# Automatic Discovery and Transfer of MAXQ Hierarchies

**Neville Mehta** MEHTANE@EECS.OREGONSTATE.EDU
**Soumya Ray** SRAY@EECS.OREGONSTATE.EDU
**Prasad Tadepalli** TADEPALL@EECS.OREGONSTATE.EDU
**Thomas Dietterich** TGD@EECS.OREGONSTATE.EDU
Oregon State University, Corvallis OR 97331, USA

## Abstract

We present an algorithm, HI-MAT (Hierarchy Induction via Models And Trajectories), that discovers MAXQ task hierarchies by applying dynamic Bayesian network models to a successful trajectory from a source reinforcement learning task. HI-MAT discovers subtasks by analyzing the causal and temporal relationships among the actions in the trajectory. Under appropriate assumptions, HI-MAT induces hierarchies that are consistent with the observed trajectory and have compact value-function tables employing safe state abstractions. We demonstrate empirically that HI-MAT constructs compact hierarchies that are comparable to manually-engineered hierarchies and facilitate significant speedup in learning when transferred to a target task.

## 1. Introduction

Scaling up reinforcement learning (RL) to large domains requires leveraging the structure in these domains. Hierarchical reinforcement learning (HRL) provides mechanisms through which domain structure can be exploited to constrain the value function and policy space of the learner, and hence speed up learning (Sutton et al., 1999; Dietterich, 2000; Andre & Russell, 2002). In the MAXQ framework, a task hierarchy is defined (along with relevant state variables) for representing the value function of the overall task. This allows for decomposed subtask-specific value functions that are easier to learn than the global value function.

Automated discovery of such task hierarchies is compelling for at least two reasons. First, it avoids the significant human effort in engineering the task-subtask structural decomposition, along with the associated state abstractions and subtask goals. Second, if the same hierarchy is useful in multiple domains, it leads to significant transfer of learned structural knowledge from one domain to the other. The cost of learning can be amortized over several domains. Several researchers have focused on the problem of automatically inducing temporally extended actions and task hierarchies (Thrun & Schwartz, 1995; McGovern & Barto, 2001; Menache et al., 2001; Pickett & Barto, 2002; Hengst, 2002; Şimşek & Barto, 2004; Jonsson & Barto, 2006).

In this paper, we focus on the *asymmetric knowledge transfer* setting where we are given access to solved source RL problems. The objective is to derive useful biases from these solutions that could speed up learning in target problems. We present and evaluate our approach, HI-MAT, for learning MAXQ hierarchies from a solved RL problem. HI-MAT applies dynamic Bayesian network (DBN) models to a single successful trajectory from the source problem to construct a *causally annotated trajectory* (CAT). Guided by the causal and temporal associations between actions in the CAT, HI-MAT recursively parses it and defines MAXQ subtasks based on each discovered partition of the CAT.

We analyze our approach both theoretically and empirically. Our theoretical results show that, under appropriate conditions, the task hierarchies induced by HI-MAT are consistent with the observed trajectory, and possess compact value-function tables that are safe with respect to state abstraction. Empirically, we show that (1) using a successful trajectory can result in more compact task decompositions than when using only DBNs, (2) our induced hierarchies are comparable to manually-engineered hierarchies on target RL tasks, and MAXQ-learning converges significantly faster than flat Q-learning on those tasks, and

Appearing in *Proceedings of the 25$^{th}$ International Conference on Machine Learning*, Helsinki, Finland, 2008. 

(3) transferring hierarchical structure from a source task can speed up learning in target RL tasks where transferring value functions cannot.

## 2. Background and Related Work

We briefly review the MAXQ framework (Dietterich, 2000). This framework facilitates learning separate value functions for subtasks which can be composed to compute the value function for the overall semi-Markov Decision Process (SMDP) with state space $\mathcal{S}$ and action space $\mathcal{A}$. The task hierarchy $\mathcal{H}$ is represented as a directed acyclic graph called the *task graph*, and reflects the task-subtask relationships. Leaf nodes are the primitive subtasks corresponding to $\mathcal{A}$. Each composite subtask $T_i$ defines an SMDP with parameters $\langle X_i, S_i, G_i, C_i \rangle$, where $X_i$ is the set of relevant state variables, $S_i \subseteq \mathcal{S}$ is the set of admissible states, $G_i$ is the termination/goal predicate, and $C_i$ is the set of child tasks of $T_i$. $T_0$ represents the root task. $T_i$ can be invoked in any state $s \in S_i$, it terminates when $s' \in G_i$, and $(s, a)$ is called an *exit* if $\Pr(s'|s, a) > 0$.

