Assumption-based Decentralized HTN Planning

Ugur Kuter and **Robert P. Goldman** and **Josh Hamell**
SIFT, LLC
319 1st Ave N., Suite 400,
Minneapolis, MN 55401, USA
{ukuter, rpgoldman, jhamell}@sift.net

Abstract

This paper describes our approach to decentralized planning via Hierarchical Task Networks (HTNs), which we call Autonomy and Rationale Coordination Architecture for Decentralized Environments (ARCADE). ARCADE is a decentralized AI planning framework that can incorporate a number of SHOP2 HTN planner instances. Each SHOP2 instance may have a different HTN planning domain definition than the others in the framework. ARCADE does not assume full communications among the planners. For this reason, ARCADE planners must make and manage assumptions about parts of the world state that are not visible to them, including the tasks and plans of other planners. The individual planners also must operate asynchronously, and may receive new tasks, either from outside, or from other planners in ARCADE.

In this paper, we describe our assumption-based planning approach and how ARCADE coordinates multiple, asynchronously interacting HTN planners, using *assumptions* and *task queues*. We first present a formal framework, Assumption-based, Decentralized Total-order Simple Task Network (DTSTN) planning, based on Total-order Simple Task Network planning. This is necessary because of our use of SHOP2-style task semantics, instead of goal semantics. Then we describe the ARCADE framework, and how it implements the framework. Finally, we present preliminary experimental results in a simplified air operations planning domain, which shows that ARCADE realizes the expected speed-up when applied to weakly coupled planning problems. We conclude with directions for future work.

Introduction

Existing distributed and multi-agent planning systems (Torreño et al. 2017) typically focus on deterministic planning problems, with relatively simple models. They also typically assume a *single* overall planning task that must be distributed among multiple agents. Most practical applications for decentralized planning (e.g., military operations, UAV planning, and others) involve independent planners and reasoners that are responsible for accomplishing different tasks under large-scale uncertainty, while communicating their intentions and coordinating their actions. These planners often are not handed a single, large problem to be decomposed and then solved. Instead, these planners often receive their own planning problems to solve based on the organizational structures in which they are embedded (e.g., logistics and manufacturing systems separately plan to secure inputs and to make products). They may also receive additional tasks at runtime.

The problems we are interested in also involve limited and unreliable communications. Thus, our planners must operate under assumptions about peer decisions and states, where knowledge is not fully shared. Finally, in these applications the classical assumption of complete information and predictability is typically difficult or impossible to apply. For example, Seuken and Zilberstein (2008) address partial-observability and uncertainty during planning; however, these approaches cannot scale up to the large-scale planning problems and use closed-world formalisms.

Autonomy and Rationale Coordination Architecture for Decentralized Environments (ARCADE) is a decentralized planning architecture that allows multiple SHOP2 (Goldman and Kuter 2018a; Nau et al. 2003) HTN planner instances to generate plans for planning tasks concurrently and asynchronously. Each planner may generate plans for tasks issued by other planners or received as input from outside. Our contributions in this paper are as follows:

- We present a formalism for assumption-based HTN planning, which allows SHOP2 to generate plans for execution in the presence of other (cooperative) agents, when agent-to-agent communications are unreliable and the environment is not fully observable.
- We describe ARCADE, our decentralized planning framework based on the above formalism. ARCADE takes as input a number of HTN planning problem specifications for all or a subset of the planners. ARCADE then coordinates the asynchronous operations of multiple SHOP2 instances.
- We describe how ARCADE communicates tasks to the planners by generating new HTN planning problem specifications, and by publishing those specifications. The planners can sign up to meet those requests if the tasks involved belong to the domain descriptions of those planners, and involve domain entities which those planners control. During the decentralized planning process, ARCADE ensures the plans generated in this way are sound and consistent, brokering solutions to conflicts between the decisions made by different planners.
- We are currently using ARCADE in various Air Operations planning scenarios. We present our preliminary experi-

ments and results in a high-level version of this domain, developed for publication purposes. Our preliminary results are promising: decentralized planning shows substantial scalability improvements over centralized planning via SHOP2, as one would expect. We also provide a representative result on the experiments we are conducting on the various individual components of ARCADE.

In the immediately following section we start by reviewing totally ordered simple task network (TSTN) planning. We then build our framework, *Assumption-based, Decentralized* TSTN planning, on the basic TSTN definitions. Critically, these definitions allow us to characterize what it means for Decentralized TSTN (DTSTN) plans to be consistent with each other. We then explain the ARCADE approach, which builds consistent sets of DTSTN plans, using assumptions to lazily bind resources to tasks, and to enable agents to reason in the context of beliefs about each others' likely actions and state. We present preliminary experimental results that show the efficiency of our approach. Finally, we conclude with a review of related work and conclusions (including future directions).

