a formalism for characterizing the three vastly different approaches, and describe the metrics and managers of

[1] http://www.darpa.mil/ipto/Programs/coordinators/

CSC. We also present the experimental results from the independent DARPA evaluation of the three approaches, and discuss the performance of the three systems, which yields the ideas of predictability and the value of global, accurate and timely information. We hope that these insights help develop better solution concepts for these important domains.

The CTAEMS Model

The Coordinators problem is formally modeled using a version of the TAEMS (Task Analysis, Environment Modeling, and Simulation) framework (Lesser et al. 2004) called CTAEMS (Boddy et al. 2007):

Methods: Each agent $i \in I$ has a distinct set of activities M_i, known as *methods*, that they can perform ($M_{i_1} \cap M_{i_2} = \emptyset$, if $i_1 \neq i_2$), but they can execute only one at a time. If a method is started at time s_m, it will occupy the agent's time for a duration $\delta_m \in \Delta_m$ with probability $p(\delta_m)$ and yield a quality $q_m \in Q_m$ with probability $p(q_m)$ when it completes at end time $e_m = s_m + \delta_m$. A method must be attempted in order to yield quality and each method can only be attempted once. The uncertainty regarding the duration and quality is resolved only upon completion. Method m will yield zero quality if it started before its *release* time r_m or completes after its *deadline* d_m. A method may be aborted during execution. This yields zero quality ($q_m = 0$) for the method but frees the agent to perform other methods.

Tasks: The team reward function is represented using a task decomposition graph. The root task represents mission quality. Tasks nodes can be decomposed into many levels of subtasks. Let $C(n)$ denotes the children of node n. Leaf nodes ($C(n) = \emptyset$) are methods. A node can be a child of at most one node. Let $q_n(t)$ denote the quality of any node n at time t. Each task node n is associated with a *node operator* $\odot_n$, such as max, min, or sum which takes the qualities of its children as input and yields the quality of the node as the output: $q_n(t) = \odot_n(\{q_{\tilde{n}}(t)\}_{\tilde{n} \in C(n)})$. Other operators include $syncsum$, for which the quality is the sum of the qualities of the children that began execution at the earliest time: $q_n(t) = \sum_{\hat{n} \in \hat{C}(n)} q_{\hat{n}}(t)$ where the synchronized children are $\hat{C}(n) = \{\hat{n} \in C(n) : s_{\hat{n}} = \min_{\tilde{n} \in C(n)} s_{\tilde{n}}\}$ and $s_n = \min_{\tilde{n} \in C(n)} s_{\tilde{n}}$ is the start time of node n. The $sumand$ operator adds the sum of the childrens' qualities only if they are all positive: $q_n(t) = \sum_{\tilde{n} \in C(n)} q_{\tilde{n}}(t)$ if $q_{\tilde{n}}(t) > 0 \; \forall \tilde{n} \in C(n)$ and $q_n(t) = 0$, otherwise.

Links: The task decomposition graph can also include directional links, called *non-local effects (NLEs)* in CTAEMS. Each link l is associated with a *link operator* $\overrightarrow{\odot}^l_{n_s,n_t}(\cdot)$ where n_s and n_t are the source and target nodes of the link. An *enables* link requires that the quality of the source node be positive at the start time of a target method for the target method to obtain positive quality. A *disables* link precludes a target method from achieving positive quality if it started after the source node achieved positive quality. If the target node of a link is a task, it can be interpreted as multiple links from the source to all descendant methods of the target. Additional links include *facilitates* which modifies the quality and duration distributions of target methods to make methods take less time and yield more quality in proportion to the quality achieved by the source. The *hinders* link has the opposite effect.

Reward Function: Let $I_{\{\cdot\}}$ be an indicator function, $L^T(m)$ be the links whose target node is m (or an ancestor of m) and $n_s(l)$ be the source node of link l. Then, the method quality for node m at time t is $q_m(t) =$

$$q_m I_{\{t \geq e_m\}} I_{\{s_m \geq r_m\}} I_{\{e_m \leq d_m\}} \Pi_{l \in L^T(m)} \overrightarrow{\odot}^l_{n_s(l),m}(\cdot).$$

The arguments of the link operators can vary based on their type. The method qualities propagate through task node operators to determine the root node quality. The objective of the team is to maximize $q_0(T)$, the root quality at terminal time T. While additional modifications can increase expressibility, the preceding constructs are sufficiently rich to create problems of great complexity.

Information: Agents are not given the complete task decomposition graph, known as the *objective view*, and thus have incomplete knowledge of the team reward function. Instead, they are given a subgraph, or *subjective view*, which has (1) methods they own, (2) ancestors of these methods and (3) nodes that are sources or targets of links with a node in (1) or (2). An agent only knows distributions, releases and deadlines of the methods in its subgraph.

Each agent is given an initial schedule consisting of a set of methods and start times, computed by a centralized solver. Only the agent owning a method will observe the actual start time, end time and quality obtained for that method. While agents are free to share this information, unanticipated delay and failure (getting zero quality) forces agents to modify the policies generated from this schedule. Exogenous events such as distribution changes, release/deadline changes and the arrival of new tasks can alter the reward function during execution and force agents to modify their plans. So, even an optimal policy for the original problem can be invalidated at run-time. Thus, this problem model incorporates all the challenging characteristics discussed earlier.

