On representing planning domains under uncertainty

Felipe Meneguzzi
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
meneguzz@cs.cmu.edu

Yuqing Tang
Graduate Center
City University of New York
New York, NY 10016, USA
ytang@cs.gc.cuny.edu

Katia Sycara
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
katia@cs.cmu.edu

Simon Parsons
Brooklyn College,
City University of New York
Brooklyn, NY 11210 USA
parsons@sci.brooklyn.cuny.edu

***Abstract*—Planning is an important activity in military coalitions and the support of an automated planning tool could help military planners by reducing the cognitive burden of their work. Current AI planning paradigms use two different types of formalism to represent the planning problem. Each of these formalisms entails different inference algorithms and representation of results.**

On the one hand plans in non-stochastic domains are represented using declarative logic-based formalisms, an example of which is Hierarchical Task Networks (HTNs). In HTNs, domains are represented in terms of task decompositions of increased detail in relation to the actions that must be carried out. In general, declarative formalisms are easier for humans to understand. On the other hand, stochastic planning is often represented in terms of large probability functions that exhaustively specify the likelihood of relevant world changes when actions are executed, as exemplified by Markov Decision Processes (MDPs). Stochastic domain specifications can easily become challenging to a human designer as the problem size increases, worse still, solver algorithms degrade quickly with increased domain size.

In order to facilitate domain modeling for planning under uncertainty, we propose a method of deriving stochastic domain specifications in the MDP formalism from a description using the HTN formalism. This method can reduce the resulting MDP state-space through an intermediate representation using Binary Decision Diagrams (BDDs). The benefits of the approach are twofold: (a) the reduction of the state space, and consequent reduction of computational burden is beneficial since it enables the representation and solving of realistic planning problems, and (b) starting from a declarative representation makes planning more comprehensible to humans, while extending the representation to stochastic domains.

I. Introduction

Contemporary military operations are "wicked" problems [11], requiring quick response while working with limited resources in domains that are highly dynamic, and in which much information is uncertain [1], [14]. Planning in military coalitions can be even more difficult because of the need to construct plans collaboratively between members of the coalition [2], [11]. Given the difficulty of the problem, several authors have suggested the deployment of AI planners in planning military operations [4], [13], with aim of augmenting, rather than replacing, human planning, and helping to reduce the cognitive burden on the human planners [9].

Existing approaches to planning in artificial intelligence are essentially divided into stochastic and non-stochastic one, each of which uses distinct formalisms to represent problems as well as algorithms to derive solutions to these problems. Non-stochastic planning is represented in terms of operators with preconditions and effects, and planning consists of finding a sequence of operator instances that leads from an initial state to a goal state. Alternatively, stochastic planning is often represented in terms of large probability functions that exhaustively specify the likelihood of relevant world changes when actions are executed, as well as a reward function specifying a preference over certain states in the world. Thus, stochastic planning usually consists of selecting actions that maximize the likelihood that an agent arrives and stays at states with the largest rewards, that is maximizing the expected rewards given the uncertainty. Given the non-deterministic nature of the domain, which means that actions can result in one of a set of states, the solution to a planning problem is not a definite sequence of actions, but rather a *policy* which indicates the optimal action to take in any given world-state.

One particular formalism for domain representation in deterministic planning is the hierarchical task network (HTN), which encodes not only actions with their preconditions and effects, but also domain knowledge in the form of a hierarchy of tasks that can be refined from a high-level objective into the actions required in the environment. HTN planning algorithms have been shown to be very efficient at deriving solutions [6] to complex deterministic problems, and while some stochastic applications of HTNs have been proposed [8], [10], they are not generally used in stochastic domains. Conversely, one of the most widely studied formalisms for planning under uncertainty is the Markov decision process (MDP) [3], in which the evolution of the environment is modelled as a Markov chain, and the goals of the planner are implicitly represented in a reward function that defines, for each state, the reward of executing a certain action.

Deterministic planning domains are generally easy to visualize and understand, as the transitions between states are clearly defined in the operators, and the resulting plans are intuitive and easily understood. In contrast, the definitions of stochastic planning problems problem quickly become unwieldy as the number of states increase. As the number of states goes up, so does the size of the transition probability tables, and there is one such table for each action in the domain. As a consequence, although MDPs are an elegant mathematical formalism for representing stochastic domains, it is not straightforward for non-specialists to model domains using this formalism.

