Automatic Discovery and Transfer of Task Hierarchies in Reinforcement Learning

Neville Mehta[1] and Soumya Ray[2] and Prasad Tadepalli[1] and Thomas Dietterich[1]
[1]Oregon State University, Corvallis, OR 97331
[2]Case Western Reserve University, Cleveland, OH 44106

Abstract

Sequential decision tasks present many opportunities for the study of transfer learning. A principal one among them is the existence of multiple domains that share the same underlying causal structure for actions. We describe an approach that exploits this shared causal structure to discover a hierarchical task structure in a source domain, which in turn speeds up learning of task execution knowledge in a new target domain. Our approach is theoretically justified and compares favorably to manually-designed task hierarchies in learning efficiency in the target domain. We demonstrate that causally motivated task hierarchies transfer more robustly than other kinds of detailed knowledge that depend on the idiosyncrasies of the source domain and are hence less transferable.

Introduction

Sequential decision tasks such as trip planning and real-time strategy games offer special challenges and opportunities for studying transfer learning. These domains are complex, and good performance requires selecting long chains of actions to achieve subgoals needed for ultimate success. Reinforcement learning in these domains, because it involves a process of exploratory trial-and-error, can take a very long time to discover these long action chains. Fortunately, it is often possible study smaller versions of these domains that share the same fundamental structure but that involve fewer objects and smaller state spaces. Reinforcement learning on these smaller domains is much faster. If it can discover the shared structure and transfer it to the large scale domains, then this provides a much more efficient way of achieving good performance.

Our approach is based on the claim that the key to transfer learning is to discover and represent deep forms of knowledge that are invariant across multiple domains. Consider the problem of driving to work. There are surface aspects, such as the amount of time it takes to get from home to the office or the selection of the best route, that may be highly regular, but they are unlikely to transfer when you move to a new city. On the other hand, the task structure involved in driving, such as starting the car, driving, obeying traffic laws, parking, etc., depends only on the *causal structure* of the actions involved, and hence transfers more successfully from one city to another. We are interested in transferring task knowledge between source and target domains that share the same causal structure, that is, the actions in both domains depend upon and influence the same state variables. This is weaker than assuming that the behavior of actions is exactly identical in two domains. For instance, although two different cars may have very different engines that accelerate at different rates, they both speed up when you press the accelerator and slow down when you hit the brakes.

Many domains have hierarchical task structure that, once acquired, can lead to faster learning of skills and efficient planning. The areas of hierarchical planning (Nau et al. 2003) and its close cousin, hierarchical reinforcement learning (Barto and Mahadevan 2003), are both based on this insight. Although most of the research in these areas employs task hierarchies designed by hand which requires significant human effort and expertise, there is some work on learning these hierarchies (Reddy and Tadepalli 1997; Langley and Choi 2006; Hogg, Kuter, and Munoz-Avila 2009; Jonsson and Barto 2006; Mehta et al. 2008b). Since tasks are like subroutines, automatically learning tasks addresses the question of what constitutes a good subroutine, a long-standing fundamental question in computer science.

Our work builds on the previous research and automatically induces task hierarchies based on two key claims: (1) It is possible to uncover the hierarchical task structure in a domain by analyzing the causal relationships between actions in successful trajectories. (2) The hierarchical task structure is more robustly transferable across domains than other kinds of knowledge such as the utilities of being in different states (the value function) or the utilities of executing different actions (the action-value function).

We show how a task hierarchy can be efficiently discovered from a single source trajectory by exploiting the knowledge of the domain dynamics. In this respect, our work resembles explanation-based learn-

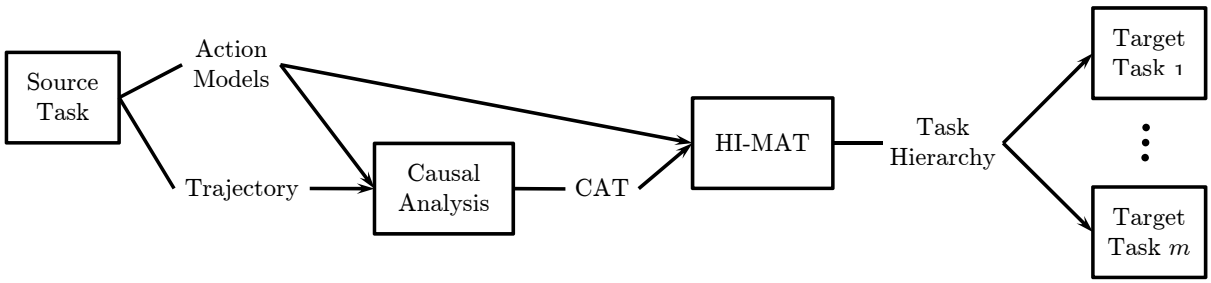


Figure 1: The architecture of our system. Action models and a single trajectory from the source task are analyzed to generate a causally annotated trajectory (CAT). The CAT, along with the action models, is then provided to the HI-MAT algorithm, which discovers a task hierarchy. The structure of this task hierarchy facilitates faster learning in a range of target tasks that share the causal structure of the source task.

ing (EBL) which generates explanations of successful trajectories to deduce sound knowledge (Minton 1988; Tadepalli and Dietterich 1997; Nejati, Langley, and Konik 2006). However, unlike the EBL paradigm which relies on complete knowledge of the actions to learn sound control rules, we only learn based on the qualitative causal structure of actions, which is more readily available for domains and yields more widely transferable knowledge. The transferred knowledge we focus on is the hierarchical task structure, which can be further specialized to learn detailed action policies in the target task through further experience.

The high-level schema of our approach is shown in Figure 1. The action models and a successful trajectory are extracted from a source task and analyzed to identify the causal relationships between the actions in the trajectory. Our hierarchy discovery algorithm, HI-MAT (Hierarchy Induction via Models and Trajectories), leverages this causally annotated trajectory to discover a coherent task hierarchy that minimizes the number of inter-task causal links. This task hierarchy is transferable for faster learning in all target tasks that share the causal structure of the source task.

The rest of the article is organized as follows. The next section provides the technical background needed to describe our approach in more detail. It also presents our main illustrative domain, which is a real-time strategy game called Wargus. This is followed by the main technical section, which covers the approach. The section on empirical evaluation demonstrates that the induced task hierarchies are comparable in effectiveness to manually-designed hierarchies and that transferring structured knowledge in the form of task hierarchies across some domains can be more effective than transferring the parameters of the optimal policy or value function. The article concludes with a discussion on related and future work.

Background and Component Technologies

In this section, we first briefly describe reinforcement learning. Next, we describe the Wargus domain, which serves as a running example for the rest of the article and which provides a basis for empirical evaluations. Finally, we discuss the two component technologies of our approach: action models and task hierarchies.