CTAEMS Examples

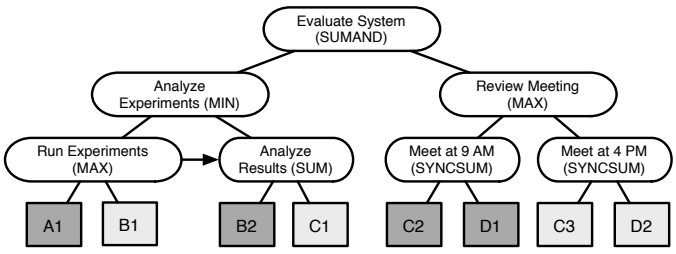

Figure 1: An Example Scenario

Consider the scenario in Figure 1. The graph shows task (oval) and method (rectangle) nodes for a plan where four agents (A, B, C, D) have an objective of evaluating their system. This requires both analyzing experiments and having a review meeting. To analyze experiments, the team needs to run experiments which enables performing an analysis. The initial plan is $A1, B2, C2, D1$. However, $A1$ fails due a hardware error (outcome uncertainty). Realizing this,

Agent B starts the experiments on his hardware ($B1$). Unfortunately, $B1$ takes longer than anticipated (duration uncertainty) and Agent B cannot complete the analysis ($B2$) before the deadline (temporal constraints). Agent C can do the analysis more quickly ($C1$), but it precludes having the review meeting at 9 AM. Agents C and D must switch to the 4 PM meeting time ($C3, D2$). This example is a microcosm of some of the difficulties that arise in coordinating in an uncertain temporally constrained environment. If any of the method changes did not occur, they would have failed.

While the example might seem trivial, the difficulties are amplified as the scale grows. A reward graph with approximately 30 agents and 1000 nodes is shown in Figure 2. The Coordinators program evaluation involved scenarios of up to 100 agents and over 13000 nodes.

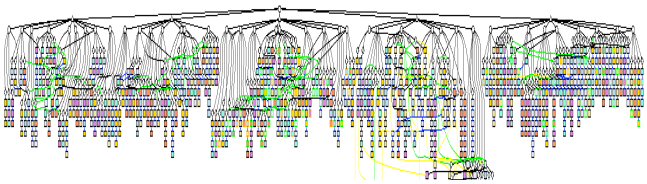

Figure 2: Reward Graph for a 30-agent 1000-node Scenario

Solution Concepts

The three approaches to solving the Coordinators problem were vastly different. Here, we propose a formalism for the general problem which can capture all these approaches. This will help compare and evaluate their performance. To find an optimal solution to the general multi-agent coordination problem, one must solve:

$$\pi^* = \arg\max_{\pi \in \Pi} V(\pi, \eta^*), \;\; \Pi = \{\pi\}, \pi_i : X \rightarrow A_i$$

where $\pi = [\pi_1 \; \pi_2 \cdots]$ is the joint policy, V is the evaluation function, η^* is the full information required by the evaluation function, and Π is the space of joint policies composed of individual policies $\{\pi_i\}$, where each is a mapping from the complete state space X to an individual action space A_i.

For the Coordinators problem, the evaluation function V is the expected root quality at the terminal time and the full information η^* is the complete reward graph and distributions, releases and deadlines for all methods. The actions available to the i-th agent, A_i are to start a method in M_i, abort an executing method, or do nothing. If one could (1) solve this problem offline, given that evaluating V for a single policy requires analyzing all sample paths of the system, (2) investigate the entire space of policies, (3) store the entire joint-state-space-to-action mapping in a single agent, and (4) communicate the joint state in a timely manner, the results would still be suboptimal when η^* changed during execution. So, even given a good initial policy, which is difficult to obtain, agents will have to change their policy dynamically during execution. Another reason for dynamic policy modification is that bounded rationality prevents agents from investigating a large policy space. Thus, it is better to develop a policy incrementally as the uncertainty resolves itself during execution:

$$\pi_i^t = \arg\max_{\pi_i^t \in \widehat{\Pi}_i^t} \widehat{V}(\pi_i^t, \eta_i^t), \;\; \widehat{\Pi}_i^t \subset \{\pi_i^t\}, \pi_i^t : Z(\eta_i^t) \rightarrow A_i$$

where the i subscript indicates the parameter will vary by agent and the t superscript indicates parameters can change over time. For any $z \in Z(\eta_i^t)$ there is a mapping $\theta(z) \subset X$ which indicates the set of joint states that leads to that circumstance. These must be distinct: $\theta(z_1) \cap \theta(z_2) = \emptyset$. The circumstance set is a function (or determiner) of the information η_i^t that can (or needs to) be delivered at the time when actions need to be taken. If one wants to use the joint state space as the circumstance set ($Z(\eta_i^t) = X$), then one must be able to construct a system that can deliver that information. A *schedule* is a policy where the circumstance set is time ($z \in Z(\eta_i^t) \Rightarrow z = t$). The choice of Z determines the *policy representation* of the solution concept. Given the policy representation, $\widehat{\Pi}_i^t$ denotes the subset of policy space that will be investigated during a policy modification phase at time t. Policies in this space will be evaluated by $\widehat{V}$ which also is a function (or determiner) of the information that is propagated. To have a dynamic policy, there must be an action $a \in A_i$ that is "modify policy" along with circumstances that trigger it. The choices of $Z, \eta_i^t, \widehat{\Pi}_i$, and $\widehat{V}$ characterize a solution concept.