Our work aims to use HTN models, which are more user-friendly, to automatically construct MDPs. In this paper we propose a step towards this overall aim, showing how to use HTNs to describe MDPs, thus allowing stochastic domains to be modelled using HTNs that are then translated into MDPs in order to be solved. A key difference between these formalisms is that they have distinct representations of a world state. Non-stochastic formalisms generally represent a state as a set of first-order logic predicates representing properties of the environment, which are individually modified through the execution of operators, whereas stochastic formalisms use monolithic states that are changed in their entirety upon execution of an action. In effect, states in an MDP are much less descriptive than those in an HTN, and represent a simplification of the state representation used in the HTN formalism. Thus, if a domain description in the HTN formalism is to be converted into an MDP, it is important to devise a method of converting the HTN state-space into an efficient equivalent MDP state-space. We propose using binary decision diagrams (BDDs) as a means to reason about the equivalence of HTN states in order to arrive at the minimal equivalent MDP set of states. The benefits of the approach are twofold: (a) the reduction of the state space, and consequent reduction of computational burden is beneficial since it enables the representation and solving of realistic planning problems, and (b) starting from a declarative representation makes planning more comprehensible to humans, while extending the representation to stochastic domains.

This paper is organized as follows. Section II reviews the formal background for HTNs and MDPs, and introduces an illustrative example to be used throughout the paper. We proceed to explaining how we establish equivalent problem representations in these formalisms in Section III with a naïve look at the state space. Section IV elaborates on the issue of optimizing the MDP state-space using BDDs. Finally we conclude the paper in Section V.

II. Background

In this section we introduce the main concepts that will be used in the paper, and provide a formal model of these concepts. In order to illustrate our approach in the context of the ITA, we introduce a scenario adapted from [5], describing it in terms of the formalisms introduced in this section, and using it throughout the paper to gradually show how our approach converts from HTNs to MDPs.

A. MDPs

We consider an MDP (adapted from [7]) to be a tuple $\Sigma = (S, A, P)$ where S is a finite set of states, A is a finite set of actions, P is a state-transition system. The state-transition system defines a probability distribution for each state transition. Here, given $\{s, s'\} \in S$ and $a \in A$, $P_a(s'|s)$ denotes the probability of transitioning from state s to state s' when executing action a. The solution of an MDPs is a *policy*, which indicates the best action to take in each state. We will represent a policy as a total function mapping states into actions, so a policy π is represented as a function $\pi : S \rightarrow A$. Goal states in MDP are generally represented indirectly through *utility functions*, which typically assign a value $u(a_j, s_i)$ to the choices of actions a_j in states s_i. Such information makes it possible to compute the value of a a given state under a particular policy — it is the expected value of carrying out the policy from that state:

$$V^*(s) = \max_{a \in A(s)} [u(a, s) + \sum_{s' \in S} \Pr_a(s'|s) V^*(s')]$$

The *optimal* policy $\pi^*(s)$ is then the policy that maximises this value:

$$\pi^*(s) = \arg\max_{a \in A(s)} [u(a, s) + \sum_{s' \in S} \Pr_a(s'|s) V^*(s')]$$

and we can find such a policy by various means. One is through *value iteration*, solving:

$$V_{i+1}(s) = \max_{a \in A(s)} [u(a, s) + \sum_{s' \in S} \Pr_a(s'|s) V_i(s')]]$$

and then identifying the utility maximising action at each state.

B. HTNs

We consider an HTN planning domain to be a pair $\mathcal{D} = (\mathcal{A}, \mathcal{M})$ where $\mathcal{A}$ is a finite set of actions (or operators) and $\mathcal{M}$ to be a finite set of methods, while we use the HTN definition from [8] (actually an STN from [7]) whereby an HTN is a pair $\mathcal{H} = (T, C)$ where T is a finite set of tasks to be accomplished and C is a set of partial ordering constraints on tasks in T that, taken together make T totally ordered. The set of tasks contains primitive and non-primitive tasks. All tasks have *preconditions*, specifying a state that must be valid before the task can be carried out, and primitive tasks (*i.e.* actions that are executed in the environment) have *effects*, specifying changes in the state that was valid before the action was executed. For the purposes of this work, we consider preconditions and effects to be propositions p_i, so each task t_i has a set of preconditions $precond(t_i) = \{p_1, \ldots, p_n\}$, and each non-primitive task/action has a set of effects $effects(t_i) = \{p_1, \ldots, p_m\}$.

In the planning literature effects are generally represented as add and delete lists[1], here we consider that effects are specified as a list of positive and negated propositions, and that if the positive version of a proposition was present in the state prior to the execution of an action with a negated effect, that proposition is removed from the resulting state in order to preserve consistency, while if a negated proposition is present in a state and its positive version is present in the effects of an action then that proposition becomes true in the resulting state. For example, if a state $s_i = \{p_1, \neg p_2, p_3\}$ is valid before an action a_i with $effects(a_i) = \{\neg p_1, p_2\}$ is executed, the state resulting from the execution of a_i is $s_{i+1} = \{\neg p_1, p_2, p_3\}$.