The set $S_i$ is defined using a projection function that maps a world state to an abstract state defined by a subset of the state variables. A *safe* abstraction function only merges world states that have identical values. The local policy for a subtask $T_i$ is a mapping $\pi_i : S_i \mapsto C_i$. A hierarchical policy $\pi$ for the overall task is an assignment of a local policy to each $T_i$. A *hierarchically optimal policy* for a given MAXQ graph is a hierarchical policy that has the best possible expected total reward. A hierarchical policy is *recursively optimal* if the local policy for each subtask is optimal given that all its child tasks are in turn recursively optimal.

HEXQ (Hengst, 2002) and VISA (Jonsson & Barto, 2006) are two existing approaches to learning task hierarchies. These methods define subtasks based on the changing values of state variables. HEXQ employs a heuristic that orders state variables based on the frequencies of change in their values to induce an *exit-option* hierarchy. The most frequently-changing variable is associated with the lowest-level subtask, and the least frequently-changing variable with the root. VISA uses DBNs to analyze the influence of state variables on one another. The variables are partitioned such that there is an acyclic influence relationship between the variables in different clusters (strongly-connected components). Here, state variables that influence others are associated with lower-level subtasks. VISA provides a more principled rationale for HEXQ's heuristic – a variable used to satisfy a precondition for setting another variable through an action typically changes more frequently than the other variable. A key difference between VISA and HI-MAT is the use of a successful trajectory in addition to the DBNs. In Section 5.1, we provide empirical evidence that this allows HI-MAT to learn hierarchies that are exponentially more compact than those of VISA.

The algorithm developed by Marthi et al. (2007) takes a search-based approach to generating hierarchies. Flat Q-value functions are learned for the source domain, and are used to sample trajectories. A greedy top-down search is conducted for the best-scoring hierarchy that fits the trajectories. The set of relevant state variables for each task is determined through statistical tests on the Q values of different states with differing values of the variables. In contrast to this approach, HI-MAT relies less on direct search through the hierarchy space, and more on the causal analysis of a trajectory based on DBN models.

## 3. Discovering MAXQ Hierarchies

In this work, we consider MDPs where the agent is solving a known conjunctive goal. This is a subset of the class of stochastic shortest-path MDPs. In such MDPs, there is a goal state (or a set of goal states), and the optimal policy for the agent is to reach such a state as quickly as possible. We assume that we are given factored DBN models for the source MDP where the conditional probability distributions are represented as trees (CPTs). Further, we are given a successful trajectory that reaches the goal in the source MDP. With this in hand, our objective is to automatically induce a MAXQ hierarchy that can suitably constrain the policy space when solving a related target problem, and therefore achieve faster convergence in the target problem. This is achieved via recursive partitioning of the given trajectory into subtasks using a top-down parse guided by backward chaining from the goal. We use the DBNs along with the trajectory to define the termination predicate, the set of subtasks, and the relevant abstraction for each MAXQ subtask.

We use the Taxi domain (Dietterich, 2000) to illustrate our procedure. Here, a taxi has to transport a passenger from a source location to a destination location within a $5 \times 5$ grid-world. The *pass.dest* variable is restricted to one of four special locations on the grid denoted by R, G, B, Y; the *pass.loc* could be set to R, G, B, Y or in-taxi; *taxi.loc* could be one of the 25 cells. The goal of $pass.loc = pass.dest$ is achieved by taking the passenger to its intended destination. Besides the four navigation actions, a successful Pickup changes *pass.loc* to in-taxi, and a successful Putdown changes *pass.loc* from in-taxi to the value of *pass.dest*.

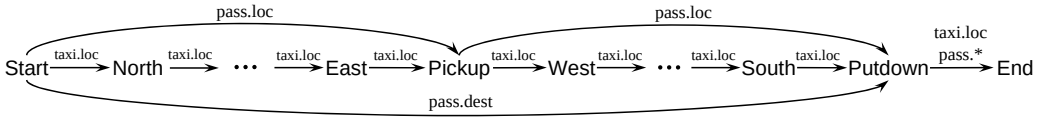



*Figure 1.* A sample CAT for the Taxi domain.

### 3.1. Definitions and Notation

We say that a variable $v$ is *relevant* to an action $a$ if the reward and transition dynamics for $a$ either check or change $v$; it is *irrelevant* otherwise. The set of *trajectory-relevant* (t-relevant) variables of $a$, a subset of the relevant variables, are the variables that were actually checked or changed when $a$ was executed in the trajectory. A *causal* edge $a \xrightarrow{v} b$ connects $a$ to another action $b$ ($b$ following $a$ in the trajectory) iff $v$ is t-relevant to both $a$ and $b$, and irrelevant to all actions in between. A *sink* edge, $a \xrightarrow{v} \mathsf{End}$ connects $a$ with a dummy End action iff $v$ is relevant to $a$ and irrelevant to all actions before the final goal state; this holds analogously for a *source* edge $\mathsf{Start} \xrightarrow{v} a$. A *causally annotated trajectory* (CAT) is the original trajectory annotated with all the causal, source, and sink edges. Moreover, the CAT is preprocessed to remove any cycles present in the original trajectory (failed actions, such as an unsuccessful Pickup, introduce cycles of unit length). A sample CAT for Taxi is shown in Figure 1.