Reinforcement Learning

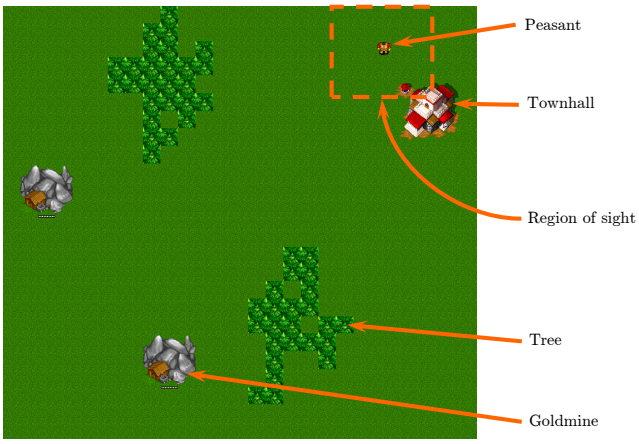
Consider the following scenario: an agent is trapped in an unfamiliar maze and it would like to find the exit as quickly as possible. It can move around, and when it exits the maze it receives positive feedback. This is an example of a *sequential decision-making* problem, where the optimal solution is a sequence of actions taken by the agent that satisfies a global optimality criterion such as minimum path length. *Reinforcement learning* (RL) studies such problems and models them as Markov decision processes (MDPs). An MDP consists of a set of world *states*, a set of *actions* that the agent can execute to change the state of the world, a *transition* function that determines how the state of the world changes with respect to the executed action, and a *reward* function that represents the immediate feedback the agent receives when executing an action in a state. The transition and reward functions can be stochastic, that is, the resultant state and reward upon executing an action can exhibit random variations. In a *factored* state space, states are described by a set of variables. For instance, the state in the maze can be represented by a vector of coordinates rather than by a unique serial number. An MDP has the Markov property which implies that future states are independent of past states given the present state and action.

The *policy* of an agent in an MDP specifies what action the agent will execute in each state. To solve an MDP, the agent must find an *optimal* policy — one that accumulates the maximum attainable expected reward. In the maze example, such a policy would follow the shortest path to the exit from every location in the maze. The expected cumulative reward for following a policy that eventually terminates from any state is well defined and finite. The expected cumulative reward when starting in state s and executing policy π is called the *value* of s with respect to π. The optimal policy and its value function can be computed through

classical approaches based on dynamic programming when the transition dynamics and the reward function (collectively called the domain model) are known and the number of states is small. If the domain model is unknown, then we can either apply model-based RL, which learns the action and reward models and applies dynamic programming, or model-free RL such as Q-learning, which learns a value function over state-action pairs and avoids the need to learn a domain model (Sutton and Barto 1998).

Dynamic programming suffers from the "curse of dimensionality" — it does not scale to large real-world domains where the number of states is too large (exponential in the number of state variables) to store in a table. One way around this is to approximate the value function or the policy over a small number of informative features derived from the state variables. Another approach that greatly mitigates the curse is called hierarchical reinforcement learning (HRL), which leverages the natural task structure in the domain to decompose the MDP into several sub-MDPs, solves them separately, and recomposes them to solve the MDP. The following sections describe this approach more fully through the running example of the Wargus resource-gathering domain.

Wargus Resource-Gathering Domain



(a) Wargus map.

Variable	Description
$p.l$	Peasant's location
$p.r$	Peasant's resource (gold/wood)
$r.g, r.w, r.t$	Indicators for goldmine, forest, or townhall regions
$q.g, q.w$	Quota indicators for gold and wood.

(b) Wargus state variables.

Figure 2: Wargus Resource-Gathering Domain.

Wargus is a real-time strategy game. Two players inhabit a world that contains entities such as peasants, goldmines, and townhalls, and resources such as gold and wood. To win, a player must collect resources, build various kinds of units (e.g., townhalls, lumber mills, footmen, dragons), and then deploy those units to defeat the enemy in battle. We focus on the resource-gathering aspect of Wargus, where our agent needs to collect a specified quota of gold and wood as quickly as possible given an arbitrary map as shown in Figure 2a. To achieve the resource goal, the agent must command peasants to collect gold from goldmines and harvest wood from forests and deposit those resources at townhalls. The game state is described through the following variables: $p.l$ represents the coordinate location of peasant p, $p.r$ indicates if the peasant is empty-handed (empty) or carrying a resource (gold or wood), the binary $r.g, r.w, r.t$ variables indicate if there is a goldmine, forest, or townhall in the peasant's immediate vicinity, and the binary $q.g, q.w$ variables indicate whether the required quotas of gold and wood have been met.

The actions available to the agent are MineGold, ChopWood, Deposit, Goto(loc) for mining gold when in the vicinity of a gold mine, chopping wood when in the vicinity of a forest, and navigating to a location loc on the map (this uses the internal path-finding routine of the game). If MineGold is executed when an empty peasant p is in the vicinity of a goldmine, then this will change $p.r$ to gold; otherwise, this action will have no effect (the game state does not change). Similarly, a successful ChopWood will change $p.r$ from empty to wood. A successful Deposit will deposit the peasant's resource at the townhall, set $p.r$ to empty, and provide a positive reward to the agent. Even though the actions have deterministic outcomes, they appear to behave nondeterministically due to the incomplete representation of the world provided by the state variables. The overall goal of achieving the requisite quotas $q.g = 1 \wedge q.w = 1$ is attained by having peasants repeatedly collect gold and wood from goldmines and forests, and deposit these resources at townhalls. Employing Boolean variables to represent the map's topography and the resource quotas actually results in non-Markovian system dynamics and is approximated with probability distributions within the MDP. For example, transitioning from an unmet quota to meeting it actually depends upon the number of deposits, but this is approximated by the distribution seen at the leaf for $q.g'$ in Figure 3.

Action Models

Our approach to discovering task hierarchies relies on compact, inspectable action models. An action model represents the effect that executing an action has on the state variables. For instance, depositing gold at a townhall changes the peasant's resource variable $p.r$ from gold to empty. We employ *dynamic Bayesian networks* (DBNs) with *context-specific independence* to represent the action models (see Figure 3). A DBN is a bipartite directed graph; each node represents a state variable and each edge represents a direct causal dependence (Dean and Kanazawa 1990). The first stage of the graph (shown on the left in Figure 3) represents the state variables before the action is executed, and the second stage (on the right) represents the state after