Preliminaries: TSTN planning

We use the same definitions for logical substitutions, atoms, constant and variable symbols, positive and negative literals in a finite function-free first-order language, as in (Ghallab, Nau, and Traverso 2004). A *state* is a collection, s, of ground atoms. In our work, we adopt a restricted case of HTN planning called *Total-order Simple Task Network* (TSTN) planning (Ghallab, Nau, and Traverso 2004):[1]

```
(:action start-order
 :parameters
  (?o - order ?n ?n1 - count)
 :precondition (and
                 (waiting ?o)
                 (stacks-avail ?n)
                 (next-count ?n1 ?n))
 :effect (and (not (waiting ?o))
              (started ?o)
              (not (stacks-avail ?n))
              (stacks-avail ?n1)))
```

Figure 1: An example operator schema from the openstacks domain (Helmert, Do, and Refanidis 2010). SHOP2 can use PDDL action definitions.

- A TSTN domain is a quadruple:

$$\mathcal{D} = \langle \mathrm{ops}(\mathcal{D}), \mathrm{tasks}(\mathcal{D}), \mathrm{meths}(\mathcal{D}), \omega(\mathcal{D}) \rangle$$

- Each operator $o \in \mathrm{ops}(\mathcal{D})$ is a triple

$$o = \langle \mathrm{name}(o), \mathrm{precond}(o), \mathrm{effects}(o) \rangle$$

where $\mathrm{name}(o)$ is a *task* (see below), and $\mathrm{precond}(o)$ and $\mathrm{effects}(o)$ are sets of literals called o's *preconditions* and *effects*. See Figure 1 for an example operator schema, and Figure 2 for an operator. If a state s satisfies $\mathrm{precond}(o)$, then o is *executable* in s, producing the state $\gamma(s, o) = (s - \{\text{all negated atoms in effects}(o)\}) \cup \{\text{all non-negated atoms in effects}(o)\}$.

```
(:action (start-order o1 s2 s1)
 :precondition (and
                 (waiting o1)
                 (stacks-avail s2)
                 (next-count s2 s1))
 :effect (and (not (waiting o1))
              (started o1)
              (not (stacks-avail s2))
              (stacks-avail s1)))
```

Figure 2: Operator from the openstacks schema in Figure 1. In this case `o1`, `s1`, and `s2` $\in \omega(\mathcal{D})$, and the task is `(start-order o1 s2 s1)`.

- $\mathrm{tasks}(\mathcal{D})$ is the finite set of ground tasks, such that $\mathrm{tasks}(\mathcal{D}) = \mathrm{prims}(\mathcal{D}) \cup \mathrm{comps}(\mathcal{D})$, where $\mathrm{prims}(\mathcal{D})$ is the set of primitive tasks and $\mathrm{comps}(\mathcal{D})$ is the (disjoint) set of nonprimitive (or complex) tasks in the planning domain.

  A task, t, is a symbolic representation of an activity. Syntactically, a task looks like a term (functor and arguments from the universe of the domain). If t is also the name of an operator, then τ is *primitive*; otherwise τ is *nonprimitive*. Primitive tasks can be instantiated into actions, and nonprimitive tasks need to be decomposed into subtasks.
- $\omega(\mathcal{D})$ is the universe of entities in the planning domain. We have seen in Figures 1 and 2, that $\omega(\mathcal{D})$ is used to form tasks (and hence operator and method names).
- A *method*, m, is a prescription for how to decompose a task into subtasks. m is a tuple:

$$m = \langle \mathrm{task}(m), \mathrm{precond}(m), \mathrm{subtasks}(m) \rangle,$$

where $\mathrm{task}(m) \in \mathrm{tasks}(\mathcal{D})$ is the task m can decompose, $\mathrm{precond}(m)$ is a set of preconditions, and $\mathrm{subtasks}(m) = (t_1, \ldots, t_j), t_i \in \mathrm{tasks}(\mathcal{D})$ is a sequence of subtasks, the *expansion* of $\mathrm{task}(m)$. See Figure 3 for an example.

```
(:pddl-method (open-all-stacks)
  open-a-stack-and-recurse
  (exists (?n ?n1 - count)
    (and (stacks-avail ?n)
         (next-count ?n ?n1)))
  (:ordered (open-new-stack ?n ?n1)
            (open-all-stacks)))
```

Figure 3: Example method from the openstacks domain for the task `(open-all-stacks)`. The precondition is that there be a stack available (an `?n1` that is not yet open). The subtasks are to open a new stack, and then open any remaining stacks, recursively. This is a lifted method schema, corresponding to multiple ground methods.

A TSTN planning problem is a tuple: $P = \langle \mathcal{D}, s_0, T_0 \rangle$, where $\mathcal{D}$ is a TSTN planning domain, s_0 is the initial state,

[1] In future work, we plan to generalize this – see discussion in the Conclusions.

and T_0 is an initial sequence of tasks. (1) If T_0 is the empty sequence, ϵ then the only solution is the empty plan $\pi = \langle\rangle$, and π's *derivation* (the sequence of actions and method instances used to produce π) is $\delta = \langle\rangle$. If the current set of tasks is $t_1 \dots t_n$ and (2) t_1 is a primitive task, there is an operator α with name$(\alpha) = t_1$, and α is executable in s_i producing a state s_1, then if $\mathcal{P}' = \langle \mathcal{D}, s_1, T' \rangle$ has a solution π with derivation δ, then the plan $\alpha \bullet \pi$ is a solution to w_i (where $\bullet$ is concatenation) whose derivation is $\alpha \bullet \delta$. (3) If t_1 is nonprimitive and there is a method m such that task$(m) = t_1$, and if s_0 satisfies precond(m), and if $P' = (s_0, \text{subtasks}(m) \bullet T', O, M)$ has a solution π with derivation δ such that δ only uses objects from the agent i's tasks T, then $\alpha \bullet \pi$ is a solution to P and its derivation is $m \bullet \delta$.