Other Approaches

Two of the approaches that were investigated in the Coordinators program extended distinct and prevalent schools of thought in planning and scheduling: Markov Decision Processes (MDPs) and Simple Temporal Networks (STNs).

The MDP-based approach (Musliner et al. 2006) addressed the infeasibility of reasoning over the joint state space by setting the circumstance set to a subset of local state space that is reachable from the current local state, $Z(\eta_i^t) = \widehat{X}_i^t \subset X_i$. The process of updating $\widehat{X}_i^t$ is referred to as "unrolling". During a policy modification phase, all policies that could be formed with the currently unrolled state space are investigated. The best actions for these states was determined by a value function, $\widehat{V}^{MDP}$, based on the agent's subjective view of the reward function. Values for the frontier of the unrolled state space are determined through a fast greedy search. Negotiated *commitments*, $\eta_i^{t,MDP}$, bias the value function to induce desired coordinated behavior (Witwicki and Durfee 2007).

The STN-based approach (Smith et al. 2007) addressed temporal uncertainty by using a time interval (instead of a point) as the circumstance that denotes feasible start times for a method to be executed: $Z(\eta_i^t) = \{(\underline{s}_m, \bar{s}_m)\} \; \forall m \in M_i^S \subset M_i$. As execution evolves, the system used STN *constraint propagation*, $\eta_i^{t,STN}$, to update the start intervals of methods in M_i^S. A policy modification phase was triggered if execution was forced outside the given set of intervals. The policy space considered included changes to the time intervals or the method set to be executed. The evaluation function, $\widehat{V}^{STN}$, was the root quality of the subjective reward, computed using the expected outcomes of

methods. Agents engaged in speculative joint policy modifications that were profitable according to $\widehat{V}^{STN}$.

The CSC System

The policy representation of CSC is similar to that of the STN-based system. It is a method set to be executed where each method has a start time interval and a *priority*: $Z(\eta_i^t) = \{(\underline{s}_m, \bar{s}_m, \rho_m)\}\ \forall m \in M_i^S \subset M_i$. If a method m_1 with priority ρ_{m_1} is executing at time $\underline{s}_{m_2}$ and $\rho_{m_2} > \rho_{m_1}$, we will abort m_1 and start executing m_2.

Each agent is given an *initial schedule*, $\{t_m\}_{m \in M_i^S}$, computed by a centralized solver before execution. This is used to create the initial policy, i.e., the executable method set and start windows, in both the CSC and STN-based systems. The MDP-based approach uses the initial schedule to create their initial unrolled state spaces, $\{\widehat{X}_i^0\}$. While the STN-based system updates the start windows and constraints throughout execution, CSC does not. Instead, we use a different information sharing mechanism based on the graph representation of the reward function. The mechanism quickly propagates simple metrics that are triggers for actions and evaluators for policy modification.

We assign an agent to each node n in the graph with the responsibility to update and disseminate all metrics associated with that node: $\mu_n = \{\mu_n^\tau\}_\tau$. Currently, we assign each node randomly to an agent whose subjective view contains it. Each metric μ_n^τ can either be local, i.e., determined by that node, or a function of metrics that exist on neighboring nodes, $B(n)$, i.e., nodes connected to n via a parent-child relationship or a directional link (e.g, enables, etc.). Whenever any metric is updated, its new value must be propagated to its neighbors. Each metric type τ has an operator $\otimes^\tau$ that defines how it is updated:

$$\mu_n^\tau = \otimes^\tau(\{\mu_{\hat{n}}\}_{\hat{n} \in B(n)})$$

We want these operations to be simple and fast so information can propagate through the graph quickly.

Local metric updates can occur when agents act or make observations that affect the method nodes they own (e.g., start times, end times, quality obtained). These updates are propagated to neighboring nodes which can cause metric updates in those nodes and ultimately result in a cascade of information flow through the graph. Appropriately chosen metrics can give agents more timely and accurate assessments of global state and the reward function from which to decide when and how to make policy modifications.

The information available to an agent is a collection of metrics on the nodes in its subjective view: $\eta_i^t = \{\mu_n(t)\}_{n \in G_i}$ where G_i is the subgraph known to the i-th agent and $\mu_n(t)$ represents the values of all metrics at node n at time t. These metrics trigger policy changes, $Z(\eta_i^t)$ and influence the evaluation functions $\widehat{V}(\pi_i^t, \eta_i^t)$. Policy modifications are performed by set of *managers*, each responsible for a particular type of policy modification. Thus, the policy modification space $\widehat{\Pi}_i^t$ will differ based on the manager that is triggered.

CSC Metrics

A simple metric calculated via the CSC information sharing mechanism is *quality accumulated* at a node at the current time t: $\mu_n^q(t) = q_n(t)$. A similar simple metric is *quality status* of a node, which indicates if it is possible for a node to get more quality: $\mu_n^{qs}(t) = I_{\{p[q_n(T) > q_n(t)] > 0\}}$. From these simple metrics, we can compute more advanced metrics.

