Coordination of Highly Contingent Plans

David J. Musliner, Honeywell Laboratories, Minneapolis, MN, david.musliner@honeywell.com
Robert P. Goldman, SIFT, LLC, Minneapolis, MN, rpgoldman@sift.info
Edmund H. Durfee, Jianhui Wu, Dmitri A. Dolgov,
University of Michigan, Ann Arbor, MI, {durfee, jianhuiw, ddolgov}@umich.edu
Mark S. Boddy, Adventium Labs, Minneapolis, MN, mark.boddy@adventiumlabs.org

Abstract—*Conventional military planning systems construct plans with very limited flexibility. In the future, military plans will evolve into a much more expressive, contingent form. This paper describes how Honeywell's distributed Coordinator agents reason about complex domains to construct and execute highly contingent plans. The agents operate in a very dynamic environment in which complex hierarchical tasks can arrive unpredictably and the agents have to build coordinated joint plans on the fly, while execution proceeds. Using carefully limited forms of inter-agent communication, the agents develop agreements on their future coordinated behavior and rely on those agreements to build highly contingent plans (partial policies) that specify what actions they should take in a wide variety of possible futures. As mission execution proceeds and the tasks yield varying outcomes, the agents must rapidly, continually coordinate and adapt their plans. The result is a distributed multi-agent system capable of building and flexibly executing complex, highly-contingent coordinated mission plans.*

1. Introduction

Conventional military planning systems construct plans with very limited flexibility; often there is only a baseline operational plan with a few hand-crafted contingency branches that essentially amount to deploying reserved assets which otherwise remain unused. In the future, as automated planning systems become more sophisticated and military operations become more automated, military plans will evolve into a much more expressive, contingent form. Plans will be built to account, ahead of time, for operational tasks that take varying time, have varying levels of success, and should be combined in widely different ways depending on what earlier tasks (and adversaries) have accomplished. Constructing such contingent plans in a distributed coalition environment and coordinating the distributed execution of those plans will become much harder than current coalition activities. The DARPA COORDINATORS program is exploring the core computational issues that underlie exactly these problems.

This paper describes how Honeywell's distributed COORDINATOR agents reason about complex domains to construct and execute highly contingent plans. The agents operate in a very dynamic environment in which complex hierarchical tasks can arrive unpredictably and the agents have to build coordinated joint plans on the fly, while execution proceeds. Using carefully limited forms of inter-agent communication, the agents develop agreements on their future coordinated behavior and also develop highly contingent plans (partial policies) that specify what actions they should take in a wide variety of possible futures. As mission execution proceeds and the tasks yield varying outcomes, the agents must rapidly, continually coordinate and adapt their plans.

Our current solution combines restricted forms of inter-agent coordination agreements with dynamic, probabilistic projections of possible future worlds in the form of Markov Decision Problems (MDPs). By carefully guiding and pruning the projection, or unrolling, of the MDP model of possible future states, our COORDINATOR agents attempt to focus their decision-making attention on the partial plans with the highest probability of being useful. The MDP formulation allows our COORDINATOR agents to produce highly contingent plans in the form of partial policies, specifying what actions to take in all the possible future states explored so far. Novel technical elements of our COORDINATOR agents include their ability to exploit problem structure to dramatically reduce the complexity of future planning, their methods for guiding the MDP unrolling process, and their ability to continue unrolling during mission execution.

The result is a distributed multi-agent system capable of building and flexibly executing complex, highly-contingent coordinated mission plans.

2. The Coordinators Problem

Our work is being done in the context of the DARPA-funded COORDINATORS program, which aims to identify, prototype, and evaluate technical approaches to scheduling and managing distributed activity plans in dynamic environments. As a motivating example, consider the following scenario. A hostage has been taken and might be held in one of two possible locations. Rescuing the hostage requires that both possible locations are entered by special forces *simultaneously*. As the activities to move personnel and materiel into place are pursued, delays may occur or actions intended to achieve precursor objectives may have unexpected results (e.g., failure). COORDINATOR agent systems will be associated with the various human participants. COORDINATOR agents should monitor the distributed plans and manage them as the situation evolves, to increase their effectiveness and make them more likely to succeed.

In general, a set of COORDINATOR agents is meant to work together to maximize the reward gained by the group as a whole. In other words, the problem is to compute an effective *joint* policy for the agent society, in which the actions taken by one agent can depend on the state of the group as a whole, not just the local state of that agent. The agents are time-pressured: each agent must make timely action decisions during execution. Furthermore, the problem must be solved in a distributed fashion.