Assumption-based Decentralized TSTN Planning

In ARCADE, an *assumption* is an ordered pair $e = \langle \mathsf{cond}, \mathsf{cost} \rangle$, where cond is a literal and cost is a non-negative real number that denotes the cost of validating the cond. More precisely, the cost is a heuristic estimate of the cost of checking to see whether the assumption is guaranteed to hold at run time or not. Two assumptions, $e_1 = \langle \mathsf{cond}_1, \mathsf{cost}_1 \rangle$ and $e_2 = \langle \mathsf{cond}_2, \mathsf{cost}_2 \rangle$ are *inconsistent*, if either (1) $\mathsf{cond}_1 = \text{not}(\mathsf{cond}_2)$, or *vice versa*; or (2) $\mathsf{cond}_1 = \mathsf{cond}_2$ and $\mathsf{cost}_1 \neq \mathsf{cost}_2$. Assumptions that are not inconsistent are consistent.

A *decentralized TSTN planning agent* is a tuple of the form $A = \langle \mathcal{D}, \omega(A), \text{ops}(A), \text{meths}(A) \rangle$, where $\mathcal{D}$ is a TSTN planning domain as defined above, $Q(A)$ is a *task agenda*, $\omega(A) \subseteq \omega(\mathcal{D})$ is the subset of entities that the agent can manipulate, $\text{ops}(A) \subseteq \text{ops}(\mathcal{D})$ and $\text{meths}(A) \subseteq \text{meths}(\mathcal{D})$ are the sets of tasks, operators, and methods for this agent. The task agenda of a planning agent is a basic queue data structure, which only allows tasks to be accomplished in chronological order.

Note that the tasks of an agent A, $\text{tasks}(A) \subseteq \text{tasks}(\mathcal{D})$ is defined as $\bigcup_{o \in \text{ops}(A)} \text{name}(o) \cup \bigcup_{m \in \text{meths}(A)} \text{task}(m)$. In other words, A can decompose a task $t \in \text{tasks}(A)$ as long as it has either an operator or a method definition for t. Otherwise, A pauses its planning process, and requests ARCADE to find another agent that can perform t for it. When ARCADE receives such a request, ARCADE sends t to the other planning agents in the framework. If there exist one or more agents that can achieve t, these agents can start planning for t via their own methods and operators. For example, in the openstacks shipping domain, we could imagine an agent that is responsible for shipping the orders, and an agent that is responsible for adding the products to stacks.

We define an *assumption-based planning state*, s, as a collection of ground literals, $\text{facts}(s)$, and assumptions $\text{assumps}(s)$. The collection of literals in an assumption-based state describes the facts that a planning agent knows to be either true or false. The assumptions model the beliefs (as opposed to the knowledge) of the agent. That is, if an assumption of the form $\langle \mathsf{cond}, \mathsf{cost} \rangle$ is in the agent's state, where cond may be a positive or negative ground literal, this means that the agent makes a belief assertion, however, the agent does not know whether that assertion is correct or not. Validating that assertion is costly during planning; if the agent's assertion is proved to be wrong, any assumption-based plan that is conditional on assertion must be repaired. The cost value in the assumption estimates this cost.

An assumption-based planning state is *consistent* if: (1) the known facts, $\text{facts}(s)$ are consistent, (2) there does not exist an assumption $a_\perp \in \text{assumps}(s)$ such that $a_\perp = \langle \mathsf{cond}, \mathsf{cost} \rangle$ and the negation of cond is in $\text{facts}(s)$ (3) the set of assumptions is consistent.

An *assumption-based plan*, κ, is a sequence of pairs of the form $\langle b, \alpha \rangle$ where b is an assumption-based state and α is an action. Two assumption-based plans, κ_1 and κ_2, are consistent if (1) κ_1 and κ_2 are individually consistent, (2) κ_1 and κ_2 are well-formed TSTN plans, and (3) for all $(b_1, \alpha_1) \in \pi_1$ and $(b_2, \alpha_2) \in \pi_2$, b_1 and b_2 are consistent. We trivially generalize the definition of pairwise consistency to group consistency being the case of all pairs being consistent.

A *decentralized TSTN planning problem* is a tuple of the form $P = (\mathcal{A}, B_0, \mathcal{T}_0)$ where $\mathcal{A}$ is the finite set of planning agents for the decentralized planning problem. B_0 is a collection of *initial* assumption-based states such that for each $A \in \mathcal{A}$, there exists an assumption-based state $b_0(A) \in B_0$. Similarly, $\mathcal{T}_0$ is a collection of TSTNs such that there is a TSTN $T_0(A) \in \mathcal{T}_0$ for each agent $A \in \mathcal{A}$. A *solution* to a decentralized TSTN planning problem P is a collection of assumption-based plans that are consistent.