Backbone Value (μ^{bb+}): The backbone value estimates the probability that the root task will fail, if a given node fails ($p[q_0(T) = 0 | q_n(T) = 0]$). If node n has achieved positive quality, $\mu_n^{bb+} = 0$, since it cannot fail. If it cannot improve ($\mu_n^{qs}(t) = 0$), $\mu_n^{bb+} = 0$, since it cannot alleviate the failure. The failure of a node can affect the root task through its parent and nodes it enables. The price of failure via enablement is the probability that failure of the enabled node $e(n)$ causes root failure:$p[q_0(T) = 0 | q_{e(n)}(T) = 0] = \mu_{e(n)}^{bb+}$. If the parent node $P(n)$ has a conjunctive node operator, e.g., $\odot_{P(n)} \in \{min, sumand\}$, then the price of failure is the probability that failure of the parent causes root failure: $\mu_{P(n)}^{bb+}$. If the parent has a disjunctive node operator, e.g., $\odot_{P(n)} \in \{max, sum\}$, then the price of failure of the parent can be split among the children who could alleviate that failure, i.e., those that can still accumulate quality. Thus, we have $\mu_n^{bb+}(t) =$

$$I_{\{q_n(t)=0\}}\ \mu_n^{qs}(t)\ \max\left\{f^{bb+}\left(\mu_{P(n)}^{bb+}, \odot_{P(n)}\right), \mu_{E(n)}^{bb+}\right\}$$

where $\mu_{E(n)}^{bb+}$ are the backbone values of all nodes enabled by n, $f^{bb+}\left(\mu_{P(n)}^{bb+}, \odot_{P(n)}\right) = \mu_{P(n)}^{bb+}$ for conjunctive $\odot_{P(n)}$, and $f^{bb+}\left(\mu_{P(n)}^{bb+}, \odot_{P(n)}\right) = \mu_{P(n)}^{bb+} / \sum_{\tilde{n} \in C(P(n))} \mu_{\tilde{n}}^{qs}(t)$ for disjunctive $\odot_{P(n)}$. The $\max$ operation chooses the single most damaging path to the root in the graph. An alternative metric uses the $\sum$ operation.

The μ^{bb+} metric can help identify critical methods to execute even though the reason for their criticality might not be visible in the agent's subjective view. If an agent could execute m_1 or m_2 and $q_{m_1} \gg q_{m_2}$ but $\mu_{m_1}^{bb+} = 0$ and $\mu_{m_2}^{bb+} = 1$, it would realize that even though m_1 was a much higher quality method, it does not affect root failure, while failure to execute m_2 assures root failure.

Backbreaker Value (μ^{bb-}): The backbreaker value estimates the probability that the root task will fail, if the node does not fail ($p[q_0(T) = 0 | q_n(T) > 0]$). The success of a node can damage the root task through disables links. If a node n has achieved positive quality, $\mu_n^{bb-} = 0$, since it cannot mitigate the damage. If it cannot improve ($\mu_n^{qs}(t) = 0$), $\mu_n^{bb-} = 0$, since it cannot cause failure through disablement. The price of success via disablement is the probability that failure of the disabled node $d(n)$ causes root failure: $\mu_n^{bb-} = p[q_0(T) = 0 | q_{d(n)}(T) = 0] = \mu_{d(n)}^{bb+}$, the backbone value of $d(n)$. This price is passed to a node's children as the success of the children cause success in the parent. If a node's parent has a disjunctive node operator, then $\mu_n^{bb-} = \mu_{P(n)}^{bb-}$ because $q_n(t) > 0 \rightarrow q_{P(n)}(t) > 0$. If a node's parent has a conjunctive node operator, then all children must succeed for the parent to succeed so $\mu_{P(n)}^{bb-}$ can be

split among n and its siblings. Thus, we have $\mu_n^{bb-}(t) =$

$$I_{\{q_n(t)=0\}}\ \mu_n^{qs}(t)\ \max\left\{f^{bb-}\left(\mu_{P(n)}^{bb-}, \odot_{P(n)}\right), \mu_{D(n)}^{bb+}\right\}$$

where $\mu_{D(n)}^{bb+}$ are the backbone values of all nodes disabled by n and f^{bb-} is the opposite of f^{bb+} with respect to conjunctive and disjunctive operators.

The μ^{bb-} metric can help identify methods to avoid executing even though the reason might not be visible in the agent's subjective view. If an agent could execute m_1 or m_2 and $q_{m_1} \gg q_{m_2}$ but $\mu_{m_1}^{bb-} = 1$ and $\mu_{m_2}^{bb+} = 0$, it would realize that even though m_1 was a much higher quality method, it will assure root failure and m_2 is a better method to execute. The backbone and backbreaker metrics can be adjusted to prevent the failure of non-root tasks by setting their backbone values to one.

Schedule Probability (μ^p): The schedule probability metric is the likelihood that the current node will obtain positive quality at $t = T$ under the current policies of all agents: $\mu_n^p(t) = p[q_n(T) > 0|\pi^t]$. While this is not entirely trivial to calculate, as it requires an appropriate approximation operator $\otimes^p$ at task nodes, we omit the details of how this is calculated. We mention it because it is needed for the following advanced metric.