[1]An add list specifies the facts that become true as a result of the action — the things that must be added to the description of the state as a result of the action — and the delete list represents the facts that become false as a result of the action — the things that must be deleted from the description of the state given that such descriptions typically only include things that are true of the state.

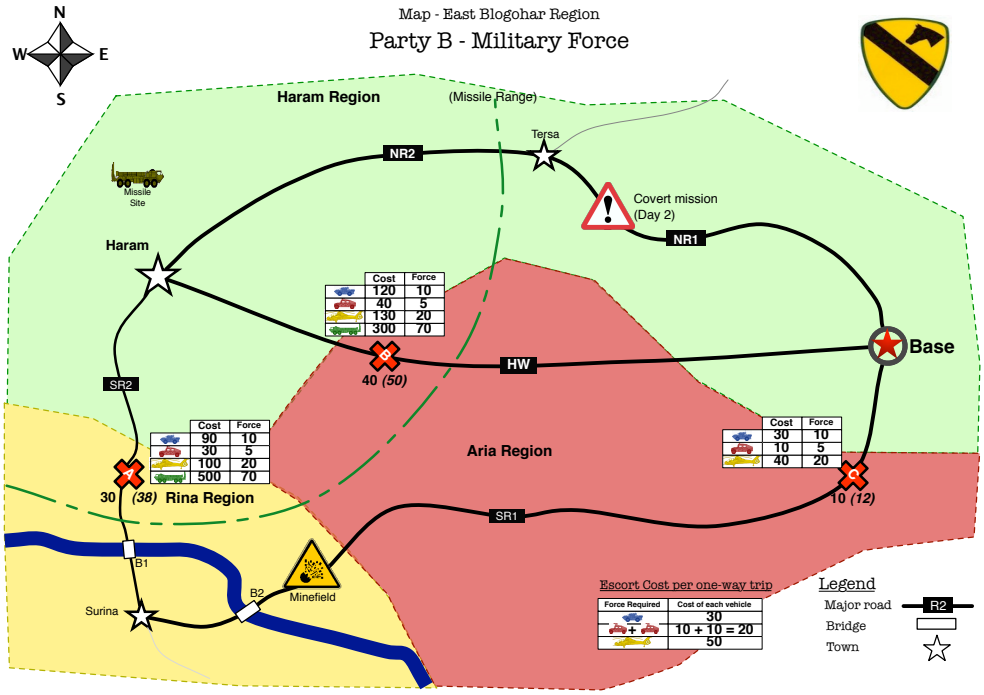

Fig. 1. The map of Party B from the Blogohar scenario.

Primitive tasks are action instances from $a \in \mathcal{A}$, while non-primitive tasks denote tasks that can be decomposed using an appropriate method $m \in \mathcal{M}$. A method describes how a non-primitive task can be decomposed into subtasks. We represent methods as tuples $m = (s, t, T', C')$, where s is a *precondition*, also denoted by $precond(m)$, specifying what must hold in the current state for a task t to be decomposed into new tasks $t_i \in T'$ with constraints $c_j \in C'$. Constraints specify the *order* in which certain tasks must be executed and are represented by the *precedes* relation, where $t_i \prec t_j$ means that task t_i must be executed *before* t_j. Conversely, the *succeeds* relation represents the opposite ordering, where $t_i \succ t_j$ means that task t_i must be executed *after* t_j. When a method is applied to decompose a certain task t_u within an HTN $\mathcal{H} = (T, C)$, a new HTN $\mathcal{H}' = (T'', C'')$ is generated where $T'' = (T - \{t_u\}) \cup T'$ and $C'' = (C - C_{t_u}) \cup (C'_{t_u} \cup C')$. Here C_{t_u} is the set of constraints containing the decomposed task t_u (*i.e.* $\{c \in C | c = t_u \prec t_i \vee c = t_u \succ t_i\}$), and C'_{t_u} is a new set of constraints created by replicating each previously removed constraint (*i.e.* the constraints of C_{t_u}) with each element of T' replacing t_u. For example if HTN $\mathcal{H}$ contains a task t_u and a task t_i and one constraint including t_u, $t_u \prec t_i$, and a substitution $m = (\top, t_u, \{a_1, a_2\}, \{a_1 \prec a_2\})$ is applied, the resulting set of constraints will be $\{a_1 \prec t_i, a_2 \prec t_i, a_1 \prec a_2\}$. In this initial approach, we assume that direct or indirect recursion is not possible within the methods in $\mathcal{M}$, that is, a fully decomposed HTN should be a tree structure.