Although this is a problem of joint action, the problem solving is necessarily distributed, for reasons both definitional and practical. The definitional reasons include the fact that each agent has only a partial, local model of the problem, and the agents are prohibited (for organizational reasons) from building a complete joint model of the situation. The practical reasons include the sheer scope of the problem to be solved.

Each agent's partial problem model (aka domain model) includes the actions that the agent can execute, which are stochastic, rather than deterministic, and some of the actions its peers can perform. The problem model also provides *partial* information about the rewards that the society as a whole will receive for reaching various states. This model is not static: the agent can receive information about action outcomes and problem model updates during execution. Therefore, agents must be able to manage and reformulate policies reactively.

3. C-TÆMS

COORDINATORS researchers have jointly defined a common problem domain representation based on the original TÆMS language [1]. The new language, C-TÆMS [2], provides a semantically sound subset of the original language, representing multi-agent hierarchical tasks with stochastic outcomes and complex hard and soft interactions. Unlike other hierarchical task representations, C-TÆMS emphasizes complex reasoning about the utility of tasks, rather than emphasizing interactions between agents and the state of their environment.

C-TÆMS permits a modeler to describe hierarchically-structured tasks executed by multiple agents. A C-TÆMS task network has *nodes* representing *tasks* (complex actions) and *methods* (primitives).[1] Nodes are temporally extended: they have durations (which may vary probabilistically), and may be constrained by release times (earliest possible starts) and deadlines. Methods that violate their temporal constraints yield zero quality (and are said to have *failed*). At any time, each C-TÆMS agent can be executing at most one of its methods, and no method can be executed more than once.

A C-TÆMS model is a discrete stochastic model: methods have multiple possible outcomes. Outcomes dictate the *duration* of the method, its *quality*, and its *cost*. Quality is constrained to be non-negative, and duration must be an integer greater than zero. Cost is not being used in the current work. Quality and cost are unitless, and there is no fixed scheme for combining them into utilities. For the initial COORDINATORS experiments, we treat quality as non-normalized utility (we will use the terms "utility" and "quality" pretty much interchangeably).

[1]The terminology is somewhat unfortunate, since conventional HTN planners refer to their composite actions as *methods* and their primitives as operators.

To determine the overall utility of a C-TÆMS execution trace, we must have a mechanism for computing the quality of tasks (composite actions) from the quality of their children. Every task in the hierarchy has associated with it a "quality accumulation function" (QAF) that describes how the quality of its children are aggregated up the hierarchy. The QAFs combine both logical constraints on subtask execution and how quality accumulates. For example, a :MIN QAF specifies that all subtasks must be executed and must achieve some non-zero quality in order for the task itself to achieve quality, and the quality it achieves is equal to the minimum achieved by its subtasks. The :SYNCSUM QAF is an even more interesting case. Designed to capture one form of synchronization across agents, a :SYNCSUM task achieves quality that is the sum of all of its subtasks that start at the same time the earliest subtask starts. Any subtasks that start after the first one(s) cannot contribute quality to the parent task.

The quality of a given execution of a C-TÆMS task network is the quality the execution assigns to the root node of the task network. C-TÆMS task networks are constrained to be trees along the subtask relationships, so there is a unique root whose quality is to be evaluated. C-TÆMS task networks are required to have a deadline on their root nodes, so the notion of the end of a trace is well-defined. One may be able to determine bounds on the final quality of a task network before the end of the trace, but it is not in general possible to determine the quality prior to the end, and it may not even be possible to compute useful bounds.

Traditional planning languages model interactions between agents and the state of their environment through preconditions and postconditions. In contrast, C-TÆMS does not model environmental state change at all: the only thing that changes state is the task network. Without a notion of environment state, in C-TÆMS task interactions are modeled by "non-local effect" (NLE) links indicating inter-node relationships such as enablement, disablement, facilitation, and hindrance.

Figure 1 illustrates a simple version of the two-agent hostage-rescue problem described earlier. The whole diagram shows a global "objective" view of the problem, capturing primitive methods that can be executed by different agents (A and B). The COORDINATORS agents are *not* given this view. Instead, each is given a (typically) incomplete "subjective" view corresponding to what that individual agent would be aware of in the overall problem. The subjective view specifies a subset of the overall C-TÆMS problem, corresponding to the parts of the problem that the local agent can directly contribute to (e.g., a method the agent can execute or can enable for another agent) or that the local agent is directly affected by (e.g., a task that another agent can execute to enable one of the local agent's tasks). In Figure 1, the unshaded boxes indicate the subjective view of agent-A, who can perform the primitive methods Move-into-Position-A and Engage-A. The "enable" link indicates a non-local effect dictating that the Move-into-