Decentralized Planning Framework

We designed our decentralized planning framework with the following objectives in mind:

1. *Asynchronous decentralization and planning:* Different SHOP2 instances must be able to receive their TSTN planning problems at different points in time during decentralized planning and they must be able to work on those problems concurrently, and each at its own pace.
2. *Task-centric assumption-based coordination:* SHOP2 instances must be able to exchange subtasks during planning, based on the assumptions each makes and whether or not a planner is capable of generating plans for specific tasks.
3. *Hierarchical localized plan adaptation and repair:* Each SHOP2 instance in the decentralized planning framework must use localized replanning and plan repair algorithms (Goldman and Kuter 2018b) to provide consistency and correctness over its assumptions, which might be invalidated by the decisions and plans made by other SHOP2 instances.

We will focus on objectives (1) and (2) in the rest of this paper. We discuss objective (3) in a forthcoming, related paper (Goldman and Kuter 2018b). Given a decentralized TSTN planning problem $P = (\mathcal{A}, B_0, \mathcal{T}_0)$ as defined above, a SHOP2 planning agent. A_i, starts to generate solution plans to the tasks in its agenda $Q(A_i)$. Initially, this agenda contains only the initial task sequence T_0 that is specified for this agent in the input planning problem description. Given its initial state b_0, A_i extracts a task t_0 from its agenda, creates a local

TSTN planning problem $P_i = (\mathcal{D}, b_0, \{t_0\})$, and calls SHOP2 on this local TSTN planning problem to generate a solution plan.

During local TSTN planning, if A_i generates a task t such that $t \notin \text{tasks}(A_i)$,[2] then A_i escalates this task to ARCADE, which in turn publishes t for all of the other agents in $\mathcal{A}$. If there exists at least one other agent, say A_j, in ARCADE such that $t \in \text{tasks}(A_j)$, A_j will insert t into its own task agenda, $Q(A_j)$. If no agent is able to generate a plan for t, then ARCADE notifies A_i and A_i backtracks to consider other alternative task decompositions. This is done by imposing a timeout, a settable parameter, on agent-to-agent requests.

In our current system, if A_i escalates a task t to ARCADE for other agents to plan for t, A_i pauses its planning until a response comes back. This is done to maintain a consistent planning state throughout all the tasks in A_i's agenda while different planners work on different tasks at different times. A_i incorporates the supplying agent ($agent_j$)'s plan into its own plans. This involves progressing the current planning state of the A_i by applying the actions in the plan being incorporated.

If multiple agents generate plans for t, A_i selects one of them, and drops the others. A_i (through ARCADE), informs any un-selected agents that they can drop their plans for t. In our current implementation, the choice over the plans generated by the other agents is greedy: A_i selects the first plan it receives from the other agents and rejects any other responses. We are developing a theory for how to use the assumption costs to make such selections. ARCADE already uses the cost of validating an assumption as a way to declobber plans of different agents, if necessary (see below), and a similar mechanism can be used as a heuristic to choose between the plans of different agents.

In ARCADE, a planning agent **generates assumptions** in two cases: (1) while it is accomplishing a task and (2) the agent generates assumptions while incorporating another agent's plans into its solution. We discuss these two cases below.

Generating assumptions during planning Although our formal discussion describes TSTNs and DTSTSs in a propositional context, SHOP2 is, in fact, a *lifted* (first order) planner, and performs many of its tasks using *unification*.

During TSTN planning for a task t with precondition $p(x)$, for example, if SHOP2 cannot find a substitution for x satisfying $p(x)$, it normally backtracks. In ARCADE, we modified this behavior so that SHOP2 can generate an assumption, instead of backtracking, and continuing the search with that assumption asserted into its state representation.

For example, suppose t is a task for taking an image, for which it must assign a UAV with appropriate instruments (e.g., cameras). In the current state it fails to do so, because its planning state is incomplete, and it cannot determine whether the suitable UAV will be available. Standard SHOP2 would backtrack at this point. In ARCADE, SHOP2 has the alternative of generating an *assumption*, here for example assuming that uav1 will be available: $\langle \mathsf{available}(\mathsf{uav1},\ 1600), 1\rangle$.

One of the key challenges for a planner is to determine whether an assumption is too specific or too general. Most modern planning systems eagerly ground variables in their action schemas. IPC planners typically preprocess and ground problem specifications and domain models *a priori*. Lifted planners such as SHOP2 ground variables on the fly but they do so at the first point where a ground value is matched during search. If those variable-binding choices do not lead to solution plans later on in search, the planner backtracks and tries other possible groundings (Nau et al. 2003). This is a major scalability issue, even for SHOP2; as has been shown in experiments over a decade now, SHOP2's performance can degrade exponentially (Nau et al. 2003).

In decentralized planning, this performance degradation is more dire because of uncertainty and incomplete information induced due to the operations of multiple planners. In addition, backtracking over decisions that involve other agents is even more time-consuming than it is in centralized planning.