All non-primitive tasks must be fully decomposed into primitive tasks before a full plan can be derived from the HTN, *i.e.* it must contain no non-primitive tasks.

C. The Blogohar Scenario

We adapt the "Blogohar" scenario from Burnett *et al.*[5], which involves planning tasks for two human players trying to accomplish various objectives within the same region while optimizing the usage of limited resources while avoiding negative interference between the planners. One of the planners (party *A*) controls an international aid agency whose objective consists of evacuating injured civilians from the towns, whereas the other planner is the commander of a military force (party *B*) whose objective consists of neutralizing insurgent strongholds within the regions, maximizing the number of captured insurgents. In the original scenario, both players collaborate and share information to decide on mutual commitments in order to generate plans to deploy their resources and accomplish their individual objectives. We however, focus on modeling the planning activities of party B in terms of an HTN that is to be converted into an MDP, allowing an MDP solver to devise an optimal policy that maximizes captured insurgents while minimizing resource usage.

The resources available to party B consists of a limited number of combat vehicles of various types that can be employed to attack the three insurgent strongholds in the area, illustrated in Figure 1. Each stronghold is garrisoned by a certain number of insurgents representing the maximum number of prisoners that can be taken when that stronghold is overrun as well as the minimum necessary *force* required to capture the stronghold. Employing the minimum amount of force in an attack entails capturing a minimum number of

insurgents, so that although it might be cheaper to employ exactly the required force in any given attack, this results in fewer captured opponents. Each vehicle type provides a certain amount of force to an attack, so the task of a planner is to concentrate enough force to capture each stronghold while maximizing the overall number of captured insurgents. The region has three towns: Haram, Surina and Tersa; as well as three insurgent strongholds: A, B and C and are connected by the roads shown in Figure 1. In order to reach each stronghold, vehicles (which start at the base to the east of the map) must be moved through the roads in the region making sure that strongholds along the selected routes have been neutralized before the road can be fully used.

We now proceed to encoding a subset of the domain regarding party B using the HTN formalism. Following Section II-B, we need to define actions and methods for an HTN domain $\mathcal{D} = (\mathcal{A}, \mathcal{M})$. In our model, party B has two available actions: one action $a^{a(V,T)}$ to attack a location T using a certain vehicle V, and another action $a^{mv(V,F,T,R)}$ to move a vehicle V from a location F to a location T through road R. Thus, the set of actions available to party B is the following:

$$\mathcal{A} = \{a^{a(V,T)}, a^{mv(V,F,T,R)}\}$$

Now, domain knowledge that helps the planner to decide the sequence of actions to take to accomplish a goal is specified in terms of methods, which for our domain consists of the following set:

$$\mathcal{M} = \{m^{DI(A)}, m^{AHu(T)}, m^{AA(T)}, m_1^{Mv(V,T)}, m_2^{Mv(V,T)}\}$$

Where the meaning of each method is as follows:

- $m^{DI(A)}$ is the method to defeat insurgents in location A
- $m^{AHu(T)}$ is the method to attack T with a Humvee
- $m^{AA(T)}$ is the method to attack T with an APC
- $m_n^{Mv(V,T)}$ are the various method to move a vehicle V to T

Recall that methods are used to decompose tasks in an HTN, so we need to specify the set of tasks within the domain. Throughout this paper we adopt the convention that each concrete action has a corresponding (primitive) task named t_i^X, where X specifies the action in the domain and i is a unique identifier for the task, *e.g.* task that consists of executing action $a^{a(V,T)}$ is represented as $t_i^{a(V,T)}$. Moreover, non-primitive tasks are named after the methods that can decompose them, *e.g.* a task that can be decomposed using method $m^{DI(A)}$ is represented as $t_i^{DI(A)}$. Now, using this convention we define the specific components of each method as follows:

- $m^{DI(A)} = (A = a, t^{DI(a)},$
  $\{t^{AHu(a)}, t^{AA(a)}, t^{AHe(a)}\},$
  $\{t^{AHu(a)} \prec t^{AA(a)}, t^{AA(a)} \prec t^{AHe(a)}\})$
- $m^{AHu(T)} = (vehicle(humvee, V) \wedge \neg committed(V),$
  $t^{AHu(T)},$
  $\{t^{Mv(V,T)}, t^{a(V,T)}\},$
  $\{t^{Mv(V,T)} \prec_1 t^{a(V,T)}\})$
- $m^{AA(T)} = (vehicle(apc, V) \wedge \neg committed(V), t^{AA(T)},$
  $\{t^{Mv(V,T)}, t^{a(V,T)}\},$
  $\{t^{Mv(V,T)} \prec t^{a(V,T)}\})$

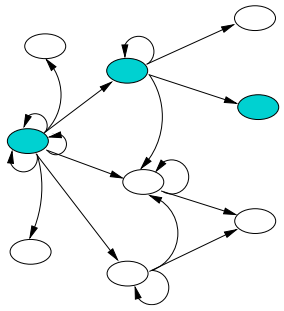

Fig. 2. A graphical representation of an MDP. The colored nodes and the links between them represent a trajectory through the state space.