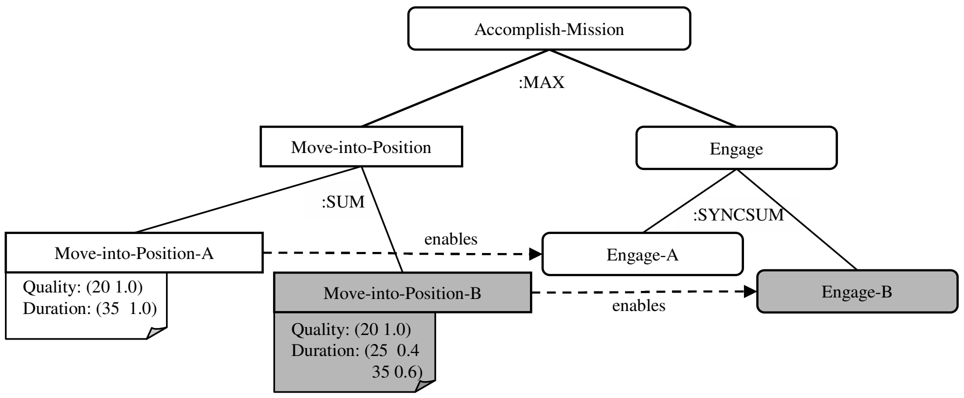

Figure 1: A simple C-TÆMS task network for two agents, illustrating some of the representation features. Some details have been omitted for brevity.

Position-A method must be completed successfully before the agent can begin the Engage-A method. The diagram also illustrates that methods may have stochastic expected outcomes; for example, agent-B's Move-into-Position-B method has a 40% chance of taking 25 time units and a 60% chance of taking 35 time units. The :SYNCSUM QAF on the Engage task encourages the agents to perform their subtasks starting at the same time (to retain the element of surprise).

4. Solution Approach: Markov Decision Processes

Given a C-TÆMS task network with stochastic method outcomes, we can frame the objective COORDINATORS problem as a multi-agent Markov Decision Process (MDP) [3]. Briefly, an MDP is akin to a finite state machine, except that transitions are probabilistic, rather than deterministic or nondeterministic. Agents may also receive reward (which may be either positive or negative) for entering some states. Typically, this reward is additive over any trajectory through the state space (some adjustments are needed in the case of MDPs of infinite duration). The solution to an MDP is a *policy* — an assignment of action choice to every state in the MDP — that maximizes *expected utility*. Expressing the COORDINATORS problem as an MDP provides a sound theoretical basis for decision-making and action under uncertainty. Furthermore, there are relatively simple, efficient algorithms for finding optimal policies. However, the state space size of the MDPs can be enormous.

A single COORDINATOR agent's C-TÆMS task model specifies a *finite-horizon* MDP. The problems are finite-horizon because C-TÆMS problems have finite duration, with no looping or method retries. However, the MDP tends to be quite large for even modest-sized C-TÆMS problems because of the branching factor associated with uncertain outcomes, and because of the temporal component of the problem. For example, even a single applicable method with three possible durations and three possible quality levels gives us a branching factor of nine. In addition, time is a critical aspect of TÆMS problems: methods consume time and NLEs can have associated delays (so WAIT is often a useful action alternative). Furthermore, an agent can always abort a method that it is executing, and choose to start a different method. So the branching factor is never less than two at every time tick, in a full consideration of the (single-agent) problem.

Multi-agent C-TÆMS MDPs are even worse. If one were to formulate a centralized COORDINATORS problem directly as an MDP, the action space would have to be a tuple of assignments of actions to each agent. Each agent's policy could be dependent on all the possible actions that the other agents could choose, and all the outcomes they could receive. Naturally this causes an explosion in the state space of the problem. Beyond complexity, there are other reasons we cannot construct the optimal multi-agent MDP. COORDINATORS problems are time-constrained and truly distributed: each COORDINATOR agent gets only a limited subjective view and a limited time to build and execute its plans, so forming a perfectly optimal, centralized joint policy is not feasible. Furthermore, information security policies may prevent the agents from sharing their local views completely.

Therefore, we have a developed a distributed COORDINATOR agent system that tries to retain the principled advantages of an MDP-based approach while supporting truly distributed operations and information hiding, in a time-adaptive manner. Each agent builds a *partial* MDP for its local subjective problem, to support its own decision-making about what actions (methods) it should perform. The partial MDP is incrementally extended as more deliberation time is available to the agent, so that it becomes complete and locally-optimal if sufficient time is available.