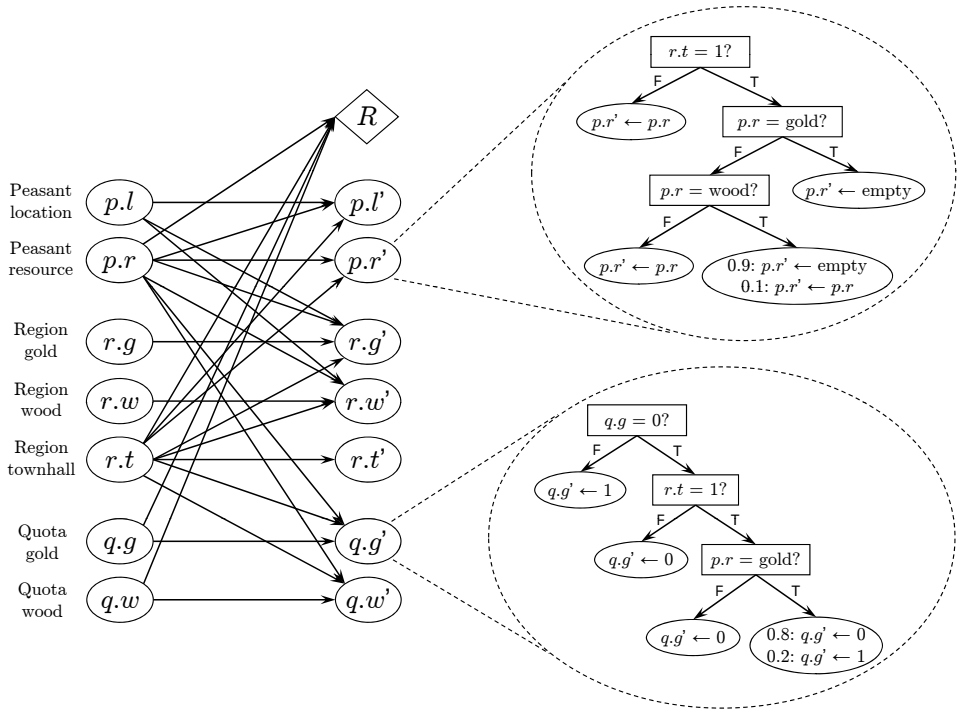


Figure 3: The dynamic Bayesian network model for the Deposit action in the Wargus domain. The left and right columns of elliptical nodes represent the state variables before and after the action is executed. The diamond node labeled R represents the immediate reward received after the action is executed. The peasant resource $p.r'$ and the gold quota $q.g'$ nodes have been expanded to show the decision tree representation of their conditional probability distributions. The internal nodes of the tree check conditions involving the state variables, such as the peasant being near a townhall ($r.t = 1?$), and the leaves contain the distributions over resultant values, such as executing Deposit near a townhall when carrying gold results in the peasant's resource being empty ($p.r' \leftarrow$ empty) with probability 1, or that the gold quota is met with probability 0.2 if all the conditions are satisfied.

the action is executed. The reward received when executing the action is represented by the diamond-shaped node in the second stage.

Each node v in the second stage contains an inspectable description of the conditional probability distribution $\Pr(v|\text{Parents}(v))$, where $\text{Parents}(v)$ is the set of state variables with edges leading into v. We assume that this conditional probability distribution is represented as a decision tree that captures context-specific dependencies. The internal choice nodes of the decision tree at node v test the values of the $\text{Parents}(v)$ and the leaf nodes specify a probability distribution over the values of v in the resulting state.

A DBN for the Deposit action in Wargus is shown in Figure 3. The tree structure for $p.r'$ represents the fact that it remains unchanged if the peasant is not near a townhall or when it is not carrying gold or wood; otherwise, the variable changes (with high probability) to reflect the fact that the Deposit succeeds and $p.r$ becomes empty. This captures the fact that the probability distribution over $p.r'$ depends on the context — it is more compact than representing the probability distribution as a table that enumerates all possible combinations of values of the parents (Boutilier et al. 1996).

In the work described in this paper, the action models have been hand-engineered. In other work, Wynkoop and Dietterich (2008) have shown how to learn these action models from exploratory trial-and-error trajectories.

Task Hierarchies

Task hierarchies play a central role in efficient planning, plan recognition, and explanation generation. They are essential for the success of automated planning in large real-world applications where planning at the level of primitive actions is intractable. Task hierarchies enable the planner to plan at multiple time scales where plans at higher levels (larger temporal scales) are refined into sub-plans at lower levels (finer temporal scales). For instance, someone planning to attend an international conference is likely to first arrange for funding before registering for the conference, which in turn may be considered before choosing which travel website to use. The planning process is unlikely to begin by considering which muscle to move first, although the plan execution could indeed begin that way. The structure of the task hierarchy and the state abstraction possible at each task constrain the hierarchical policies and conse-

quently speed up learning.

Our particular approach is based on the MAXQ framework for representing task hierarchies (Dietterich 2000). Each task in a MAXQ task hierarchy has a goal or termination condition that describes what the task is trying to achieve and the conditions under which it can be invoked. It also has a set of child tasks that it can invoke recursively to achieve its goal and a set of relevant state variables, which is called the *state abstraction*. The state abstraction constrains the representation of the value function for policies within that task. The root of the task hierarchy corresponds to the overall MDP, while the leaves correspond to primitive actions. Given a MAXQ task hierarchy, a *hierarchical* policy is a collection of local policies, one for each task. Each local policy maps the abstracted state within the task to its child tasks. During execution of a hierarchical policy, a task invoked by its parent follows its local policy until a state is reached where its goal or termination condition is true. At this point, control returns to the calling parent task, along with the reward obtained during its execution. Typically, an agent searches for a *recursively optimal* policy, where the local policy for each task optimizes the expected total reward within that task assuming that all the descendant tasks are solved in a recursively optimal fashion.

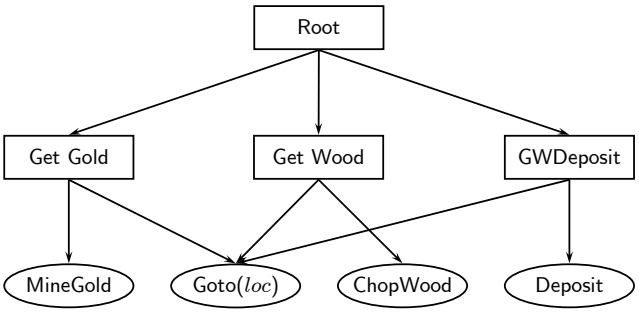


Figure 4: A task hierarchy for the Wargus resource-gathering domain.

The Wargus resource-gathering domain is too complex for a non-hierarchical solution, because the larger the map, the larger the state space and the larger the number of navigation actions available at each state. However, the task hierarchy shown in Figure 4 decomposes the overall problem into simpler subproblems. The Root task tries to learn a policy for meeting the overall resource quota in the original MDP by solving and combining the solutions of three subtasks: a GetGold subtask that moves a peasant to the vicinity of a goldmine and then mines for gold, a GetWood subtask that does the same for wood, and a GWDeposit subtask that brings a peasant carrying gold or wood back to the townhall to deposit the resource. Note that because GetGold is not involved in getting wood or depositing anything, it does not need to know anything about the townhall ($r.t$) and wood ($r.w$) regions; further, this task is only active when the peasant is carrying nothing ($p.r$ = empty). Similarly, GWDeposit does not need to know about mining gold or chopping wood and is active only when the peasant is carrying something. Thus, each of these subtasks describes a sub-MDP with fewer state variables and actions than the original problem.

In standard HRL, a domain-specific task hierarchy such as the one in Figure 4 is designed by hand and provided to the system. Value functions for different subtasks are learned by applying hierarchical RL algorithms such as MAXQ-0 learning. This hierarchical approach mitigates the curse of dimensionality of flat MDPs for two reasons. First, it increases the temporal scales at which decisions are made. For example, consider a peasant who is moving toward a goldmine. In the hierarchical MDP, the peasant would be executing the GetGold subtask, so when it reaches the mine, it will immediately mine gold. But in the flat MDP, the agent must re-evaluate all actions at each time step, so once the peasant reaches the mine, the agent must decide whether to mine gold or perhaps navigate to a forest instead. The hierarchy serves to guide trail-and-error exploration. The second benefit of the hierarchy is that it enables state abstraction, so that subtasks can employ a lower-dimensional representation for their value functions, which speeds learning.

Technical Approach

We consider source and target MDPs where the agent seeks to achieve a known goal. 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 DBN action and reward models characterizing the source MDP, as well as a successful trajectory. Our objective is to automatically induce a task hierarchy that can suitably constrain the policy space in the source domain and can transfer to achieve faster learning in related target problems.