Root Gain (μ^α): The root-gain metric estimates the change in the probability of success of the root task as a factor of the change in the probability of success of the node given the current policies: $\mu_n^\alpha = \partial\mu_0^p/\partial\mu_n^p$. Thus, if the schedule probability of only node n was increased by $d\mu_n^p$, the schedule probability of the root would increase by $d\mu_0^p = \mu_n^\alpha d\mu_n^p$. To determine how to calculate μ_n^α, we decompose μ_0^α as follows: $d\mu_0^p/d\mu_n^p =$

$$\frac{\partial\mu_0^p}{\partial\mu_{P(n)}^p}\frac{\partial\mu_{P(n)}^p}{\partial\mu_n^p} + \sum_{\tilde{n}\in E(n)}\frac{\partial\mu_0^p}{\partial\mu_{\tilde{n}}^p}\frac{\partial\mu_{\tilde{n}}^p}{\partial\mu_n^p} - \sum_{\tilde{n}\in D(n)}\frac{\partial\mu_0^p}{\partial\mu_{\tilde{n}}^p}\frac{\partial\mu_{\tilde{n}}^p}{\partial\mu_n^p}$$

$$= \mu_{P(n)}^\alpha g_n(P(n)) + \sum_{\tilde{n}\in E(n)}\mu_{\tilde{n}}^\alpha g_n(\tilde{n}) - \sum_{\tilde{n}\in D(n)}\mu_{P(n)}^\alpha g_n(\tilde{n})$$

where $P(n)$ is the parent of n, $E(n)$ are the nodes enabled by n, $D(n)$ are the nodes disabled by n, and $g_n(\tilde{n}) = \partial\mu_{\tilde{n}}^p/\partial\mu_n^p$.

This decomposes μ_n^α into parameters that can be determined from metrics in the node's neighborhood. The μ^α values are available as metrics and $g_n(\tilde{n})$ can be calculated from μ^p as follows: $g_n(\tilde{n}) = (1-\mu_{\tilde{n}}^p)/(1-\mu_n^p)$ if $\tilde{n} = P(n)$ and $\odot_{\tilde{n}}$ is disjunctive and $g_n(\tilde{n}) = \mu_{\tilde{n}}^p/\mu_n^p$ if $\tilde{n} = P(n)$ and $\odot_{\tilde{n}}$ is conjunctive, $\tilde{n} \in E(n)$, or $\tilde{n} \in D(n)$. [2] If an agent is choosing among otherwise equivalent methods, utilizes μ^α and executes the activity $m^* = \arg\max_m\{\mu_m^\alpha\ p[q_m(T) > 0]\}$, it will choose the activity that increases the probability of root success the most.

Target Quality (μ^{tq}) The target quality of a node is the quality value beyond which there will be no contribution to the quality at the root: $\mu_n^{tq} =$

$$\arg\max_{q_n}\{q_n : \exists\epsilon > 0 \text{ s.t. } p[q_0(T|q_n+\epsilon) - q_0(T|q_n)] > 0\}.$$

[2] discussion of singularity conditions are omitted due to space

To calculate this, we need to use some other simple metrics. Let μ_n^m be the maximum quality that node n could obtain without the aid of facilitation. This is important because it's the upper bound on the input for facilitation links. With μ_n^m, we can construct μ_n^f, the maximum quality that node n could obtain with the aid of facilitation and the current state of hinders links. Given these metrics, we have $\mu_n^{tq} =$

$$\min\left\{\mu_n^f, \max\{\mu_{P(n)}^{tq}, \mu_n^m I_{\{F(n)\neq\emptyset\}}, \epsilon I_{\{E(n)\neq\emptyset\}}\}\right\}.$$

This states that the target quality of n should be the smaller of the maximum achievable quality μ_n^f and the largest goal quality which could be your parent's target quality $\mu_{P(n)}^{tq}$, the upper bound for facilitation μ_n^m if n a facilitator ($F(n) \neq \emptyset$), or some small positive quality $\epsilon > 0$ if n is an enabler ($E(n) \neq \emptyset$). This metric is very useful because verifying $q_n(t) \geq \mu_n^{tq}(t)$ reveals whether a task or method is useful.

All the metrics discussed here are ultimately a function of method qualities $q_m(t)$ and time which are dynamic. Hence, all the metrics are dynamic. All the computations needed to calculate the metrics are extremely simple. Thus, the CSC nearest-neighbor information-propagation mechanism can track uncertainty and dynamism during execution and deliver globally-aggregated timely metrics to agents.

CSC Managers

We now discuss the three main managers that perform dynamic policy modifications in CSC.

Remover: Whenever the quality accumulated for a method is at least its target quality, $q_m(t) \geq \mu_m^{qs}$, the Remover is triggered and the method is removed from the method set M_i^S:

$$Z(\eta_i^t) = Z(\eta_i^{t-1}) \setminus (\underline{s}_m, \bar{s}_m, \rho_m)$$

If μ_m^{qs} changes while m is executing and the Remover is triggered, m will be aborted.