Given $a \xrightarrow{v} b$, the phrase "literal on a causal edge" refers to a formula of the form $v = \mathsf{V}$ where V is the value taken by $v$ in the state before $b$ is executed. We define `DBN-closure(`$v$`)` as the set of variables that influence $v$ recursively as follows. From the action DBNs, add all variables that appear in internal nodes in the CPTs for the dynamics of $v$. Next, for each added variable $u$, union `DBN-closure(`$u$`)` with this set, repeating until no new variables are added. Similarly, the set `DBN-closure(`*reward*`)` contains all variables that influence the reward function of the MDP. The set `DBN-closure(`$fluent$`)` is the union of the `DBN-closure`s of all variables in the fluent. For example, `DBN-closure(`*goal*`)` is the set of all variables that influence the goal fluent. The CAT ignores all variables $v \notin$ `DBN-closure(`*goal*`)`, namely, those variables that never affect the goal conjunction.

### 3.2. The HI-MAT Algorithm

Given a CAT and the MDP's goal predicate (or recursively, the current subtask's goal predicate), the main loop of the hierarchy induction procedure is illustrated in Algorithm 1. The algorithm first checks if two stopping criteria are satisfied (lines 2 & 4): either the trajectory contains only a single primitive action, or it consists of actions whose relevances are identical. (In the latter case, any further partitioning would yield subtasks with the same abstraction as the parent.) Otherwise, it first initializes the set of "unsolved" goals to the set of literals in the goal conjunction (line 9). It then selects any unsolved goal $u$, and finds the corresponding subtask (line 12). Algorithm 2 returns indices $i, j$ marking the boundaries of the subtask in the CAT. If this CAT segment is nontrivial (neither just the initial state nor the whole trajectory), it is stored (line 17), and the literals on causal edges that enter it (from earlier in the trajectory) are added to the unsolved goals (line 18). This ensures that the algorithm parses the entire trajectory barring redundant actions. If the trajectory segment is equal to the entire trajectory, this implies that the trajectory achieves only the literal $u$ after the ultimate action. In this case, the trajectory is split into two segments: one segment contains the prefix of the ultimate action $a_n$ with the preconditions of $a_n$ forming the goal literals for this segment (line 14); the other segment contains only the ultimate action $a_n$ (line 15). CAT scanning is repeated until all subgoal literals are accounted for.

The only way trajectory segments can overlap is if they have identical boundaries, and the ultimate action achieves the literals of all these segments. In this case, the segments are merged (line 23). Merging replaces the duplicative segments with one that is assigned a conjunction of the subgoal literals.

The HI-MAT algorithm partitions the CAT into unique segments, each achieving a single literal or a conjunction of literals due to merging. It is called recursively on each element of the partition (line 27). It can be proved that the set of subtasks output by the algorithm is independent of the order in which the literal $u$ is picked (line 11).

#### 3.2.1. Subtask Detection

Given a literal, a subtask is determined by finding the set of temporally contiguous actions that are closed with respect to the causal edges in the CAT such that the final action achieves the literal. The idea is to group all actions that contribute to achieving the specific literal being considered. This procedure is shown in Algorithm 2.

**Algorithm 1** HI-MAT
Input: CAT $\Omega$, Goal predicate $G$.
Output: Task $\langle X, S, G, C\rangle$; $X$ is the set of relevant variables, $S$ is the set of non-terminal states, $G$ is the goal predicate, $C$ is the set of child actions.