- $m_1^{Mv(V,T)} = (T = a, t^{Mv(V,T)},$
  $\{t^{mv(V,base,tersa,nr1)}, t^{mv(V,tersa,haram,nr2)},$
  $t^{mv(V,haram,a,sr2)}\},$
  $\{t^{mv(V,base,tersa,nr1)} \prec t^{mv(V,tersa,haram,nr2)},$
  $t^{mv(V,tersa,haram,nr2)} \prec t^{mv(V,haram,a,sr2)}\})$
- $m_2^{Mv(V,T)} = (T = a, t^{Mv(V,T)},$
  $\{t^{mv(V,base,haram,hw)}, t^{mv(V,haram,a,sr2)}\},$
  $\{t^{mv(V,base,haram,hw1)} \prec t^{mv(V,tersa,haram,nr2)}\})$

Finally, we need to define an initial HTN representing the planning goal, which in our scenario consists of defeating the insurgents at stronghold a, which is represented as follows:

$$\mathcal{H} = (defeatInsurgents(a), \emptyset)$$

III. Using HTNs to represent MDPs

If we have an HTN and an MDP that represent the same domain, using the same set of actions and the same set of states, it is clear that there must be some relationship between them. They both, after all, capture the same information. The MDP model of a domain can be visualised as a directed hypergraph (see Figure 2) where the nodes are states and the edges that connect states represent actions — where an arc connects one state to many other states, that action can lead to those other states. As Simari [12] points out, a trajectory through the state-space, for example that tracing the colored nodes in Figure 2, is the result of a single outcome of a specific action in each state, and is close to the notion of a plan in an HTN. This is the intuition from which we begin.

In particular, from an HTN domain $\mathcal{D}$ and an HTN $\mathcal{H}$, it is possible to induce a directed graph similar to the execution structure Σ_π from [8] by considering all possible decompositions of each non-primitive task, and get a structure to which the technique similar to that of Simari [12] can be applied to model MDP states. Here, each primitive task t_i in the HTN associated with an action $a \in \mathcal{A}$ will map to a state representing the world state achieved immediately after the execution of a. However, unlike the execution structure in [8], the state transitions will be determined via the ordering constraints in the HTN, so that there will be a non-zero probability of transitioning from a state t_i to a state t_j in the MDP if and only if it can be determined that t_i immediately precedes t_j in any potential plan generated by the MDP.

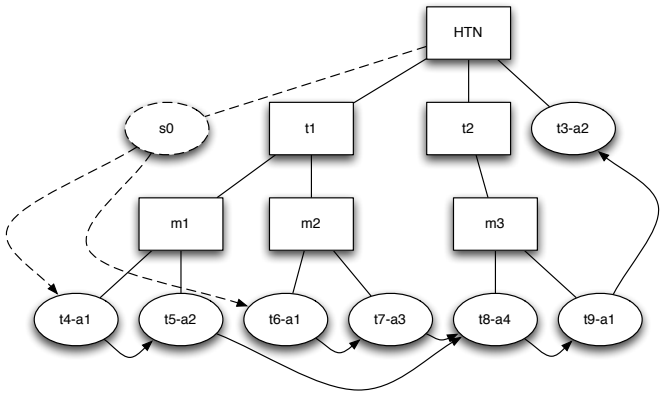

Fig. 3. Fully decomposed HTN, rectangles denote non-primitive nodes while ellipses denote primitive nodes. Dashed elements are not present in the HTN but are added for the MDP.

We represent the immediate sequential precedence relation as $t_i \prec_1 t_j$, and the complementary relation as $t_j \succ_1 t_i$. These immediate relations can be inferred from the arbitrary relations imposed by the HTN methods from the previous section. We shall use these relations in our algorithm to derive the transition functions for each action in the MDP.