Because each agent's subjective view may not accurately convey how local method quality contributes to the overall team mission quality, simply solving local MDPs for optimal policies is not sufficient. We must have the agents communicate to share information about their plans and expectations, so that agents whose problems interact can coordinate effectively. To that end, our agents also have a coordination/negotiation capability that allows them to efficiently reach joint agreements about how they will coordinate over interactions portions of the full C-TÆMS problem.

5. Partial MDPs: “Informed Unrolling”

We refer to the process of converting a C-TÆMS probler into an MDP problem as “unrolling,” because it involve projecting forward from an initial state (where no method have been executed) to imagine future possible states of th C-TÆMS network in which some methods have been execute at particular times and have received particular outcome: The core unrolling algorithm is thus a simple state-spac enumeration process where an MDP state is expanded b creating the successor stats that result from each of the possibl action choices and their outcomes. These successor states ar added to an *openlist* of un-expanded states, and the proces ideally continues until the openlist is empty and the full MD state space has been enumerated.

Since full enumeration of even single-agent C-TÆM: MDPs is often impractical, we have developed a techniqu for heuristically-guiding the enumeration of a subspace of th full MDP. Our *informed unroller* (IU) algorithm prioritizes the openlist of states waiting to be unrolled based on an estimate of the likelihood that the state would be encountered when executing the optimal policy from the initial state. The intent is to guide the unrolling algorithm to explore the most-probable states first.

One cannot determine the probability of reaching a state without considering the policy followed by the agent. Therefore, the IU intersperses policy-formulation (using the Bellman backup algorithm) with unrolling. This means that we must be able to find an (approximately) optimal policy for partial MDP state spaces, which means we must have a heuristic to use to assign a quality estimate to leaf nodes in our search that do not represent complete execution traces. We have developed a suite of alternative heuristics for estimating intermediate state quality, since the problem of finding a good heuristic is quite difficult.

Early results from our evaluation of the IU algorithm against a complete solution of (small) MDPs are promising. For example, in Figure 2 we show a comparison of the performance of the informed unroller against the complete unrolling process. In these small test problems, the informed unroller is able to find a high-quality policy quickly and to return increasingly effective policies given more time. This allows the IU-agent to flexibly trade off the quality and timeliness of its policies.

The IU approach is related to the “approximate dynamic programming” algorithms discussed in the control theory and operations research literature [4]. These approaches derive approximate solutions to MDP-type problems by estimating, in various ways, the “cost to go” in leaf nodes of a limited-horizon portion of the full state space. While our exploration of the literature is not yet complete, initially we believe that a key difference in our IU approach is the notion of time-dependent horizon control and unrolling-guidance (vs. just estimation of leaf-node reward for policy derivation).

The IU method is a special case of the find-and-revise algorithm schema [5] (which is a generalization of algorithms such as LAO^* [6]). $LDFS$-family algorithms use knowledge of the initial state(s) and heuristics to generate a state subspace from which a policy can be abstracted. A find-and-revise algorithm finds a state in the network for which the current value estimate is inaccurate, and revises the value for that state (e.g., by generating successors, and propagating the value functions backwards in standard MDP fashion).

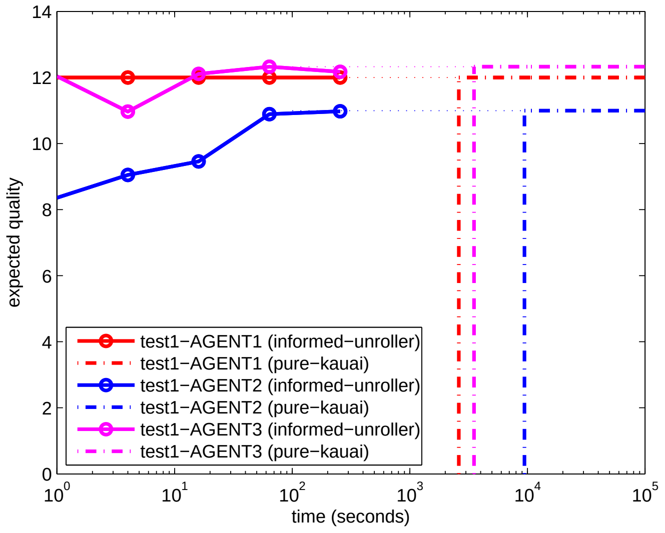

Figure 2: The Informed Unroller can find near-optimal policies much faster than building the complete MDP.