1: $n \leftarrow$ Number of actions in $\Omega$ excluding Start and End
2: **if** $n = 1$ **then** // Single action
3: **return** $\langle \textsc{RelVars}(\Omega), \mathcal{S}, \text{true}, a_1\rangle$
4: **else if** CheckRelVars($\Omega$) **then** // Same relevance
5: $S \leftarrow$ All states that reach $G$ via $\textsc{Actions}(\Omega)$
6: **return** $\langle \textsc{RelVars}(\Omega), S, G, \textsc{Actions}(\Omega)\rangle$
7: **end if**
8: $\Psi \leftarrow \emptyset$ // Trajectory segments
9: $U \leftarrow \textsc{Literals}(G)$
10: **while** $U \neq \emptyset$ **do**
11: Pick $u \in U$
12: $(i, j, u) \leftarrow$ CAT-Scan($\Omega, u$)
13: **if** $i = 1 \wedge j = n$ **then**
14: $\Psi \leftarrow \Psi \cup \{(1, n-1, v) : v \in \textsc{Precondition}(a_n)\}$
15: $\Psi \leftarrow \Psi \cup \{(n, n, \emptyset)\}$
16: **else if** $j > 0$ **then** // Last segment action $\neq$ Start
17: $\Psi \leftarrow \Psi \cup \{(i, j, u)\}$
18: $U \leftarrow U \cup \{v : \exists k < i \; \exists l \; a_k \xrightarrow{v} a_l \in \Omega, i \leq l \leq j\}$
19: **end if**
20: $U \leftarrow U - \{u\}$
21: **end while**
22: **while** $\exists (i, j, u_1), (i, j, u_2) \in \Psi$ **do**
23: $\Psi \leftarrow (\Psi - \{(i, j, u_1), (i, j, u_2)\}) \cup \{(i, j, u_1 \wedge u_2)\}$
24: **end while**
25: $C \leftarrow \emptyset$
26: **for** $t \in \Psi$ **do**
27: $\langle X_t, S_t, G_t, C_t\rangle \leftarrow$ HI-MAT(Extract($\Omega, t_i, t_j), t_u$)
28: $C \leftarrow C \cup \{\langle X_t, S_t, G_t, C_t\rangle\}$
29: **end for**
30: $X \leftarrow \textsc{RelVars}(\Omega) \cup \textsc{Variables}(G)$
31: $S \leftarrow$ All states that reach $G$ via $C$
32: **return** $\langle X, S, G, C\rangle$

**Algorithm 2** CAT-Scan
Input: CAT $\Omega$, literal $u$.
Output: $(i, j, u)$; $i$ is the start index, $j$ is the end index.

1: Set $j$ such that $a_j \xrightarrow{u}$ End $\in \Omega$
2: $i \leftarrow j - 1$
3: **while** $i > 0$ **and** $\forall v \; \exists k \; a_i \xrightarrow{v} a_k \implies k \leq j$ **do**
4: $i \leftarrow i - 1$
5: **end while**
6: **return** $(i + 1, j, u)$

As before, when considering causal edges in line 3, we can ignore all causal edges that are labeled with variables not in the `DBN-closure` of any variable in the current unsolved goal list. Because of the way we construct the CAT, we can show that this procedure will always stop before adding an action which has a relevant variable that is not relevant to the last action in the partition. Note that the temporal contiguity of the actions we assign to a subtask is required by the MAXQ-style execution of a policy. A hierarchical MAXQ policy cannot interrupt an unterminated subtask, start executing a sibling subtask, and then return to executing the interrupted subtask.

#### 3.2.2. Termination Predicate

After finding the partition that constitutes a subtask, we assign a set of child tasks and a termination predicate to it. To assign the termination condition to a subtask, we consider the relational test(s) $t_u$ in the action and reward DBNs involving the variable $u$ on the causal edge leaving the subtask (line 27 of Algorithm 1). When a subtask's relational termination condition involves other variables not already in the abstraction, these variables are added to the state abstraction (line 30), effectively creating a parameterized subtask. For example, consider the navigation subtask that terminates when $taxi.loc = pass.dest$ in the Taxi domain. The abstraction for this subtask already involves $taxi.loc$. However, $pass.dest$ in the relational test implies that $pass.dest$ behaves like a parameter for this subtask.

#### 3.2.3. Action Generalization

To determine if the set of primitive actions available to any subtask should be expanded, we follow a bottom-up procedure (not shown in Algorithm 1). We start with subtasks that have only primitive actions as children. We create a *merged* DBN structure for such a subtask $T$ using the incorporated primitive actions. The merged DBN represents possible variable effects after any sequence of these primitive actions. Next, for each primitive action that we did not see in this trajectory, we consider the subgraph of its DBN that only involves the variables relevant to $T$. If this is a subgraph of the merged DBN of $T$, we add this action to the set of actions available to $T$. The rationale here is that the added action has similar effects to the actions we observed in the trajectory, and it does not increase the set of relevant variables for $T$. For example, if the navigation actions used on the observed trajectory consisted only of North and East actions, this procedure would also add South and West to the available actions for this subtask. When considering subtasks that have non-primitive children, we only consider adding actions that have not been added to any of the non-primitive children.

Given the termination predicate and the generalized set of actions, the set of relevant variables for a subtask is the union of the set of relevant variables of the merged DBN (described above) and the variables appearing in the termination predicate (line 30). Computing the relevant variables is similar to explanation-based reinforcement learning (Tadepalli & Dietterich,

1997) except that here we care only about the set of relevant variables and not their values. Moreover, the relevant variables are computed over a set rather than a sequence of actions.