Thus, in the propositional case the MDP resulting from an HTN $\mathcal{H}$ has the same number of states as non-primitive task nodes in the fully decomposed HTN plus one state representing the initial state. In order to create a fully decomposed HTN (containing all possible method decompositions), instead of choosing one method to apply to each non-primitive task, we apply all applicable methods and collect all constraints resulting from all possible method applications. Graphically this is equivalent to an and/or tree where non-primitive tasks become branch nodes and primitive tasks become leaves [10]. Each method applicable to a non-primitive task becomes an *or* child of the non-primitive task, whereas the tasks that would replace the non-primitive task become *and* children of each applicable method. We later extend this mapping for the first-order case and take into consideration variable assignments for the description of the HTN.

The collection of constraints resulting from the full decomposition of the original HTN can then be used to construct the transition functions for each action. We represent a fully decomposed HTN as $\mathcal{H}* = (T*, C*)$, and consider that each task $t_i \in T*$ (*i.e.* the set of non-primitive tasks in $\mathcal{H}*$) is labelled with an action $a \in \mathcal{A}$, and we also consider that we can make "inferences" on the sequential constraints in $C*$ so that we can check if two tasks are related through $\prec_1$.

As we have seen, each primitive task in the fully decomposed HTN is considered to represent the state achieved immediately after executing the action associated with it. In consequence, the resulting transition tables uniformly distribute the probability of transitioning from all possible immediate preceding states into the state currently under consideration. For example, consider an HTN $\mathcal{H}_1 = (\{t1, t2, t3\}, \{t1 \prec t2, t2 \prec t3\})$, with a domain $\mathcal{D} = (\{a1, a2, a3, a4\}, \{m1, m2, m3\})$, in which:

- $m1 = (\top, t1, \{a1, a2\}, \{a1 \prec a2\})$
- $m2 = (\top, t1, \{a1, a3\}, \{a1 \prec a3\})$
- $m3 = (\top, t2, \{a4, a1\}, \{a4 \prec a1\})$
- $t1$ and $t2$ are non-primitive tasks;
- $t3$ is a primitive task to execute $a2$

The full decomposition of $\mathcal{H}_1$ is shown in Figure 3, with arrows denoting the transitions that are derived through the algorithm. The resulting MDP has eight states: the seven primitive tasks $t3$, $t4$, $t5$, $t6$, $t7$, $t8$ and $t9$, plus the initial state $s0$. Furthermore the transition function for action $a1$ would have 0.5 probability of transitioning from $s0$ to either $t4$ or $t6$, and probability 1 of transitioning from $t8$ to $t9$; while $a2$ would have probability 1 for transitioning from $t4$ to $t5$, as well as for $t9$ to $t3$.

Returning to the scenario introduced in Section II-C its initial HTN $\mathcal{H} = (defeatInsurgents(a), \emptyset)$ can be fully decomposed as $\mathcal{H}*$, shown below:

$$
\begin{aligned}
\mathcal{H}* = (\{ & \\
& t_0, t_1^{mv(humvee1,base,tersa,nr1)}, \\
& t_2^{mv(humvee1,tersa,haram,nr2)}, t_3^{mv(humvee1,haram,a,sr2)}, \\
& t_4^{a(humvee1,a)}, t_5^{mv(humvee1,base,haram,hw)}, \\
& t_6^{mv(humvee1,tersa,haram,nr2)}, t_7^{a(humvee1,a)}, \\
& t_8^{mv(apc1,base,tersa,nr1)}, t_9^{mv(apc1,tersa,haram,nr2)}, \\
& t_{10}^{mv(apc1,haram,a,sr2)}, t_{11}^{a(apc1,a)} \\
& t_{12}^{mv(humvee1,base,haram,hw)}, \\
& t_{13}^{mv(humvee1,tersa,haram,nr2)}, t_{14}^{a(humvee1,a)} \\
\}, & \\
\{ & t_0 \prec_1 t_1, t_0 \prec_1 t_5, \\
& t_1 \prec_1 t_2, t_2 \prec_1 t_3, t_3 \prec_1 t_4, \\
& t_5 \prec_1 t_6, t_6 \prec_1 t_7, \\
& t_4 \prec_1 t_8, t_7 \prec_1 t_8, \\
& t_4 \prec_1 t_{12}, t_7 \prec_1 t_{12}, \\
& t_8 \prec_1 t_9, t_9 \prec_1 t_{10}, t_{10} \prec_1 t_{11}, \\
& t_{12} \prec_1 t_{13}, t_{13} \prec_1 t_{14}, \\
\}) &
\end{aligned}
$$

Given the fully expanded state-space represented by $\mathcal{H}*$, the technique we propose would initially generate an MDP with 15 states (t_0 through to t_{14}) and two actions. Each action in the resulting MDP requires a distinct transition matrix, denoting the probability of moving from one state to an adjacent one when executing a certain action. As an example, we show the transition matrix for action $a^{mv(V,F,T,R)}$ in Table I, generated from the precedence constraints $C*$ of $\mathcal{H}*$.