In particular, an agent A_i knows the set of constant symbols in $\omega(A_i)$, but it does not know about $\omega(A_j)$ for $j \neq i$. Thus, it cannot generate an assumption that involves grounded conditions about other agent's planning states. To address this challenge, we have developed a *late-binding* approach for SHOP2 to use logical *skolemization* as in automated reasoning works (Genesereth and Nilsson 1987). In particular, SHOP2 delays binding a variable symbol that appears in an assumption condition; instead, SHOP2 replaces it with a skolem function that specifies the properties, as constraints, of the constant that should be bound to that variable for a sound plan. After A_i's SHOP2 generates a plan with skolem functions in it as a solution, ARCADE post-processes the plan and generates variable bindings according to the generated constraints during planning. Our preliminary experiments in the subsequent section show the potential benefits of this approach. In principle, however, post-processing may still fail to generate bindings successfully for some of the skolem functions. In that case, ARCADE treats the binding failure as a plan-failure discrepancy and triggers its plan adaptation and repair process.

Coordinating over other agents' assumptions When a planning agent A_i receives an assumption-based plan κ_j from another agent A_j and attempts to incorporate κ_j into its plans, it may find that κ_j is inconsistent with its own plan, κ_i. When merging, each agent will verify the consistency of the plans, using the definitions given above. If an inconsistency is found between two assumptions, ARCADE compares the cost of validating the inconsistent assumptions. If A_i's own cost of adapting to assumption violation is greater than those of A_j, then ARCADE notifies A_j about the contradictions and the cost models over them, requiring A_j to adapt to its assumptions. Otherwise, A_i casts the contradiction as a plan discrepancy (using our plan repair framework (Goldman and Kuter 2018b)) and adapts its own plans. In ARCADE, currently all ties are broken randomly.

If $agent_j$ cannot adapt its plans to alleviate the contrac-

[2]This will happen when A_i expands a task t' using a method with t as a subtask.

tion, then it returns a failure. At this point, A_i attempts to adapt its plan even if it was deemed heuristically more costly. If A_i generates a solution, then it incorporates it into its assumption-based plans. Otherwise, the goal is not satisfied and A_i returns failure.

Calculating costs of assumption violations When an agent generates assumptions, it also rapidly calculates a cost estimate. This estimate attempts to characterize how much work it would be to adapt its plans (if it can at all), if the assumption is violated. In ARCADE, this is done by a combination of plan critiquing and generating adaptation options (i.e., alternative plan repairs that are different from each other in terms of the objects used and how the adapted tasks are accomplished). The latter is the topic of a forthcoming paper. We have described our work on plan critiquing in (Goldman, Kuter, and Schneider 2012; Mueller et al. 2017) in detail; we summarize it briefly below.

Our plan critiquing system, Murphy (Goldman, Kuter, and Schneider 2012), generates counterexamples for a SHOP2 plan to explain possible breakdown cases for that plan. In ARCADE, Murphy critiques plans to assess assumption violations due to the plans and assumptions made by the other planners in the framework. Murphy translates a plan into a "counter-planning" problem, combining a representation of the initial plan with the definition of a set of uncontrolled actions. These uncontrolled actions may be the actions of other agents in the ARCADE framework, actions of the exogenous agents in the environment, either friendly, indifferent or hostile, or they may be exogenous events that simply occur. The result of this translation is a disjunctive planning problem that we further process in order to play into the strengths of existing classical planners. Using this formulation, a classical planner can find counterexamples that illustrate ways a plan may go awry.

In ARCADE, the translation yields a new counter-planning problem and domain, in which goals are logically-negated assumption conditions. For example, an assumption of the form (cond cost) produces a goal literal of the form ($\neg$cond). Such counter-planning goals can then be solved to find a counterexample or, if searched exhaustively without finding a counterexample, indicates that no counterexample exists. There are three components to the translation process: (1) generating a "skeleton" for the original plan that ensures that the original plan actions occur in the proper order; (2) formulating a goal expression that will cause a planner to search for a counterexample and (3) encoding the resulting disjunctive goal expression in a way that is amenable to the use of a standard PDDL planner.

If a counterexample is found, the agent generates multiple adaptations of the plan, treating the counterexample as a discrepancy that breaks the plan. For each of these adaptations, the agent calculates the cost of the new plans. The adaptation with the maximum cost is used to update the cost of violating the assumption. Intuitively, this approach calculates the worst case that i's plans must be adapted if they are violated by other agent's assumptions and plans. As described above, the cost estimate of violating the assumption is then used to compare assumptions of different ARCADE agents and guide coordination and deconfliction of the agents' knowledge states.

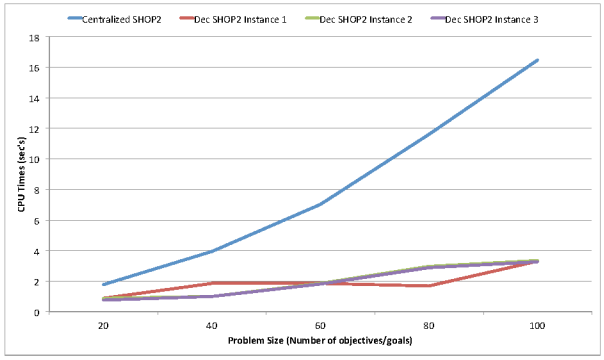

Figure 4: Comparison of run times using SHOP2 in a centralized fashion versus using our decentralized approach.