## 4. Theoretical Analysis

In this section, we establish certain theoretical properties of the hierarchies induced by the HI-MAT algorithm. We consider a factored SMDP state-space $\mathcal{S} = D_{x_1} \times \ldots \times D_{x_k}$, where each $D_{x_i}$ is the domain of variable $x_i$. We assume that our DBN models have the following property.

**Definition 1** *A DBN model is maximally sparse if for any $y \in Y$ where $Y$ is the set of parents of some node $x$ (which represents either a state variable or the reward node), and $Y' = Y - \{y\}$,*

$$\exists y_1, y_2 \in D_y \quad \Pr(x|Y', y = y_1) \neq \Pr(x|Y', y = y_2).$$

Maximal sparseness implies that the parents of a variable have non-trivial influences on it; no parent can be removed without affecting the next-state distribution.

A task hierarchy $\mathcal{H} = \langle V, E \rangle$, is a directed acyclic graph, where $V$ is a set of task nodes, and $E$ represents the task-subtask edges of the graph. Each task node $T_i \in V$ is defined as in Section 2.

A trajectory-task pair $\langle \Omega, T_i \rangle$, where $\Omega = \langle s_1, a_1, \ldots, s_n, a_n, s_{n+1} \rangle$ and $T_i = \langle X_i, S_i, G_i, C_i \rangle$, is consistent with $\mathcal{H}$ if $T_i \in V$, and $\{s_1, \ldots, s_n\} \subseteq S_i$. If $T_i$ is a primitive subtask then $n = 1$, and $C_i = a_1$. If $T_i$ is not primitive then $\{s_1, \ldots, s_n\} \cap G_i = \emptyset$, $s_{n+1} \in G_i$, and there exist trajectory-task pairs $\langle \Omega_j, T_j \rangle$ consistent with $\mathcal{H}$ where $\Omega$ is a concatenation of $\Omega_1, \ldots, \Omega_p$ and $T_1, \ldots, T_p \in C_i$.

A trajectory $\Omega$ is consistent with a hierarchy $\mathcal{H}$ if $\langle \Omega, T_0 \rangle$ is consistent with $\mathcal{H}$.

**Definition 2** *A trajectory $\langle s_1, a_1, \ldots, s_n, a_n, s_{n+1} \rangle$ is non-redundant if no subsequence of the action sequence in the trajectory, $a_1, \ldots, a_n$, can be removed such that the remaining sequence still achieves the goal starting from $s_1$.*

**Theorem 1** *If a trajectory $\Omega$ is non-redundant then HI-MAT produces a task hierarchy $\mathcal{H}$ such that $\Omega$ is consistent with $\mathcal{H}$.*

**Proof sketch:** Let $\Omega = \langle s_1, a_1, \ldots, s_n, a_n, s_{n+1} \rangle$ be the trajectory. The algorithm extracts the conjunction of literals that are true in $s_{n+1}$ (and not before), and assigns it to the goal, $G_i$. Such literals must exist since, otherwise, some suffix of the trajectory can be removed while the rest still achieves the goal, violating the property of non-redundancy. Since the set $S_i$ is set to all states that do not satisfy $G_i$, the condition that all states $s_1, \ldots, s_n$ are in $S_i$ is satisfied.

Whenever the trajectory is partitioned into a sequence of sub-trajectories, each sub-trajectory is associated with a conjunction of goal literals achieved by that sub-trajectory. Hence, the above argument applies recursively to each such sub-trajectory. □

**Definition 3** *A hierarchy $\mathcal{H}$ is safe with respect to the DBN models $M$ if for any trajectory-task pair $\langle \Omega, T_i \rangle$ consistent with $\mathcal{H}$, where $T_i = \langle X_i, S_i, G_i, C_i \rangle$, the total expected reward during the trajectory is only a function of the values of $x \in X_i$ in the starting state of $\Omega$.*

The above definition says that the state variables in each task are sufficient to capture the value of any trajectory consistent with the sub-hierarchy rooted at that task node.

**Theorem 2** *If the procedure HI-MAT produces a task hierarchy $\mathcal{H}$ from $\Omega$ and the DBN models $M$ then $\mathcal{H}$ is safe with respect to $M$. Further, if the DBN models are maximally sparse, for any hierarchy $\mathcal{H}'$ which is consistent with $\Omega$ and safe with respect to $M$, and $T_i = \langle X_i, S_i, G_i, C_i \rangle$ in $\mathcal{H}$, there exists $T'_i = \langle X'_i, S'_i, G'_i, C'_i \rangle$ in $\mathcal{H}'$ such that $X_i \subseteq X'_i$.*

**Proof sketch:** By the construction procedure, in any segment of trajectory $\Omega$ composed of primitive actions under a subtask $T_i$, all primitive actions check or set only the variables in $X_i$. Thus, changing any other variables in the initial state $s$ of $\Omega$ yielding $s'$ does not change the effects of these actions according to the DBN models. Similarly, all immediate rewards in the trajectory are also functions of the variables in $X_i$. Hence, the total accumulated reward and the probability of the trajectory only depend on $X_i$, and the hierarchy produced is safe with respect to $M$.