Notice that the probability of transitioning using $a^{mv(V,F,T,R)}$ is zero between states that are not reachable by move actions, for example t_{13} to t_{14}. Furthermore, transition probabilities are uniformly distributed among the destination states when different HTN expansions create

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
t_0	0	0.5	0	0	0	0.5	0	0	0	0	0	0	0	0	0
t_1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
t_2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
t_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_4	0	0	0	0	0	0	0	0	0.5	0	0	0	0.5	0	0
t_5	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
t_6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
t_7	0	0	0	0	0	0	0	0	0.5	0	0	0	0.5	0	0
t_8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
t_9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
t_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
t_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE I
TRANSITION MATRIX CREATED FROM THE FULLY EXPANDED HTN $\mathcal{H}*$ FOR ACTION $a^{mv(V,F,T,R)}$.

multiple precedence constraints starting from the same state, as is the case for the transitions between t_4 to t_8 and t_4 to t_{12}, effectively representing the choice between moving either a Humvee or an APC out of the base soon after the first attack on stronghold A.

IV. INCREASING EFFICIENCY

Up to this point, the mapping of HTN states to MDP states has been pretty much equivalent to the mapping defined by [12], with the number of MDP states being proportional to the number of all possible primitive tasks in the HTN. This proportion is not useful if we want the solution for the resulting MDPs to be scalable. However, recall that each non-primitive task has a set of preconditions and a set of effects, and the sequential execution of primitive actions will lead to a history of states. In the case of a fully decomposed HTN, each task in the HTN represents a set of states that could possibly be true given the previous history of executed primitive tasks, which is dependent on the methods actually chosen to decompose non-primitive tasks (or, in the case of [8], given the probabilities of success for the primitive tasks chosen for execution).

Using a technique similar to Kuter's [8], it is possible to generate BDDs representing sets of fine-grained HTN states (*i.e.* propositional formulae) possible at each primitive task node in the expanded HTN, and reduce the set of coarse states (*i.e.* whatever is true after an action) in the resulting MDP by comparing the BDDs (representing what is true) at the states to be translated into the MDP. In the worst case scenario, the number of states that result by aggregating similar states using the corresponding BDD comparisons will be the same as for Simari's approach (*i.e.* one per HTN primitive node), while depending on how much overlap exists between possible states, this number can be much reduced. As an example, the states represented by t_4 and t_6 in Figure 3 are clearly equal, as they both represent the changes that occur immediately after executing a_1 from the initial state s_0. Other equivalences might also occur, and BDDs seem to be a suitable data structure to detect this efficiently.

In many scenarios, there can be many methods to achieve the same effects. Correspondingly, there can be many decompositions to achieve the same task. While there can be many ways to achieve the same task, their execution paths can contain very similar segments. As we can see in the example introduced in Section II-C, t_4 and t_7 correspond to the situation such that $humvee1$ attacks point a, and then from t_4 and t_7 on, no matter which methods the agent will take, the two sets of possible execution paths are the same, since all possible plans generated afterwards are equal.

After the concept of fully decomposed HTN has been introduced above, the equivalency of states can be detected from two points of view with respect to solving the MDP:

- Directly explore the execution structure of the resulting MDP, and detect whether two states have the same set of consequent execution paths
- With utility introduced, explore the execution structure of the resulting MDP, and detect whether two states have the same set of consequent execution paths with the same expected discounted utilities

These two views can be used directly with appropriate data structure and dynamic programming techniques to aggregate the states in the resulting MDP so that we can reduce the size of the MDP.

In addition, we can utilize the preconditions defined for tasks and methods. Intuitively, if in a fully decomposed HTN $\mathcal{H}^*$ the refined preconditions of two primitive tasks t_1 and t_2 are equivalent, and all the possible execution paths after these two tasks are the same, then we can aggregate the two primitive tasks into one in the resulting MDP. We can define two states which have the same set of all the possible subsequent executions as *equivalent*. Therefore, we can investigate how the preconditions of tasks and methods relate to above two views of equivalent MDPs. If two states are equivalent whenever their preconditions are equivalent, we can decide which states to merge by detecting equivalent preconditions of these states. With BDD representing preconditions, the comparison can be done in constant time $O(1)$ time. The size of the BDDs will depend on the amount of information embedded into the preconditions of these states. In the worst

case, we can at least have the system aggregate the states with size of BDDs below certain level. In the general case, we can try to associate BDDs to all the states. We are working on a implementation that will allow us to explore the performance of these approaches.