Causal Analysis

The input trajectory is a sequence of actions that achieves the overall goal in the source problem. A trajectory in Wargus is a sequence of Goto, MineGold, ChopWood, and Deposit actions that together achieves the requisite quota of gold and wood. The trajectory is first annotated with causal information using the DBN models before it is fed to the HI-MAT algorithm. The intent of this annotation is to identify how executing actions affects the state variables that are relevant to future actions and, ultimately, the goal of the overall task. For example, we add information to indicate that the MineGold and ChopWood actions enable the subsequent Deposit action by setting the peasant's resource $p.r$ to something other than empty, which is required for Deposit to succeed.

The annotation is based on the relevance of variables to actions which is gleaned from the DBNs. More

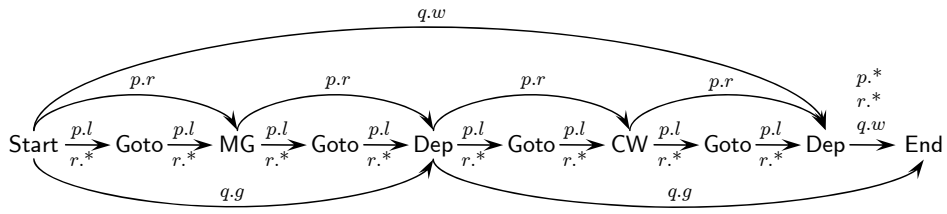


Figure 5: A causally annotated trajectory (CAT) for the Wargus domain. Variables: $p.l$ = peasant location; $p.r$ = peasant resource; $r.g$, $r.w$, $r.t$ = regions; $q.g$, $q.w$ = quotas. Actions: Goto, MG = MineGold, CW = ChopWood, Dep = Deposit; the trajectory is prepended with Start which sources all the variables, and appended with End which sinks all the variables. An edge goes from one action to another (later in the trajectory) when a variable is relevant to both, but no intervening action; edges are labeled with the associated variables. For succinctness, $p.* = \{p.l, p.r\}$, and $r.* = \{r.g, r.w, r.t\}$.

specifically, a variable v is *relevant* to an action a in state s if the reward and transition dynamics for a either check or change v in s; it is *irrelevant* otherwise. In Figure 3, the peasant being near a townhall ($r.t$) is always relevant to Deposit because it is in the root node of the tree, but the peasant's resource ($p.r$) is only relevant when the peasant is near a townhall. A *causal* edge $a \xrightarrow{v} b$ connects a to another action b (b following a in the trajectory) iff v is relevant to both a and b and irrelevant to all actions in between. A *causally annotated trajectory* (CAT) is the original trajectory annotated with all the causal edges, sandwiched between dummy Start and End actions for which all variables are defined to be relevant. The CAT is preprocessed to remove any cycles present in the original trajectory (failed actions, such as an unsuccessful Deposit when the peasant is not near a townhall). A CAT for Wargus is shown in Figure 5.

The HI-MAT Algorithm

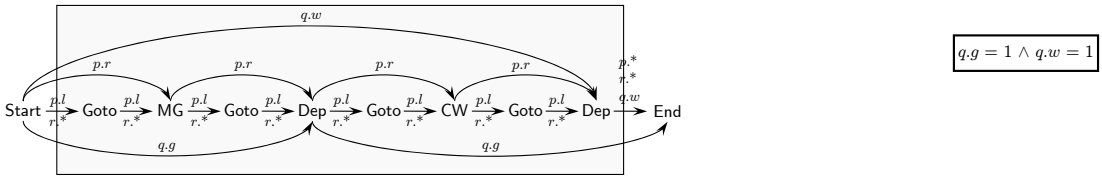
Given a CAT, the DBN models, and the MDP's goal as input, the HI-MAT algorithm discovers hierarchical structure by recursively partitioning the CAT. Each partition corresponds to a candidate subtask. This partitioning process works backward from the goals of the task. It regresses each goal through the action models to discover preconditions and employs a heuristic procedure to determine small CAT segments responsible for achieving the goal. For example, to achieve the overall goal in Wargus, a Deposit action must be executed. By analyzing the DBN for Deposit and considering the conditions that allow $q.g$ to be set to 1, the subgoal of $p.r$ = gold is identified, which then leads to the discovery of the subsequence of Goto and MineGold actions that achieve this new subgoal, and so on. The HI-MAT algorithm is outlined in Algorithm 1 and illustrated for the Wargus domain in Figure 6.

The HI-MAT algorithm partitions the CAT into unique segments, each achieving a single literal or a conjunction of literals (due to merging of segments). It is then invoked recursively on each element of the partition. HI-MAT bottoms out if either the CAT contains only a single primitive action, or it consists of actions whose relevances are identical. In the latter case, any further partitioning would only yield subtasks with the same state abstraction as the parent task and hence would not provide any further state abstraction benefit. Figure 7 shows the final task hierarchy induced by HI-MAT in Wargus with intuitive names for the subtasks based on their termination conditions and state abstraction. Note that the TGoto(townhall) subtasks have been merged by HI-MAT, because discovered subtasks with identical termination conditions and identical child tasks (recursively) are recognized as multiple calls to a unique subtask.

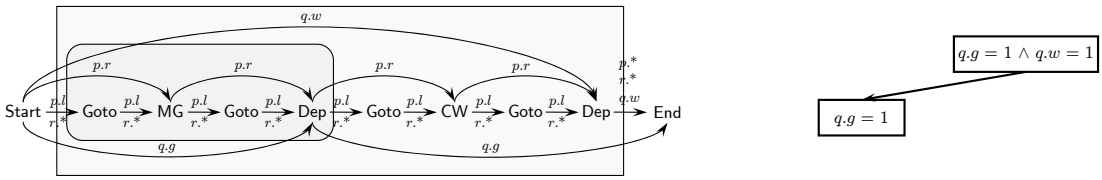
Algorithm 1 HI-MAT
Input: CAT Ω, DBN models, Goal predicate G.
Output: Task T.

```
if Ω contains a single action a then
    return task T with termination G, state abstraction based on variables relevant to a, and child a
else if actions in Ω have identical sets of relevance then
    return task T with termination G, state abstraction based on Ω's relevance, actions in Ω
Extract trajectory segments from Ω for goal literals and any literals entering the segments
if a segment = Ω then
    Create two segments: one with the ultimate action and the other with the rest and the ultimate action's precondition as its goal
Merge all overlapping segments
for every CAT segment do
    Invoke HI-MAT recursively on it to discover the subtask hierarchy and add it as a child of T
Set termination for T = G
Set state abstraction for T based on the relevant variables from Ω's merged DBN
Add all primitive actions to T that share DBN structure but are not already included
return task T
```

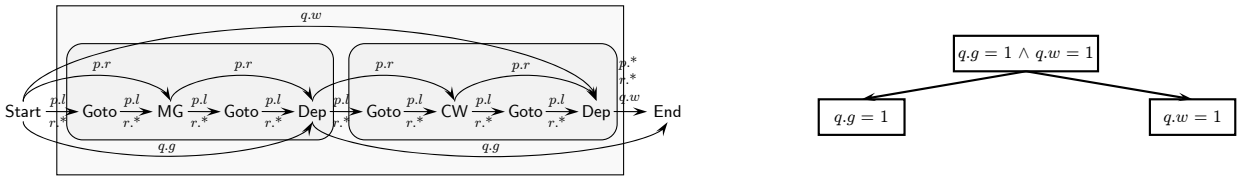
Discovering Tasks. For each literal in the goal condition, HI-MAT extracts the corresponding segment of the trajectory by finding the set of temporally contiguous actions in the CAT such that the final action achieves the literal and no other actions within the seg-