Suppose that $\mathcal{H}'$ is a consistent hierarchy which is safe with respect to $M$. Let $a_i$ be the last action in the trajectory $\Omega_i$ corresponding to the subtask $T_i$ in $\mathcal{H}$. By consistency, there must be some task $T'_i$ in $\mathcal{H}'$ that matches up with $a_i$. Recall that $X_i$ includes only those variables checked and set by $a_i$ to achieve the goal $G_i$. We claim that the abstraction variables $X'_i$ of $T'_i$ must include $X_i$. If this is not the case then, by maximal sparseness, there is a variable $y$ in $X_i - X'_i$ and some values $y_1$ and $y_2$ such that the probabilities of the next state or reward are different based on whether $y = y_1$ or $y = y_2$. Hence, $\mathcal{H}'$ would not be safe, leading to a contradiction. □

**Corollary 1** *If the DBN models are maximally sparse then the maximum size of the value function table for any task in the hierarchy produced by HI-MAT is the smallest over all safe hierarchies which are consistent with the trajectory.*

**Proof:** Direct consequence of part 2 of the previous theorem. □

The significance of the above corollary lies in the fact that the size of the value-function table is exponential in the number of variables $n_i = |X_i|$ in the abstraction of task $T_i$. If all features are binary and there are $t$ tasks then the total number of values for the value-function tables is $O(t\,2^{n_{\max}})$. Since the hierarchy is a tree with the primitive actions at the leaves, the number of subtasks is bounded by $2l$ where $l$ is the length of the trajectory. Hence, we can claim that the number of parameters needed to fully specify the value-function tables in our hierarchy is at most $O(l)$ times that of the best possible.

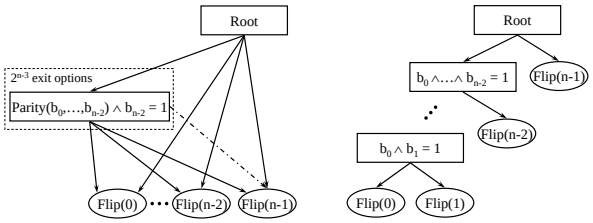
Our analysis does not address state abstractions arising from the so-called *funnel* property of subtasks where many starting states result in a few terminal states. Funnel abstractions permit the parent task to ignore variables that, while relevant inside the child task, do not affect the terminal state. Nevertheless, our analysis captures some of the key properties of our algorithm including consistency with the trajectory, safety, minimality, and sheds some light on its effectiveness.

## 5. Empirical Evaluation

We test three hypotheses. First, we expect that employing a successful trajectory along with the action models will allow the HI-MAT algorithm to induce task hierarchies that are much more compact than (or at least as compact as) just using the action models. Second, in a transfer setting, we expect that the hierarchies induced by HI-MAT will speed up convergence to the optimal policy in related target problems. Finally, we expect that the HI-MAT hierarchies will be applicable to and speed up learning in RL problems which are different enough from the source problems such that value functions either do not transfer or lead to poor transfer.

### 5.1. Contribution of the Trajectory

To highlight our first hypothesis, a modified Bitflip domain (Diuk et al., 2006) is designed as follows. The state is represented by $n$ bits, $b_0 b_1 \ldots b_{n-1}$. There are $n$ actions denoted by Flip($i$). Flip($i$) toggles $b_i$ if both




(a) VISA hierarchy. Task labels are the exit conditions; dash-dot arrows indicate exit options.

(b) HI-MAT hierarchy. Task labels are the termination conditions.

*Figure 2.* Task hierarchies for the modified Bitflip domain.

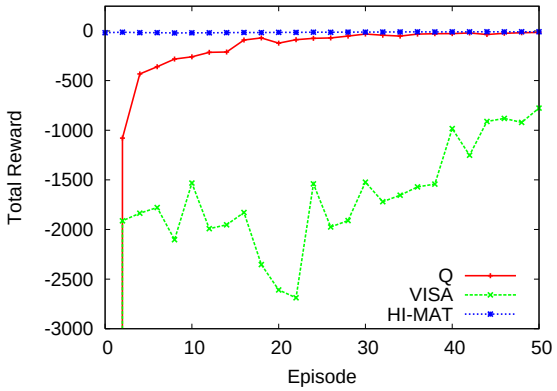



*Figure 3.* Performance of Q, VISA, and HI-MAT in the 7-bit modified Bitflip domain (averaged over 20 runs).