Opportunistic Inserter(OI): The OI is triggered whenever an agent is idle. It then investigates policy modifications of adding one method to be started immediately:

$$Z(\eta_i^t) \in \{Z(\eta_i^{t-1}) \cup (t, t', \rho_m)\}_{m\in M_i}$$

where $Z(\eta_i^{t-1})$ contains the previous method set, (t, t') is a start window that allows m to start immediately, and ρ_m is a priority that is lower than priorities of methods in the initial schedule. This ensures that when the next method is supposed to be started $t = \underline{s}_m^* = \min_{m\in M_i^S(Z(\eta_i^{t-1}))}\underline{s}_m$, the added method will be aborted if the method to be started at $\underline{s}_m^*$ has a higher priority.

The OI uses several criticality metrics to decide whether to insert a method in its current policy, and which method to insert. The agent will order useful methods based on their values of μ^{bb+}, μ^{bb-}, μ^α, and μ^p. This helps to add any methods that are critical to execute to avoid root failure, avoid methods that can cause root failure, favor the methods that will help the root succeed the most and favor the methods that will succeed at the highest rate, respectively.

Policy Manager (PM): The PM is triggered at every t and investigates policy modifications where a method in the method set M_i^S is replaced with a sibling method:

$$Z(\eta_i^t) = \{Z(\eta_i^{t-1}) \setminus (\underline{s}_m, \bar{s}_m, \rho_m) \cup (\underline{s}_{m'}, \bar{s}_{m'}, \rho_{m'})\}$$

where $m \in M_i^S$ and $m' \in C(P(m)) \cap M_i$. The set of siblings for each method is pruned by eliminating any sibling whose schedule probability μ^p or expected quality is not within some threshold of the original method's values. Methods are then pruned if they have link properties that are worse than the original method, e.g., they don't enable something that the original does or disable something useful that the original does not. From this final set, methods are ordered based on expected quality and then probability and the best is chosen. The PM is useful for managing uncertainty during execution, by replacing a method that is running long with a shorter method that is sure to succeed or swap methods quickly based on facilitation or hinders effects on method quality.

Additional managers exist to handle special node operators such as $syncsum$ or $exactlyone$ (the CTAEMS analogue to exclusive-or). These do not engage in significant policy modification but are there to ensure that the intended policy execution is not violated. We have also developed a manager to handle disables effects by changing start windows: the Backbreaker Shifter. However, it was not submitted for the independent evaluation. The idea of CSC managers is that policy modification should occur through a union of focused policy modification managers that have limited scope in which they can perform well.

Experiments

The three approaches were evaluated on scenarios constructed by an independent third party, assisted by two of the original designers of TAEMS. Additionally, each of the three teams provided a set of scenarios.The independently created data set consists of 14 groups of scenarios, each group containing 32 similar scenarios, for a total of 448 scenarios. The scenarios were produced using a scenario generator that randomly combined a variety of *templates* (Decker and Garvey 2007). Each template produces a CTAEMS structure that captures a specific coordination challenge. Examples include (1) *Synchronization*, where agents must maximize activities started at the same time, (2) *Dynamics*, where the task structure, NLEs and other constraints are changed often during execution, and (3) *NLE-Mixture*, where a group of tasks are connected by randomly chosen NLEs. Each scenario group contains a different mix of instances of these templates. Each group also differs on the settings of scenario generator parameters that controls failure rates, distributions of method outcomes, tightness of release/deadline constraints and activity overlap. The scenarios have between 25 and 100 agents, 848 and 13,662 nodes, and 104 and 827 NLEs. Proponents of each approach submitted 64 scenarios each, intended to highlight the strengths of their approach. All scenarios contain an initial schedule produced by a centralized scheduler

The simulations for the main systems were run on a cluster of Intel Core Duo machines on a Gigabit network. Each agent and the simulator ran on a different machine.[3] Message throughput was limited as all inter-agent messages were sent via the simulator where logs were kept. In the 100-agent simulations, agents were able to send between 25 and 65 messages per second. The simulator marked time with *pulses* that were one second long. Scenarios had horizons between 373 and 1,728 pulses. Method durations varied, but over 95% were between 3 and 14 pulses. In scenarios with many agents, several methods could complete on each pulse, i.e., dynamism was on the order of a second.

The score of a simulation is the quality achieved at the root task at the end of a simulation run. The scores on different scenarios varied significantly, from a few hundred to tens of thousands. To normalize each scenario, the best score obtained by any system or baseline was set to 100, and the scores of the other systems and baselines were scaled accordingly. To give some context to the performance of the systems, we ran CSC with only the Opportunistic Inserter with the added method chosen randomly.

Figure 3 shows the results. Rows represent systems and columns represent scenario groups. The numbers in parenthesis represent the number of agents in each scenario in a group.[4] The cells show the average normalized scores for all systems by scenario group. The first column is the average over all independent scenarios. The best score in each column was shaded dark grey. The light grey shading indicates that the score was significantly better than Random Insertion (using error bar analysis).

Discussion

The unmodified initial schedule performs abysmally, averaging 38% over the 448 independent tests illustrating the need for good distributed dynamic solutions. The first main result is that CSC significantly outperformed the other approaches, obtaining the highest score in 436 of 448 (97%) of the independently generated scenarios. The normalized scores over those scenarios were: CSC (98%), STN (80%), MDP (56%). We note that CSC is robust enough to score 97 on the independent tests *without* an initial schedule. The second main result is that the other approaches were mostly unable to outperform the relatively simple local algorithm of randomly adding methods to the initial schedule without perturbing it.