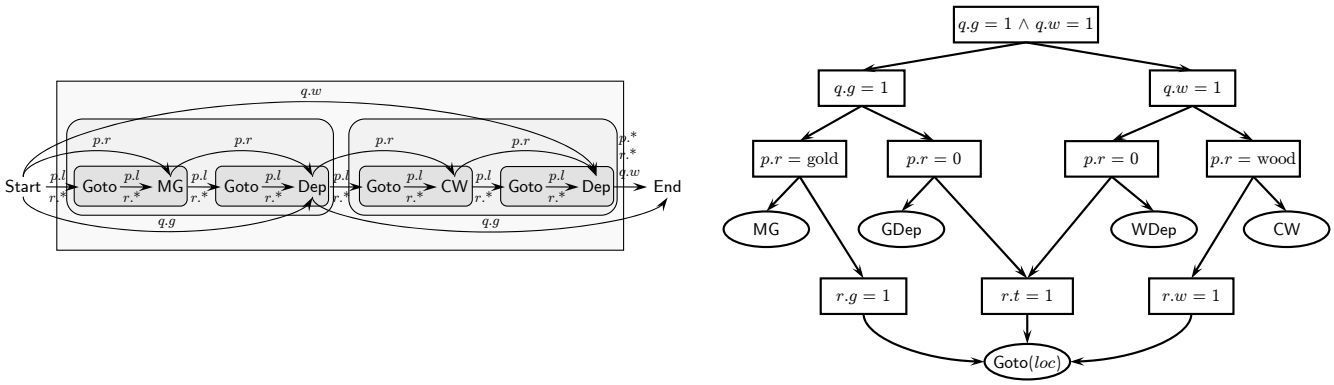
(a) The entire CAT is associated with the root task of collecting gold and wood. The root task terminates when the requisite amount of gold and wood have been collected.



(b) The segment corresponding to collecting the requisite amount of gold ($q.g = 1$) is extracted yielding the associated subtask.



(c) The segment corresponding to collecting the requisite amount of wood ($q.w = 1$) is extracted. Other literals entering this segment result in segments that are merged together with the $q.g = 1$ segment (merged goal condition not shown for clarity).



(d) The resultant task hierarchy after an entire parse of the CAT. The multiple invocations of the task that gets the peasant to the townhall ($r.t = 1$) within the CAT are merged into one unique task. However, the tasks that empty the peasant's resource ($p.r = 0$) are not merged because, while GDep and WDep are proxies for the Deposit action, the former's abstraction includes $q.g$ while that of the latter includes $q.w$ instead. The primitive Goto(loc) action is wrapped within tasks that get the agent to particular types of regions.

Figure 6: Illustrating the HI-MAT algorithm in the Wargus domain. The left-hand side shows the CAT scanning process, while the right-hand side shows the task hierarchy being built.

ment have any causal influences outside it. Note that the temporal contiguity of the actions that we assign to a task is required by the subroutine semantics of a hierarchical policy. In Figure 6b, HI-MAT extracts the segment responsible for collecting the gold quota (achieving $q.g = 1$), and discovers the subtask associated with the segment as a child of the root task. Figure 6c shows how HI-MAT extracts the segment for collecting the wood quota (achieving $q.w = 1$) by stopping the head of the segment at the action after the Dep action that achieves $q.g = 1$. Note that the causal annotation allows HI-MAT to correctly incorporate any redundant Goto actions that might be present in either of these segments.

Given an extracted segment, HI-MAT considers all literals that enter it (from earlier in the CAT) for further segment extraction to ensure that these literals are accounted for. In Figure 6c, the segment associated with collecting the wood quota ($q.w = 1$) has two incoming edges: one for the peasant's resource $p.r$, and the other for the peasant's location $p.l$. Consequently, HI-MAT extracts two segments associated with these literals. However, both these segments overlap with the segment that collects the gold quota ($q.g = 1$). (Note that extracted segments can only overlap fully when their shared ultimate action achieves the literals of all these segments.) HI-MAT merges the overlapping segments by replacing them with one that is assigned a conjunction of the subgoal literals. Thus, the merged goal condition is $q.g = 1 \wedge p.r = \text{empty} \wedge p.l = \text{townhall}$, but the second and third literals are always true when the first becomes true and we leave them out to reduce clutter. CAT scanning is repeated until all literals are accounted for; it can be proved that the set of subtasks output by the algorithm is independent of the order in which the unresolved literals are picked. If an extracted trajectory segment is equal to the entire CAT, then this implies that the segment achieves only the literal emerging out of the ultimate action. In this case, the trajectory is split into two new segments: one segment contains only the ultimate action, and the other segment contains everything prior to the ultimate action. The goal of this second segment is defined to be the preconditions of the ultimate action. For example, the CAT associated with collecting gold cannot be split further based on the annotation so it is forcibly split into two segments: one containing only the ultimate Dep action and another containing all actions prior to Dep within the CAT whose goal is to achieve the goal of getting the peasant back to the townhall with some gold.

Defining the Child Tasks and Termination Condition. To define a task, we must specify the set of child tasks that it can invoke and its termination predicate. The child tasks are those tasks that appear within all trajectory segments that constitute the task. The termination predicate is computed by taking the literal that was achieved by the segment and generalizing it subject to the conditions that appear in the DBNs. For example, although the specific condition for getting the peasant to a townhall involves both the peasant's location $p.l$ and the townhall indicator $r.t. = 1$, the DBN only checks the latter and consequently only $r.t = 1$ is the termination condition. Initially, the root task is associated with the entire CAT, and its termination condition is equal to the MDP's goal predicate as shown in Figure 6a.

Determining State Abstraction. Every task's local policy in the hierarchy is dependent only on a subset of the state variables. For instance, the agent can be oblivious to the location of forests when executing the task for collecting gold. HI-MAT tries to assign

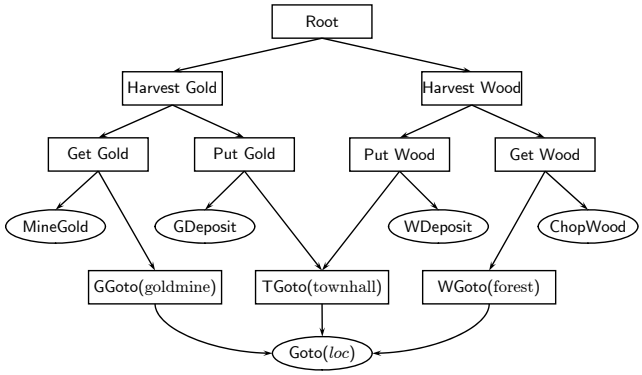


Figure 7: HI-MAT hierarchy for Wargus. The task names have been assigned intuitively based on the termination conditions. Note that although HI-MAT does not explicitly create parameterized subtasks, the different Goto subtasks are equivalent to bounding the location parameter to a goldmine, forest, or townhall because they only terminate when the peasant is in the vicinity of one of those entities.

the smallest set of relevant state variables to every task in its induced hierarchy to help expedite the learning of the local task policies. It determines this state abstraction by first constructing a *merged* DBN for the set of actions in the segment associated with the task by unioning the edges of the individual DBNs. This merged DBN represents variable dependencies for *any* sequence of these actions. Given the merged DBN for a set of actions, if we compute $\text{Parents}(\cdots(\text{Parents}(R)))$ until convergence, then the final set of variables is relevant to this set of actions.