$b_{i-1}$ is set and the parity across bits $b_0, \ldots, b_{i-1}$ is even when $i$ is even (odd otherwise); if not, it resets the bits $b_0, \ldots, b_i$. All bits are reset at the initial state, and the goal is to set all bits.

We ran both VISA and HI-MAT in this domain with $n = 7$, and compared the induced hierarchies (Figure 2). We observe that VISA constructs an exponentially sized hierarchy even with subtask merging activated within VISA. There are two reasons for this. First, VISA relies on the full action set to construct its causal graph, and does not take advantage of any context-specific independence among its variables that may arise when the agent acts according to certain policies. Specifically, for this domain, the causal graph constructed from DBN analysis has only two strongly connected components (SCCs): one partition has $\{b_0, \ldots, b_{n-2}\}$, and the other has $\{b_{n-1}\}$. This SCC cannot be further decomposed using only information from the DBNs. Second, VISA creates exit options for all strongly connected components that transitively influence the reward function, whereas only a few of these may actually be necessary to solve the problem. Specifically, for this problem, VISA creates

an exit condition for any instantiation that satisfies $\mathsf{parity}(b_0, \ldots, b_{n-2}) \wedge b_{n-2} = 1$, resulting in exponential number of subtasks shown in Figure 2(a). The successful trajectory provided to HI-MAT achieves the goal by setting the bits going from left to right, and results in the hierarchy in Figure 2(b). The performance results are shown in Figure 3. VISA's hierarchy converges even slower than the basic Q learner because the root has $O(2^n)$ children as opposed to $O(n)$.

This domain has been engineered to highlight the case when access to a successful trajectory allows for significantly more compact hierarchies than without. We expect that access to a solved instance will usually improve the compactness of the resulting hierarchy.

### 5.2. Transfer of the Task Hierarchy

To test our remaining hypotheses, we apply the transfer setting to two domains: Taxi and the real-time strategy game Wargus. The Taxi domain has been described in Section 3. The source and target problems in Taxi differ only in the wall configurations; the passenger sources and destinations are the same. This is engineered to allow value-function transfer to occur. For Wargus, we consider the *resource collection* problem. Here, the agent has units called peasants that can harvest gold and wood from goldmines and forests respectively, and deposit them at a townhall. The goal is to reach a predetermined quota of gold and wood. Since the HI-MAT approach does not currently generalize to termination conditions involving numeric predicates, the state representation of the domain replaces the actual quota variables with Boolean variables that are set when the requisite quotas of gold and wood are met. We consider target problems whose specifications are scaled up from that of the source problems, including the number of peasants, goldmines, forests, and the size of the map. In this domain, coordination does not affect the policy significantly. Thus, in the target maps, we learn a hierarchical policy for the peasants using a shared hierarchy structure without coordination (Mehta & Tadepalli, 2005). In each case, we report the total reward received as a function of the number of episodes, averaged over multiple trials.

We compare three basic approaches: (1) non-hierarchical Q-learning (Q), (2) MAXQ-learning applied to a hierarchy manually engineered for each domain (Manual), and (3) MAXQ-learning applied to the HI-MAT hierarchy induced for each domain (HI-MAT). The HI-MAT algorithm first solves the source problem using flat Q-learning, and generates a successful trajectory from it. In Taxi, we also show the performance of initializing the value-function tables with values learned from the source problem – these curves are suffixed with the phrase "with value". In Wargus, we include the performance of VISA. The results of these experiments are shown in Figures 4 and 5.

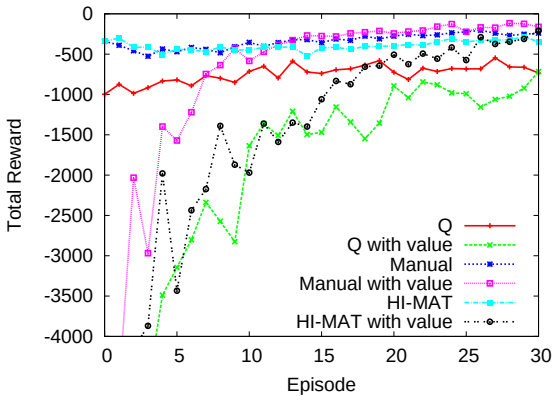



*Figure 4.* Performance in the Taxi domain (averaged over 20 runs). Source and target problems differ only in the configuration of the grid walls.