Implementation and Evaluations

We have implemented our ARCADE framework in Common Lisp, using SHOP2 as the planner for the planning agents. ARCADE allows an arbitrary number of planning agents; our current experiments use a 4-planner instance of this framework. We use an air-operations planning, command, and control domain, that we have developed, and which is a generalization of the OpenStacks domain from the International Planning Competition (Helmert, Do, and Refanidis 2010). In this domain, planning operators specify air missions to achieve a given set objectives (i.e., locations), their scheduling, and resource usage to create plans that can be given the human military operators to execute. The planning problems in this domain include a varying number of aircraft, bases, objectives, locations on a physical map layout. Our HTN methods encode strategies that describe how generate groups of missions to achieve all of the objectives under different conditions of the world.

Using the above framework, we are currently conducting several experiments with ARCADE to evaluate the approach's performance. Below, we present and discuss some representative results of our experiments.

Figure 4 shows our preliminary results to confirm the runtime scalability benefits of a decentralized approach, in contrast to a single-agent, centralized planning. Here, we use the same setting as described above, planning problems that involved tens of aircraft distributed across five bases. The x-axis shows the number of objectives we varied for evaluation purposes.

This experiment was intended as a sanity check on our decentralized planning approach. Decentralized planning over non-conflicting tasks should achieve near linear (optimal) speed-up over centralized planning. As shown in Figure 4, our experiments confirmed this expectation: as a function of increasing problem size (i.e., increasing number of tasks to be accomplished), the runtime performance of planner nodes in the decentralized approach remains linear, whereas

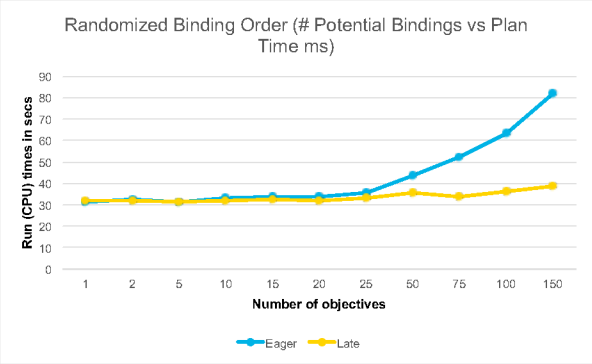

Figure 5: Comparison of eager (original) variable binding strategy in SHOP2's backtracking search and late binding.

SHOP2's runtimes increase substantially.

In addition to the scalability of the overall framework, we are also experimenting to identify which factors are key drivers of the performance of individual SHOP2 instances in our air-operations problems. One of the factors we explored is the impact of late-binding in the assumptions generated during planning. Figure 5 shows the results of a preliminary experiment using the same setting as above. The x-axis again denotes the number of objectives we have varied for the purposes here. This experiment compares the original eager-binding approach of SHOP2 with a late-binding strategy in which we randomly varied the variables that SHOP2 eagerly binds and those variables it skolemizes. The results suggest a significant performance improvement with the latter approach.

Related Work

Our work builds on the great strides on distributed, decentralized and multi-agent planning works over the years. Torreño, et al. (2017) provide an excellent up-to-date survey covering historical developments and the current state of the art. Unlike most distributed, multi-agent planning work, which focuses on domain-independent deterministic planning and heuristic search, we focus on HTN planning (Ghallab, Nau, and Traverso 2004; Nau et al. 2003; Tate, Drabble, and Kirby 1994; Wilkins 1988). HTN planning is critical to our applications, because it permits our system to capture doctrinal and procedural tasks, constraints and communication strategies. Modern HTN planners also incorporate considerably greater expressive power than standard IPC/PDDL-style planners. They typically include: capabilities for calling attached procedures, making axiomatic inferences, and performing numeric computations, all of which are critical for this approach.

Previous work on hierarchical planning and scheduling includes priority-based task planning for UAVs (Musliner et al. 2011), hybrid HTN planning and scheduling (e.g., (Schattenberg 2009; Elkawkagy et al. 2010; Bercher et al. 2014; Bercher, Keen, and Biundo 2014), SharedPlans (Grosz and Kraus 1996), multi-agent HTN planning (Dix et al. 2003; Gancet et al. 2005; Elkawkagy and Biundo 2011), HTNs with temporal reasoning (Goldman 2006; Fdez-Olivares et al. 2006), and HTNs for planning under uncertainty in world states (e.g., (Kuter et al. 2009)). Although these systems have proven themselves on research benchmarks, they (1) make specific, simplifying assumptions, which are also independent from each other and do not align well, (2) do not scale up to our planning problems in the air operations planning domain, and (3) cannot perform resilient synchronized planning.

Our work is most similar to the Shared Activity Coordination (SHAC) system of Clement and Barrett (2003). SHAC was built to support continual planning and execution for NASA missions with multiple different assets, multiple stakeholders, and limited communications. However, the nature of the communications limitations in SHAC is quite different from that of ARCADE: theirs comes from the physics of the domain (orbits, rotations, etc.), and is largely predictable, whereas ours comes from unpredictable external factors. Also, their communications issues are generally planner to executor (probe), rather than planner to planner, like ARCADE's. This shows in the differing nature of their consensus approach, and in their planning agents operating in regular cycles, unlike ours, and the SHAC agents do not require assumption-based planning.