The state abstraction for a task is the union of the set of relevant variables of the merged DBN for the task and the set of variables appearing in the termination predicate. When a task's relational termination condition involves other variables not already in the abstraction, these variables are added to the state abstraction. This has the effect of creating a task with these added variables as the formal parameters or arguments. In HI-MAT, this parameterization is implicit in the termination conditions rather than being explicit as in Dietterich (2000). For instance, the state abstraction of the navigation task only involves the peasant's location $p.l$. However, the 3 navigation tasks that terminate when the peasant is at a goldmine, forest, or townhall are equivalent to one explicitly parameterized navigation task with a single argument that is bound to a goldmine, forest, or townhall as needed.

Generalizing the Task Hierarchy. Because the CAT is based on a single successful trajectory in the source problem, not all primitive actions might be observed. Incorporating only the observed actions might limit the scope of transfer. To make the task hierarchy more generally applicable, HI-MAT checks if there are unobserved primitive actions that can be incorporated

into a task T with primitive child tasks without adding more state variables to T's current state abstraction. It does this by creating a merged DBN for the child actions and incorporating an unobserved action if its DBN is a subgraph of the merged DBN. The rationale here is to add unobserved actions that have the same causal effects as those of the child actions. For example, the given trajectory might only involve a few of the Goto actions, but all unobserved Goto actions are added to the induced task hierarchy because they all affect the same set of variables. However, if one of the Goto actions affects a different variable, then it will not be incorporated even if it might turn out to be useful in a target task.

Theoretical Analysis

In Mehta et al. (2008b), we showed that the HI-MAT algorithm enjoys the following properties under suitable conditions.

1. *Safety*: The state abstractions produced by the algorithm correctly represent the recursively optimal hierarchical value function for the hierarchy.
2. *Consistency*: HI-MAT outputs a task hierarchy consistent with the input trajectory, that is, it can generate the input trajectory with an appropriate value function.
3. *Compactness:* The abstraction of any task in the task hierarchy is maximally compact, that is, it does not contain redundant state variables.

The first property guarantees that the value function of the desired policy is representable by the hierarchy. The first two properties together imply that one could solve the input problem starting with the task hierarchy and learning an appropriate value function. Compactness determines the speed of learning in the target task after transfer — the more compact the abstraction, the fewer the number of parameters to be learned, and hence the faster the learning. The third property implies that the abstraction is as compact as possible while guaranteeing safety with respect to the input models.

Empirical Evaluation

In this section, we verify two hypotheses: (a) the hierarchies induced by HI-MAT speed up convergence to the optimal policy in related target problems, and (b) the HI-MAT hierarchies are applicable to and will accelerate learning in RL problems that are different enough from the source problems that value functions either do not transfer or lead to poor transfer.

Transfer of the Task Hierarchy

We test the transfer performance of task hierarchies within the Wargus domain. We consider target problems whose specifications—number of peasants, goldmines, forests, and the size of the map—are scaled up from those of the source problems. This scaling along with the random placement of the entities in the source and target problems ensures that, although the probability parameters of the transition dynamics differ between the two, they are structurally the same. For instance, although the townhall is in a different location in the target map, depositing wood in its vicinity employs the same causal relationships as in the source map.

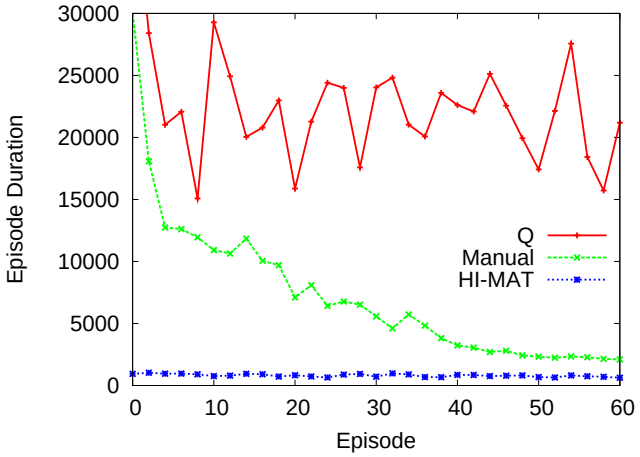


Figure 8: Performance of 3 learning algorithms in a target Wargus problem (averaged over 10 runs). Source problem: 25×25 grid, 1 peasant, 2 goldmines, 2 forests, 1 townhall, 100 units of gold, 100 units of wood; target problem: 50×50 grid, 3 peasants, 3 goldmines, 3 forests, 1 townhall, 300 units of gold, 300 units of wood. Learning algorithms: the non-hierarchical RL algorithm Q-learning (Q), the hierarchical RL algorithm MAXQ-0 applied to the manually-designed hierarchy (Manual), and MAXQ-0 applied to the HI-MAT induced hierarchy (HI-MAT). Note that although there is some improvement via experience in the HI-MAT learning curve, it looks flat because of the scale of the vertical axis required to show the learning curves of the much slower learners.

We compare 3 approaches: (1) a basic non-hierarchical RL algorithm called Q-learning (Q), (2) the hierarchical RL algorithm MAXQ-0 applied to the manually-designed hierarchy (Manual), and (3) MAXQ-0 applied to the HI-MAT induced hierarchy (HI-MAT). The source problem is solved using non-hierarchical RL, and a successful trajectory is generated from it for HI-MAT. Figure 8 shows the total duration of an episode as a function of the number of episodes experienced, averaged over 10 runs.

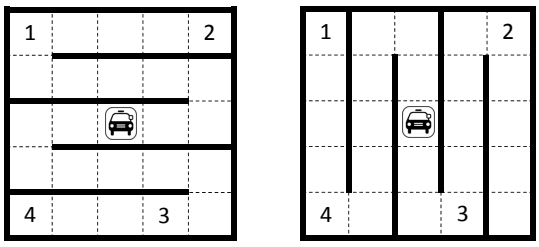
A surprise to us was that HI-MAT's hierarchy is faster to converge than the manually-designed one. It turns out that HI-MAT was able to find a more refined task hierarchy with stricter termination conditions for each subtask than our hand-written hierarchy. Consequently, the reduced policy space in the target problem yields a greater speedup in learning than reducing the number of value parameters via subtask sharing as in the manually-designed hierarchy. The improved rate of

convergence is in spite of the fact that HI-MAT does not currently merge different invocations of the same explicitly parameterized subtask, so there is room for further improvement.

This experiment reveals the spectrum of performance based on the structure of the task hierarchy transferred across problems. The performance of Q is equivalent to that of transferring the shallowest task hierarchy (where all the primitive actions are children of the root task). Although such a hierarchy is applicable to a broader range of target problems, no knowledge is transferred from the source problem and the net effect is identical to learning in the target from scratch. The manually-designed task hierarchy is slightly more constrained than the shallowest hierarchy and consequently Manual performs better in the target problem, but it has a more limited transfer horizon than Q. Although the HI-MAT hierarchy is even more constrained and consequently HI-MAT performs the best, it still has the same transferability as the manually-designed hierarchy, because the additional policy constraints are only made at the level of the causal relationships in the transition dynamics and not at the parameter level.