To understand these results, we return to our formalism for the approaches. First, when looking at the evaluation functions, $\widehat{V}^{MDP}$ assumed that a greedy forward search would provide a good estimate of the value of the boundary of the unrolled state space, and $\widehat{V}^{STN}$ used expected outcomes as input. Consider the cost of representing a method with uniform bimodal distributions for quality $q_m \in \{0, 10\}$ and duration $\delta_m \in \{1, 11\}$ with a single point distribution $q_m = 5, \delta_m = 6$.

[3]The baseline was run on a configuration where all agents ran on a single machine as Java threads. We ran a subset of scenarios in both configurations and verified that the average score differs by less than 0.1%. We didn't run the 100-agent set as each took over 4 hours.

[4]the Flexible Scheduling number is a mean

	AVERAGE	Contingent (33)	Dynamic (33)	Mixture (25)	Mixture (50)	Mixture (70)	NLE Circular (33)	NLE Mixture (50)	NLE Negative (33)	Real World-1 (33)	Real World-2 (33)	Synchronization (33)	Tight Deadlines (33)	Uncertainty (33)	Mixture (100)	CSC (21)	Flexible Sched. (22)	Distributed MDP (12)
Random Insertion	**77**	86	97	80	85	83	95	92	52	39	26	69	98	99	n/a	0	49	14
Distributed MDP	**56**	13	85	42	45	49	97	45	61	36	23	61	95	82	96	0	31	88
Flexible Scheduling	**80**	86	87	71	73	63	93	76	90	74	70	77	96	81	94	0	70	17
CSC	**98**	98	99	99	99	99	99	100	96	96	94	93	93	99	99	100	84	24

Figure 3: Experimental Results: Average Normalized Scores for Systems and Baseline Over Different Groups of Scenarios.

Both these assumption lose a lot of information because they attempt to approximate $q_0(T)$, the actual reward function, and make sacrifices to do so. CSC never attempts to approximate $q_0(T)$. Instead, it attempts to discover *features* of $q_0(T)$ such as the likelihood it is zero. On can sacrifice less for a more restricted evaluation function.

Second, it appears that the information η_i^t propagated in the other approaches do not trigger policy modifications very often. It takes leaving the unrolled state space or a feasible set of start windows to trigger a policy modification. CSC on the other hand, updates metrics on every event from every agent leading to triggers for many policy modification investigations.

Third, the policy modification spaces $\widehat{X}_i^t$ for the MDP-based and STN-based approaches are relatively rich, i.e., all policies supported by the unrolled state space for the MDP or start window shifts and joint moves in the STN. CSC policy modification spaces are very small: remove a method, add a method, switch to a sibling method. It isn't that we didn't want to make more complex moves, but getting the information to evaluate these moves properly is quite difficult. We believe that if you are in an area where you can't see very well, then you don't walk around there, because you might fall in a hole. The nonlinearity in the reward functions due to conjunctions, enables, disables, etc. assures us that these holes do exist.

The key design choices for a solution concept are the policy modification space $\widehat{\Pi}_i^t$ and the information η_i^t since the evaluation function $\widehat{V}(\cdot, \eta_i^t)$ and circumstance set $Z(\eta_i^t)$ depend on the latter. We believe that one should choose policy modification spaces for which your information is likely to match the true evaluation function. Formally, choose $\widehat{\Pi}_i^t \subset \Pi_i^t$ to maximize the probability that $\widehat{V}(\pi_1, \eta_i^t) > \widehat{V}(\pi_2, \eta_i^t)$ implies $V(\pi_1, \eta^*) > V(\pi_2, \eta^*)$ over $\pi_1, \pi_2 \in \widehat{\Pi}_i^t$. This idea of *predictability* seems to be a vital principle in these challenging domains. By inserting new methods at a lower priority, the Random Insertion baseline is restricted to a space that is less likely to damage the joint policy from the initial schedule. To deal with the restrictions that predictability induces, we advocate using information to create multiple evaluation functions that allow for multiple predictable policy modification spaces.

There seem to be lessons to be learned in how to construct this information: (1) *global* view: the MDP and STN approaches both leveraged the subjective view of agents and key parts of their evaluation functions. CSC used metrics that traveled across multiple subjective views giving a more global picture. (2) *accuracy* over approximation: information is typically a projection of either the true value function, joint policy or joint state given some assumption, e.g., greedy construction of frontier value, expected outcomes, evaluating with probability of getting positive quality. It seems to be better to get an accurate view of a smaller picture than an approximate view of the big picture, especially in domains with nonlinearity. (3) *timely* data: one must be able to act within the pace of the dynamics and uncertainty resolution of the environment. Random insertion had two key positive characteristics: it acted predictably and it acted often. Thus, to construct good policies, having timely information for making decisions is vital.

Clearly, CSC is not a complete solution. The MDP team contributed scenarios where MDPs can be fully unrolled to find the optimal solution. Their system performed well on these problems because they were relatively small, and the embedded lotteries were confined to a single agent. Therefore, they bypassed the bounded rationality and communication limitations that arise in general problems. However, they did identify reward functions for which current CSC criticality metrics cannot discriminate as well.