Another approach is to use non-primitive tasks as states in the resulting MDP. The challenge is how to determine which non-primitive tasks precede which other non-primitive or primitive tasks, and how to assign the corresponding transition probabilities and utilities. Again, we propose to explore the execution structure in the fully decomposed HTN. From the execution structure of the fully decomposed HTN, we will be able obtain the marginal probabilities of all the execution paths that can be decomposed from the non-primitive task. In a similar way, we can also obtain the expected discounted utilities of all the execution paths that can be decomposed from the non-primitive task. In this way, we will be able to obtain different MDPs for the same root task but with different level of abstractions in tasks while all these MDPs actually respect the same probability and utility model. Therefore, with these MDPs we can solve the same decision theoretic problem at any level of abstraction where the more abstract solution requires (possibly exponentially) fewer states, state transitions and utitilies. In combination with the precondition based HTN state merging approach, we might be able to compute the MDP probability associated with the non-primitive tasks without the need to unfold all the possible sequences of primitive tasks.

V. Conclusions and future work

Planning is an important activity in military coalitions and the support of an automated planning tool could help military planners by reducing the cognitive burden of their work. Current approaches to AI planning use two different types of formalism, one suitable for deterministic planning, the other suitable for non-deterministic planning. The complexity of military planning is best captured by non-deterministic planning, but the available representations are cumbersome and hard to use — adopting them may therefore increase rather than decrease the burden on the planners. However, as we describe here, it is possible to describe the planning problem as if it were a deterministic problem, and then convert the representation into a non-deterministic one. In this paper we have shown how this conversion may be carried out, and have discussed some approaches for dealing with the explosion in the size of the representation created by the naive conversion. Naive conversion has been implemented and found to be adequate for small problems where there is not much uncertainty in the environment.

While we focused on establishing the correspondence between the formal structures of HTNs and MDPs, the techniques that perform the actual conversion between problem specifications are initial efforts with inherent limitations. The main limitation is that the MDP probabilities automatically generated by our naive conversion refer mainly to the uncertainty in the planning process. Nevertheless, uncertainty arising from the world needs to be accounted for, and we envision this information being supplied with the input. Subjective probabilities can be annotated in the HTN methods and then used to calculate state transition probabilities in the resulting MDP. Our current work aims to automate the conversion process and to investigate the effectiveness of the state-space reduction techniques described in this paper.

Acknowledgement. This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] D. Aberdeen, S. Thiébaux, and L. Zhang. Decision-theoretic military operations planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, pages 402–411, Whistler, British Columbia, 2004.

[2] A. Bahrami, J. Yuan, D. Mott, and C. D. Emele. Collaborative, context-aware and chain of command sensitive planning. In *Proceedings of the Second Annual Conference of the International Technology Alliance*, London, September 2008.

[3] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[4] M. A Bienkowski and L. J. Hoebel. Integrating AI components for a military planning operation. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 1998.

[5] Christopher Burnett, Daniele Masato, Mairi McCallum, Timothy J. Norman, Joseph Andrew Giampapa, Martin Kollingbaum, and Katia Sycara. Agent support for mission planning under policy constraints. In *Proceedings of the Second Annual Conference of the International Technology Alliance*, London, UK, 2008.

[6] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1123–1128, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[7] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[8] Ugur Kuter, Dana Nau, Marco Pistore, and Paolo Traverso. Task decomposition on abstract states, for planning under nondeterminism. *Artificial Intelligence*, 173(5-6):669 – 695, 2009. Advances in Automated Plan Generation.

[9] T. L. Lenox, T. R. Payne, S. Hahn, M. Lewis, and K. Sycara. Agent-based aiding for individual and team planning tasks. In *Proceedings of IEA 2000/HFES 2000 Congress*, San Diego, July 2000.

[10] Luís Macedo and Amílcar Cardoso. Case-based, decision-theoretic, HTN planning. In Peter Funk and Pedro A. González-Calero, editors, *7th European Conference on Advances in Case-Based Reasoning*, volume 3155 of *LNCS*, pages 257–271. Springer, 2004.

[11] T. J. McKearney. Colaborative planning for military operations: Emerging technologies and changing command organizations. In *Proceedings of the Command and Control Research and Technology Symposium*, Naval Postgraduate School, Monterey, CA, June 2000.

[12] Gerardo I. Simari and Simon Parsons. On the relationship between MDPs and the BDI architecture. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1041–1048, New York, NY, USA, 2006. ACM.

[13] D. E. Wilkins and R. V. Desimone. Applying an AI Planner to Military Operations Planning. Technical Note 534, SRI International, 1993.

[14] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *JOurnal of Experimental and Theoretical Artificial Intelligence*, 7(1):121–152, January 1995.