Transferring Structure over Value

To test whether hierarchical structure transfers better than value functions, we design source and target problems in the Taxi domain (Dietterich 2000), where a taxi transports a passenger from a source location to a destination within a grid world. The source and target problems differ only in their wall configurations, while the passenger sources and destinations stay the same as shown in Figure 9. This setup is specifically engineered so that a value function from the source domain is a syntactically legal value function in the target domain (i.e., the state and action spaces are identical). However, the differing wall configurations affect the particular parameterization of the transition dynamics in the two problems. The difference between the source and target problems in the Wargus domain renders this kind of direct value-function transfer impossible.



(a) Source problem. (b) Target problem.

Figure 9: Source and target problems in the Taxi domain. In this domain, a taxi needs to get a passenger from one numbered location to another on the grid. Source and target problems differ only in the configuration of the grid walls.

Figure 10 shows a manually-designed task hierarchy for Taxi. The decomposition uses the knowledge that the destination of the passenger is irrelevant when the taxi first goes to pick up the passenger, that the source is irrelevant once the taxi has picked up the passenger, and that the location of the passenger is irrelevant when the taxi is navigating to a preselected location. The HI-MAT induced hierarchy is exactly the same except that it has two navigation tasks for picking up and dropping off the passenger instead of the single parameterized Goto(l) task. Both task hierarchies encode strong policy constraints, such as prefixing the goal for navigation based on the passenger's information, and facilitate quicker convergence to the optimal policy.

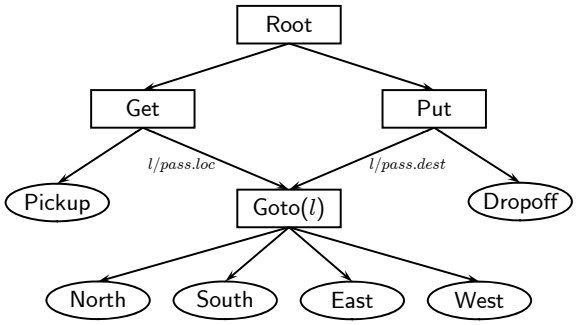


Figure 10: A manually-designed task hierarchy for the Taxi domain. The explicit binding for Goto's argument l to either the passenger's location (*pass.loc*) or destination (*pass.dest*) is part of the structural knowledge.

Figure 11 shows the performance of the three learners, Q, Manual, HI-MAT, in the Taxi domain along with their variants (suffixed with the phrase "with value") in which the value functions are initialized with optimal value functions learned in the source problem. We observe that the performance of the HI-MAT induced hierarchy converges to the optimal policy at a rate comparable to that of the manually-designed hierarchy. Although the source-target problem pairs in Taxi allow value-function transfer to occur, the target problems are still different enough that the agent has to "unlearn" the old policy through epsilon-greedy exploration. This leads to negative transfer in which transferring either the flat or the hierarchical value functions lead to worse rates of convergence to the optimal policy than transferring just the hierarchy structure with uninitialized policies (zeroed value functions) or even flat learning from scratch. Transferring the navigation policies from the source problem initially causes the agent to keep running into walls in the target problem. This indicates that transferring structural knowledge can be superior to transferring value functions, especially when the target problem differs significantly in terms of its optimal policy.

An explanation for the result is that an optimal policy is tuned to a particular instance of a domain. It will remain optimal in a very small set of related in-

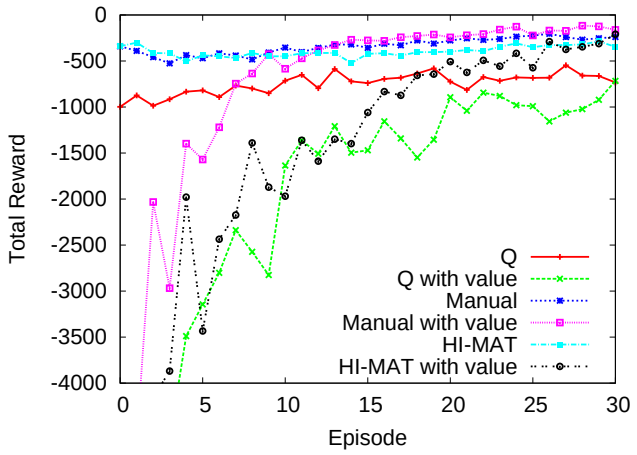


Figure 11: Performance in the target Taxi domain (averaged over 20 runs). Learning algorithms: the non-hierarchical RL algorithm Q-learning (Q), the hierarchical RL algorithm MAXQ-0 applied to the manually-designed hierarchy (Manual), MAXQ-0 applied to the HI-MAT induced hierarchy (HI-MAT), and their variants in which the optimal value function in the source is also transferred along with the structure, denoted by the phrase "with value".

stances, because it depends intimately on the transition probabilities and the immediate reward values of the instances. Consequently, the corresponding optimal value function has very limited transferability. On the other hand, a task hierarchy embodies domain knowledge at a more abstract level than the value function and is more broadly applicable across instances of the domain. Given a transferred task hierarchy, the hierarchical policy can then be optimized individually for the target problems.

Related Work on Transfer

Traditionally, reinforcement learning has focused on learning in the context of the single task or planning domain at hand. Transfer learning goes beyond this by seeking to exploit the similarities between different tasks and transferring knowledge learned in one task to another. Various elements in reinforcement learning are amenable to knowledge transfer including value functions, policies, models of system dynamics, and task hierarchies (as we have considered in our work). These elements lie along the spectrum going from transferring detailed knowledge with a narrow scope to more abstract knowledge with a broader scope of transfer. Transferring detailed knowledge, when possible, leads to huge speedup, but transferring more abstract knowledge leads to transfer across a broader range of tasks.

Value functions or policies transfer to a new task only when the target task is almost identical to the source task in terms of the action dynamics and rewards. For instance, (1) the value function can be transferred as an auxiliary reward signal for "reward shaping" when part of the state space is identical (Konidaris and Barto 2006), (2) compact policy constraints (such as when the agent should pass versus shoot in soccer-like games) can be employed for advice in the target task (Torrey et al. 2007; Taylor and Stone 2009), or (3) previously learned policies can be leveraged as macro actions when exploring in a new task (Fernández and Veloso 2006).

Task hierarchies are transferable to target tasks that are different from the source in terms of action dynamics or rewards, as long as the qualitative causal structure and variable dependencies of actions are preserved. There have been several approaches to induce hierarchical task structure in RL including HEXQ which uses a heuristic based on relative rates of change of state variables in trajectories (Hengst 2002), and VISA which learns the structure only from DBN models (Jonsson and Barto 2006). Our approach takes advantage of the DBN models as well as the trajectories to produce more compact hierarchies which result in better transfer. Extracting causal structure from the trajectories to discover the task dependencies is common to the EBL approach adopted by other systems (Tadepalli and Dietterich 1997; Nejati, Langley, and Konik 2006). However, by relying on only the qualitative structure of the domain dynamics and abstracting away the detailed quantitative model, our approach learns more widely transferable knowledge than these other approaches. Due to the generality of the transferred knowledge, the learner does require further experience in the target task to perform optimally. Besides being transferred as structural knowledge, task hierarchies also facilitate modular value-function transfer between two dissimilar RL problems that share common subtasks even though the overall flat value function may not transfer (Mehta et al. 2008a).