Existing work on Multi-Agent STRIPS (MA-STRIPS) and its successors (Brafman and Domshlak 2008; Nissim, Brafman, and Domshlak 2010; Nissim and Brafman 2013) are directly related to ARCADE. MA-STRIPS's concept of private/public actions to define an interface among the cooperating agents is similar to the assignment of subtasks with possible agents in our decentralized HTN models. One difference with formalism is that ARCADE's agents do not coordinate to generate a single solution plan and interact to satisfy its causal model during planning, as MA-STRIPS and its successors are naturally designed to do as a classical planning approach. Instead, ARCADE generates problem decomposition and refinement via TSTN planning and each agent generates plans for tasks. Another key difference between the MA-STRIPS family and ARCADE is the concept of commitments, where the former aims to use commitments to ensure the correctness of the output multi-agent plan. Instead, ARCADE treats most of the plan commitments of an agent as assumptions and ensures correctness via plan repair if those assumptions are broken by other agents' plans.

Note that an approach such as MA-STRIPS may be superior if the task involves taking a large planning problem and decomposing it into multiple subproblems for computational or even execution efficiency. These approaches would provide at least some support for automatic domain decomposition. In ARCADE, on the other hand, the domain decomposition is assumed top be given. For a new situation, built from scratch, that would involve additional work. ARCADE, however, was built for applications in large human organizations, where the structure of the organization and the scope of responsibility and authority are a given and a part of the problem to which the system must be adapted. This is why ARCADE starts from a given problem decomposition – and indeed will adapt to the "plug and plan" addition of new agents.

Unlike other existing work that only focuses on planning in decentralized systems, our decentralized planning architecture ARCADE incorporates plan generation via HTNs with plan adaptation and critiquing. Closest to our approach is probably the Continuous Planning and Execution Framework (CPEF) (Myers 1999), which responds robustly to arbitrary changes in the world, by combining plan generation, execution monitoring, and repair capabilities. CPEF executes and monitors its plans using PRS (Georgeff and Ingrand 1989), and uses HTN planning from SIPE-2 (Wilkins 1988).

There are conceptual similarities between ARCADE and previous work on planning in partially-observable domains (Kaelbling, Littman, and Cassandra 1998; Bertoli et al. 2001; 2006; Kuter et al. 2007; Bonet and Geffner 2011). Both approaches deal with high volume of uncertainty in the planning state during planing time. The latter models the uncertainty explicitly, by leaving it in the solution policies when it is more expensive to resolve it. Our approach takes a chance on possibly unexpected outcomes and state conditions by making assumptions during planning time and appreciating the fact that those assumptions that the plans are conditioned upon can be violated during execution. To reduce uncertainty, ARCADE then uses both (1) planning-time plan critiquing capabilities (Goldman, Kuter, and Schneider 2012; Mueller et al. 2017) to foresee and avoid such failures and (2) rapid HTN plan repair algorithms (Goldman and Kuter 2018b).

Conclusions

We have described our ongoing work on decentralized planning and coordination. The basis of our approach is HTN planning domain definitions employed by multiple HTN planners, i.e., in this case SHOP2, that may be assigned to different tasks or may work on the same tasks in parallel. We are currently finalizing our formalism and conducting more experiments. We plan to investigate the performance and stability of ARCADE under varying conditions of decentralization and disturbance. We will include sensitivity analyses, varying the degree of decentralization of our problems, and assessing the capability (problems successfully solved) and the stability of our assumption-based planning approach. We will also vary the rate of perturbations (external changes to the world state, addition and deletion of new tasks) to assess the stability/volatility of our assumption-based planning approach. We will explore ways to conduct comparisons with existing multi-agent planning systems, MA-STRIPS in particular.

We would like to extend ARCADE to cover task networks that are not totally-ordered. Although planning algorithms and formalisms that can model partially-ordered HTNs exist (e.g., UMCP (Erol, Hendler, and Nau 1994), SHOP2 (Nau et al. 2003), PANDA (Bercher et al. 2017), and FAPE (Dvorak et al. 2014)), we have limited ourselves to TSTN planning as the basis of our formalism to simplify the criteria for correctness for the assumption-based plans.

Another future research direction is to incorporate temporal reasoning in assumption management and coordination. There are two aspects we plan to study: (1) the lifetime of the assumptions themselves: e.g., "if an agent does not hear back about an assumed condition in t time units since the assumption was made, it will cease to accept the assumed truth value"; and (2) temporal bounds on the period over which the assumptions are supposed to hold: e.g., A_i assumes that A_j is going to perform a particular action sometime between the time points t_1 and t_2 ($t_2 > t_1$); A_i must regard this assumption as violated if it is not confirmed by t_2."

We will investigate both directions by borrowing the concept of information volatility from our previous work on Semantic Web Service Composition planning (Au, Kuter, and Nau 2005; Kuter et al. 2005). In this approach, temporal assumptions will model temporal uncertainty on a SHOP2 instance's assumptions made over the tasks and plans over other planner instances in the framework. Another possible approach, perhaps complementing the first one, is to probabilistically assess the belief that a SHOP2 instance has in the assumptions regarding another planner will fulfill its commitments to its tasks and plans.