Our technique differs from the general case, and its instances, in substantial ways. LAO^* generates a state subspace from which the optimal policy can be provably derived. The IU, on the other hand, executes online, and might lack enough time to enumerate such a state subspace even if it knew exactly which states to include. The IU is an anytime algorithm, unlike LAO^*, which runs offline. For this reason, the IU makes no claims about policy optimality; indeed, it is not even guaranteed to generate a closed policy.

The general find-and-revise algorithm family can provide guarantees weaker than those of LAO^*, but those guarantees rely on having an admissible heuristic value function for states that have not been fully explored. However, even if we had an admissible heuristic, it is not at all clear that the IU should use it. An admissible heuristic will tend to push the policy expansion to explore states where it is **possible** that the optimum will be found, in order that we not miss the optimum. However, the IU is operating in a time-pressured domain. So we should not be encouraging the system to move towards promising unexplored areas — that will tend to leave the agent with a policy that is broad but shallow, and virtually guarantee that it will “fall off policy” during execution. Instead of admissibility, we must find a heuristic function that will cause the agent to tend to build policies that trade off considerations of optimal choice against completeness/robustness of the policy. It is possible that this heuristic should be time-dependent — as the agent runs out of time for policy development, the

IU's heuristic should focus more on robustness and less on optimality.

6. Coordination

When we consider multiple COORDINATOR agents, the problem expands to finding an optimal *joint policy*. This problem is challenging because:

- The number of possible local policies for agents is in general very large, so the product space of joint policies to search through can be astronomical.
- The size and distribution of the problem makes reasoning about the global behavior of the system impossible.

To address these practical limitations, our COORDINATOR agents do not try to solve the full optimal joint policy problem. Instead, they make several simplifying assumptions and restrict the forms of solutions they will be able to find, making the search for an approximately-optimal joint solution more tractable. Our agents use limited forms of negotiation to establish a set of inter-agent commitments. These commitments represent a partial set of agreements about which agent is performing which methods, at what times. The agents then rely on those commitments when generating their partial MDP policies. The commitments are used as both assumptions (e.g., another agent has agreed to perform a method that will enable my action) and as obligations (e.g., I have agreed to perform a method that will enable another agent). Assumptions such as remote enablement agreements can be built into the local problem model by including "proxy" methods that enable the local method at the agreed-upon time. Obligations to execute methods by a particular time are met by adding extra reward to the MDP in states that satisfy the commitment. These two mechanisms bias the MDP policy-generation process towards policies that rely upon and satisfy the agent's commitments.

There are several ways in which this approach may result in sub-optimal behavior. For example, the actual optimal policy set may not adhere to a static set of commitments: to behave optimally, agents may have to adjust which enablements they will accomplish depending on how prior methods execute. To mitigate this weakness, our agents deliberate and negotiate continually, so that they can manage and adapt their commitment set and policies on the fly as methods execute.

7. Conclusions

Our multi-agent coordination system uses limited forms of negotiated commitments to bias partial-MDP policy derivation. The resulting agents are able to very quickly create initial coordinated policies, improve those policies given more deliberation time, and adapt the policies as new information arrives, including both method outcomes and new C-TÆMS problems.

In the context of coalition operations, where different agents may not be able to share some portions of their intentions, these techniques can still be applied. Enforcing information security or privacy policies could be done on a local-agent level, preventing the agent from establishing commitments about private intentions (e.g., not telling other agents that it intends to execute a particular method or task). The resulting system would be expected to perform less-optimally, given the restrictions on its search for joint policies, but the system should still be robust and capable of establishing coordinated behavior on the portions of the problem over which agents are willing to communicate.

Acknowledgments

This material is based upon work supported by the DARPA/IPTO COORDINATORS program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the U.S. Government, or the Air Force Research Laboratory.

References

[1] B. Horling, V. Lesser, R. Vincent, T. Wagner, A. Raja, S. Zhang, K. Decker, and A. Garvey, "The TÆMS white paper," University of Massachussetts, Amherst, Computer Science Department, Tech. Rep., Jan. 1999.

[2] M. Boddy, B. Horling, J. Phelps, R. P. Goldman, and R. Vincent, "C-TÆMS language specification," Apr. 2005, unpublished; available from this paper's authors.

[3] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[4] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC," in *Proc. Conf. on Decision and Control*, 2005.

[5] B. Bonet and H. Geffner, "Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to MDPs," in *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*, D. Long, S. F. Smith, D. Borrajo, and L. McCluskey, Eds., Jun. 2006, pp. 142–151.

[6] E. A. Hansen and S. Zilberstein, "LAO: a heuristic search algorithm that finds solutions with loops," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 35–62, 2001.