All the transfer methods discussed above require some correspondence between the state representations of the source and target tasks. In some cases, the transferred knowledge is based on an agent-centric representation that is common across all the tasks (Mehta and Tadepalli 2005; Konidaris and Barto 2006). In other cases, an explicit mapping between state variables is either provided or learned (Taylor and Stone 2009). In our work, we employ an agent-centric representation and allow for scaling of the domains of the state variables from the source to the target task. This is possible because the state abstractions of the subtasks only specify which variables are involved, and the hierarchical value function can be relearned for different variable domains. In contrast, relational reinforcement learning addresses transfer via a higher-order generalization of the state space. Here, the world is represented as a first-order MDP, where the states are represented via predicates in first-order logic. The relational value function or policy generalizes to all grounded instances of the first-order MDP by virtue of the relational representation (Wang, Joshi, and Khardon 2008).

Conclusion and Future Work

We presented an approach to automatically induce task hierarchies from source problems and transfer these hierarchies to related target problems that share the causal structure of the system dynamics. Given action models, our method, HI-MAT, analyzes the causal and temporal relationships among the actions in a successful trajectory and partitions the trajectory recursively into a task hierarchy. We show that these induced hierarchies perform comparably to manually-designed hierarchies and provide more effective transfer than direct transfer of the value function.

There are several future directions to pursue. Our method assumes that the given trajectory can be parsed as the execution trace of a hierarchical policy. In some domains, there are many good policies, only some of which will exhibit hierarchical structure (e.g., Kaelbling (1993)), so a trajectory extracted from a learned non-hierarchical policy is unlikely to possess this structure. In other domains, hand-designed hierarchies exploit parameter bindings in the subtask hierarchy to encode non-Markovian intentions (e.g., Parr and Russell (1995)), which greatly improves the compactness of the policies. An important open question is how to develop algorithms for discovering such hierarchical policies.

Our work has dealt with near transfer where the source and target problems share the causal structure of their domain dynamics. Although we cannot expect positive transfer between arbitrary pairs of unrelated domains, people are able to effectively transfer between much more disparate domains than what computers are able to do at present. We need to be able to transfer between increasingly disparate domains by suitably constraining the search to find an effective mapping between the state variables and actions of the two domains, for example by using analogical reasoning (Hinrichs and Forbus 2007). In our work, the structure of the task hierarchy is transferred without alteration. However, the more different the target task, the less this is likely to be useful. Various adaptations might be necessary before the hierarchy is plugged into the target problem. For instance, the termination conditions or state abstraction might need to be changed, or the task structure might need some tweaking. We could even try transferring parts of the induced task hierarchy when the entire hierarchy fails to transfer.

Acknowledgments

We thank Mike Wynkoop for many productive discussions. We are also grateful to the meticulous reviewers for their thoroughly helpful feedback. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-05-2-0249 and the Army Research Office (ARO) under Contract No. W911NF-09-1-0153. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the Air Force Research Laboratory (AFRL), ARO, or the US government.

References

Barto, A., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(4):341–379.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in Bayesian networks. In *Conference on Uncertainty in Artificial Intelligence*.

Dean, T., and Kanazawa, K. 1990. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

Dietterich, T. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Fernández, F., and Veloso, M. 2006. Reusing and building a policy library. In *International Conference on Automated Planning and Scheduling*.

Hengst, B. 2002. Discovering hierarchy in reinforcement learning with HEXQ. In *International Conference on Machine Learning*.

Hinrichs, T., and Forbus, K. 2007. Analogical learning in a turn-based strategy game. In *International Joint Conference on Artificial Intelligence*.

Hogg, C.; Kuter, U.; and Munoz-Avila, H. 2009. Learning hierarchical task networks for nondeterministic planning domains. In *International Joint Conference on Artificial Intelligence*.

Jonsson, A., and Barto, A. 2006. Causal graph based decomposition of factored MDPs. *Journal of Machine Learning Research* 7:2259–2301.

Kaelbling, L. 1993. Hierarchical reinforcement learning: Preliminary results. In *International Conference on Machine Learning*.

Konidaris, G., and Barto, A. 2006. Autonomous shaping: Knowledge transfer in reinforcement learning. In *International Conference on Machine Learning*.

Langley, P., and Choi, D. 2006. Learning recursive control programs from problem solving. *Journal of Machine Learning Research* 7:493–518.

Mehta, N., and Tadepalli, P. 2005. Multi-agent shared-hierarchy reinforcement learning. *International Conference on Machine Learning Workshop on Rich Representations in Reinforcement Learning*.

Mehta, N.; Natarajan, S.; Tadepalli, P.; and Fern, A. 2008a. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning* 73:289–312.

Mehta, N.; Ray, S.; Tadepalli, P.; and Dietterich, T. 2008b. Automatic discovery and transfer of MAXQ hierarchies. In *International Conference on Machine Learning*.

Minton, S. 1988. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic Publishers.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:929–935.

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *International Conference on Machine Learning.*

Parr, R., and Russell, S. 1995. Approximating optimal policies for partially observable stochastic domains. In *International Joint Conference on Artificial Intelligence.*

Reddy, C., and Tadepalli, P. 1997. Learning goal decomposition rules using exercises. In *International Conference on Machine Learning.*

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction.* MIT Press.

Tadepalli, P., and Dietterich, T. 1997. Hierarchical explanation-based reinforcement learning. In *International Conference on Machine Learning.*

Taylor, M., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10:1633–1685.

Torrey, L.; Shavlik, J.; Walker, T.; and Maclin, R. 2007. Relational macros for transfer in reinforcement learning. In *International Conference on Inductive Logic Programming.*

Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research* 31:431–472.

Wynkoop, M., and Dietterich, T. 2008. Learning MDP action models via discrete mixture trees. In *European Conference on Machine Learning.*

Neville Mehta is a Ph.D. candidate at Oregon State University. Besides dabbling in various facets of machine learning, his primary research interests include automating the discovery of models and hierarchical structure for efficient planning and reinforcement learning. In his spare time, he tries not to apply his research to playing the guitar or tennis.

Soumya Ray is an assistant professor in the department of Electrical Engineering and Computer Science at Case Western Reserve University. His research interests are in developing methods for statistical machine learning and reinforcement learning, and applications of these methods to challenging practical problems.

Prasad Tadepalli is Professor of Computer Science in Oregon State University. He is an expert in reinforcement learning, computational learning theory, relational learning, and probabilistic planning. He is an action editor of the *Machine Learning* journal and an associate editor of the *Journal of Artificial Intelligence Research.* His interests include learning to plan, learning from texts, structured machine learning, and integration of learning and inference.

Thomas Dietterich (AB Oberlin College 1977; MS University of Illinois 1979; PhD Stanford University 1984) is Professor and Director of Intelligent Systems Research in the School of Electrical Engineering and Computer Science at Oregon State University, where he joined the faculty in 1985. Dietterich developed the MAXQ framework for Hierarchical Reinforcement Learning as well as an early application of reinforcement learning to job shop scheduling. He is a Fellow of the AAAI, ACM, and AAAS. He has served as Technical co-Chair of AAAI-90, Technical Chair of NIPS-2000, General Chair of NIPS-2001, and first President of the International Machine Learning Society, which organizes the International Conference on Machine Learning. His research interests span machine learning, AI systems, and applications in ecological research and ecosystem management.