We are not claiming that CSC metrics are good universally, even though they performed extremely well in these problems. We described the system and proposed a framework to help develop a theory of why it's difficult to do well in these important domains and discover how we may build good solutions to the general problem. While there are many questions to be asked and answered, we believe the significance of the effort and the resulting empirical evidence indicates that CSC contains important *properties* of good solutions. The key ideas from this endeavor might be that predictability and criticality metrics are more important than the policy representation that is used. The MDP and STN approaches might be improved significantly if the information used to construct their evaluation functions were global, accurate, timely metrics and the space of policy changes were circumscribed to be more predictable.

Related Work

Earlier approaches to TAEMS problems include Generalized Partial Global Planning (GPGP) (Lesser et al. 2004) and Design-To-Criteria (DTC) (Wagner and Lesser 2000). These approaches used a notion of commitments whereby an agent promises to achieve quality on certain nodes by specified times. Similar to the MDP-base approach, agents maintain local policies to achieve their commitments and re-negotiate commitments when they cannot keep them or when the opportunity arises to achieve better quality.

Alternate scheduling models that address uncertainty are Simple Temporal Problems with Uncertainty (STPU) (Vidal and Fargier 1999) and Probabilistic Simple Temporal Problems (PSTP) (Tsamardinos 2002). STPUs model uncertain durations using lower and upper bounds. A polynomial algorithm can compute activity start times that satisfy all constraints as execution unfolds. If the constraints cannot be satisfied, STPUs provide no measure of the extent to which the constraints cannot be solved. PSTPs extend STPUs by using probability distributions to quantify duration uncertainty. PSTPs cannot be solved in polynomial time, so are not useful for constructing timely metrics.

A Distributed Constraint Optimization Problem (DCOP) based solution to CTAEMS problems is addressed in (Sultanik, Modi, and Regli 2007). Uncertainty modeling in DCOPs leads to state space explosion, making it infeasible to find solutions with current algorithms. Constraint Programming has been used to find upper bounds for optimal solutions, by averaging the optimal solutions in CTAEMS problems where uncertainty is eliminated (van Hoeve et al. 2007). These bounds assume prescient knowledge of uncertain outcomes. The timeliness of full centralization, partial centralization and decentralization schemes for multi-agent systems in dynamic domains and solving problems with graphical reward functions was investigated in (Harbers, Maheswaran, and Szekely 2007). Centralization is shown to be a poor strategy under bounded rationality and communication. Partial centralization that takes advantage of reward structure can make information sharing more timely.

Conclusion and Future Work

Real-time multi-agent planning and scheduling involving uncertainty, dynamism and a nonlinear reward function are extremely challenging. Extensions of traditional techniques such as MDPs and STNs do not immediately yield good solutions. New ideas are needed in this important and emerging problem area. We are currently in the process of extending our approach to deal with location reasoning as well as resources. These issues only exacerbate an already difficult problem. Through building the CSC system, extensive and independent testing, and developing a formalism to understand these problems, we submit that the ideas of predictability and global, accurate, timely metrics are a good starting point [5].

[5]The work presented here is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

Boddy, M.; Horling, B.; Phelps, J.; Goldman, R. P.; Vincent, R.; Long, A. C.; Kohout, B.; and Maheswaran, R. 2007. CTAEMS language specification: Version 2.04.

Decker, K., and Garvey, A. 2007. Darpa coordinators scenario generation cookbook: Version 2.7.

Harbers, T.; Maheswaran, R. T.; and Szekely, P. 2007. Centralized, distributed or something else? Making timely decisions in multi-agent systems. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, 738–743.

Lesser, V.; Decker, K.; Wagner, T.; Carver, N.; Garvey, A.; Horling, B.; Neiman, D.; Podorozhny, R.; Prasad, M. N.; Raja, A.; Vincent, R.; Xuan, P.; and Zhang, X. Q. 2004. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems* 9(1-2):87–143.

Musliner, D. J.; Durfee, E. H.; Wu, J.; Dolgov, D. A.; Goldman, R. P.; and Boddy, M. S. 2006. Coordinated plan management using multiagent MDPs. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*.

Smith, S.; Gallagher, A. T.; Zimmerman, T. L.; Barbulescu, L.; and Rubinstein, Z. 2007. Distributed management of flexible times schedules. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2007)*.

Sultanik, E.; Modi, P. J.; and Regli, W. C. 2007. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1531–1536.

Tsamardinos, I. 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Proceedings of the Second Hellenic Conference on Artificial Intelligence*, 97–108.

van Hoeve, W. J.; Gomes, C. P.; Selman, B.; and Lombardi, M. 2007. Optimal multi-agent scheduling with constraint programming. In *Proceedings of the Nineteenth Conference on Innovative Applications of Artificial Intelligence*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: From consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence* 11:23–45.

Wagner, T., and Lesser, V. R. 2000. Design-to-criteria scheduling: Real-time agent control. In *Agents Workshop on Infrastructure for Multi-Agent Systems*.

Witwicki, S., and Durfee, E. 2007. Commitment-driven distributed joint policy search. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2007)*.