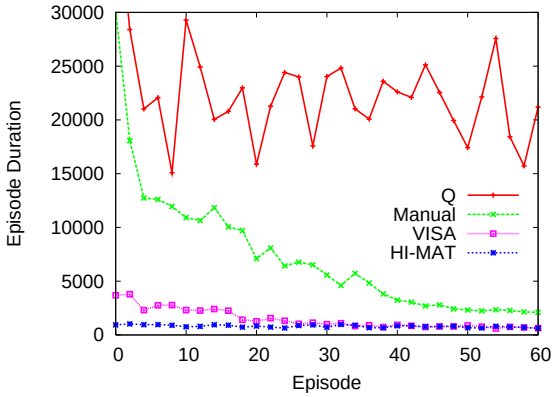



*Figure 5.* Performance in the Wargus domain (averaged over 10 runs). Source: $25 \times 25$ grid, 1 peasant, 2 goldmines, 2 forests, 1 townhall, 100 units of gold, 100 units of wood. Target: $50 \times 50$ grid, 3 peasants, 3 goldmines, 3 forests, 1 townhall, 300 units of gold, 300 units of wood.

Although the target problems in Taxi allow value-function transfer to occur, the target problems are still different enough that the agent has to "unlearn" the old policy. This leads to negative transfer evidenced in the fact that transferring value functions leads to worse rates of convergence to the optimal policy than transferring just the hierarchy structure with uninitialized policies. This indicates that transferring structural knowledge via the task-subtask decomposition can be superior to transferring value functions especially when the target problem differs significantly in terms of its optimal policy. In Wargus, the difference

between the source and target problems renders direct value-function transfer impossible even though the hierarchy structure still transfers.

In Taxi, we observe that MAXQ-learning on HI-MAT's hierarchy converges to the optimal policy at a rate comparable to that of the manually-engineered hierarchy. However, in Wargus, HI-MAT's hierarchy is faster to converge than the manually-engineered one because, by analyzing the solved source problem, it is able to find stricter termination conditions for each subtask. Consequently, reducing the policy space in the target problem leads to a greater speed-up in learning than reducing the number of value parameters via subtask sharing as in the manually-engineered hierarchy. The improved rate of convergence is in spite of the fact that HI-MAT does not currently merge subtly different instantiations of the same subtask so there is room for further improvement. VISA's performance suffers initially due to a large branching factor at the root option (which directly includes all the navigation actions).

## 6. Conclusion

We have presented an approach to automatically inducing MAXQ hierarchies from solved RL problems. Given DBN models and an observed successful trajectory, our method analyzes the causal and temporal relationships between actions, and partitions the trajectory recursively into subtasks as long as the state abstraction improves. We show that the learned hierarchies are consistent, safe, and have compact value-function tables. Our empirical results indicate that using the observed trajectory can allow us to learn more compact hierarchies. Further, in a transfer setting, our hierarchies perform comparably to manually-engineered hierarchies, and improve the rate of convergence where direct policy transfer does not help.

We are currently working on extending the approach to handle disjunctive goals. Further, in order to ensure hierarchical optimality, we may need to deal with non-zero pseudo-rewards. In related work, we are also investigating methods that learn compact DBN models of the MDP. Finally, an important challenge for the future is to investigate the transfer scenario where the induced hierarchy may need to be altered structurally in order to apply effectively to the target problem.

## Acknowledgments

We thank Mike Wynkoop and the anonymous reviewers for their input. We gratefully acknowledge the support of the Defense Advanced Research Projects Agency under DARPA grant FA8750-05-2-0249.

## References

Andre, D., & Russell, S. (2002). State Abstraction for Programmable Reinforcement Learning Agents. *AAAI* (pp. 119–125).

Şimşek, Ö., & Barto, A. (2004). Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning. *ICML* (pp. 751–758).

Dietterich, T. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Diuk, C., Littman, M., & Strehl, A. (2006). A Hierarchical Approach to Efficient Reinforcement Learning in Deterministic Domains. *AAMAS* (pp. 313–319).

Hengst, B. (2002). Discovering Hierarchy in Reinforcement Learning with HEXQ. *ICML* (pp. 243–250).

Jonsson, A., & Barto, A. (2006). Causal Graph Based Decomposition of Factored MDPs. *Journal of Machine Learning Research*, *7*, 2259–2301.

Marthi, B., Kaelbling, L., & Lozano-Perez, T. (2007). Learning Hierarchical Structure In Policies. *NIPS Hierarchical Organization of Behavior Workshop.*

McGovern, A., & Barto, A. (2001). Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. *ICML* (pp. 361–368).

Mehta, N., & Tadepalli, P. (2005). Multi-Agent Shared Hierarchy Reinforcement Learning. *ICML Rich Representations in Reinforcement Learning Workshop.*

Menache, I., Mannor, S., & Shimkin, N. (2001). Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. *ECML* (pp. 295–306).

Pickett, M., & Barto, A. (2002). PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. *ICML* (pp. 506–513).

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, *112*, 181–211.

Tadepalli, P., & Dietterich, T. (1997). Hierarchical Explanation-Based Reinforcement Learning. *ICML* (pp. 358–366).

Thrun, S., & Schwartz, A. (1995). Finding Structure in Reinforcement Learning. *NIPS* (pp. 385–392).