Acknowledgments. The work reported in this paper project is sponsored by the Air Force Research Laboratory (AFRL) under contract FA8750-16-C-0182 for the Distributed Operations program. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not reflect the views of the AFRL. Cleared for public release, no restrictions.

Thanks to the anonymous referees for many helpful suggestions that has substantially improved our original manuscript.

References

Au, T.-C.; Kuter, U.; and Nau, D. S. 2005. Web service composition with volatile information. In *ISWC*.

Bercher, P.; Biundo, S.; Geier, T.; Hoernle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain-how planning helps to assemble your home theater. In *ICAPS*.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An admissible HTN planning heuristic. In Sierra, C., ed., *IJCAI*.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid planning heuristics based on task decomposition graphs. In *Seventh Annual Symposium on Combinatorial Search*.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2006. Strong Planning under Partial Observability. *Artificial Intelligence* 170:337–384.

Bonet, B., and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI*.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*.

Clement, B. J., and Barrett, A. C. 2003. Continual coordination through shared activities. In *AAMAS*. ACM Press.

Dix, J.; Muñoz-Avila, H.; Nau, D. S.; and Zhang, L. 2003. IMPACTing SHOP: Putting an AI planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence* 37(4):381–407.

Dvorak, F.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Plan-Space Hierarchical Planning with the Action Notation Modeling Language. In *IEEE ICTAI*.

Elkawkagy, M., and Biundo, S. 2011. Hybrid multi-agent planning. In *German Conference on Multiagent System Technologies*, 16–28. Springer.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2010. Exploiting landmarks for hybrid planning. In *25th PuK Workshop Planen, Scheduling und Konfigurieren, Entwerfen*.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *AAAI*.

Fdez-Olivares, J.; Castillo, L.; Garcia-Perez, O.; and Palao, F. 2006. Bringing users and planning technology together, experiences in SIADEX. In *ICAPS*.

Gancet, J.; Hattenberger, G.; Alami, R.; and Lacroix, S. 2005. Task planning and control for a multi-uav system: architecture and algorithms. In *IEEE IROS*.

Genesereth, M. R., and Nilsson, N. J. 1987. *Logical foundations of Artificial Intelligence*. Springer.

Georgeff, M., and Ingrand, F. 1989. Decision-making in an embedded reasoning system. In *IJCAI*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Goldman, R. P., and Kuter, U. 2018a. Explicit stack search in SHOP2. Technical Report 2018-1, SIFT, LLC, Minneapolis, MN, USA.

Goldman, R. P., and Kuter, U. 2018b. Minimal perturbation plan repair for state-space HTN planning. Technical Report 2018-2, SIFT, LLC, Minneapolis, MN, USA.

Goldman, R. P.; Kuter, U.; and Schneider, A. 2012. Using classical planners for plan verification and counterexample generation. In *AAAI Workshop on Problem Solving Using Classical Planning*.

Goldman, R. P. 2006. Durative planning in HTNs. In *ICAPS*.

Grosz, B. J., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

Helmert, M.; Do, M.; and Refanidis, I. 2010. Webpage for IPC-08. Retrieved most recently May 2018.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1-2):99–134.

Kuter, U.; Sirin, E.; Parsia, B.; Nau, D.; and Hendler, J. 2005. Information gathering during planning for web service composition. *Journal of Web Semantics*.

Kuter, U.; Nau, D. S.; Reisner, E.; and Goldman, R. 2007. Conditionalization: Adapting forward-chaining planners to partially observable environments. In *ICAPS 07 Workshop on Planning and Execution for Real-World Systems*.

Kuter, U.; Nau, D.; Pistore, M.; and Traverso, P. 2009. Task Decomposition on Abstract States for Planning under Nondeterminism. *Artificial Intelligence* 173:669–675.

Mueller, J. B.; Miller, C. A.; Kuter, U.; Rye, J.; and Hamell, J. 2017. A human-system interface with contingency planning for collaborative operations of unmanned aerial vehicles. In *AIAA Information Systems-AIAA Infotech@ Aerospace (2017-1296)*. AIAA Press.

Musliner, D.; Goldman, R. P.; Hamell, J.; and Miller, C. 2011. Priority-based playbook tasking for unmanned system teams. In *AIAA*. American Institute of Aeronautics and Astronautics.

Myers, K. L. 1999. A continuous planning and execution framework. *AI Magazine* 63–69.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *JAIR* 20:379–404.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *AAAI*.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *AAMAS*.

Schattenberg, B. 2009. *Hybrid Planning And Scheduling*. Ph.D. Dissertation, Ulm University, Institute of Artificial Intelligence. URN: urn:nbn:de:bsz:289-vts-68953.

Seuken, S., and Zilberstein, S. 2008. Formal models and algorithms for decentralized decision making under uncertainty. In *AAMAS*.

Tate, A.; Drabble, B.; and Kirby, R. 1994. *O-Plan2: An Architecture for Command, Planning and Control*. Morgan-Kaufmann.

Torreño, A.; Onaindia, E.; Komenda, A.; and Štolba, M. 2017. Cooperative multi-agent planning: A survey. *ACM Computing Surveys (CSUR)* 50(6):84.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann.