Automatic Induction of MAXQ Hierarchies

Neville Mehta, Mike Wynkoop, Soumya Ray, Prasad Tadepalli, and Tom Dietterich
School of EECS, Oregon State University

Scaling up reinforcement learning to large domains requires leveraging the structure in the domain. Hierarchical reinforcement learning has been one of the ways in which the domain structure is exploited to constrain the value function space of the learner, and speed up learning[10, 3, 1]. In the MAXQ framework, for example, a task hierarchy is defined, and a set of relevant features to represent the completion function for each task-subtask pair are given [3], resulting in decomposed subtask-specific value functions that are easier to learn than the global value function.

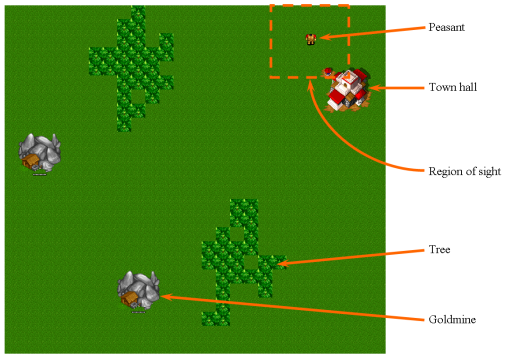
The MAXQ decomposition facilitates learning separate value functions for subtasks. The task hierarchy is represented as a directed acyclic graph. The leaf nodes are the primitive subtasks. Each composite subtask defines a semi-Markov Decision Process (SMDP) with a set of actions (which may include primitive actions or other subtasks), a set of state variables, a termination predicate which defines a set of exit states for the subtask, and a pseudo-reward function defined over the exits.

Several researchers have focused on the problem of automatically inducing temporally extended actions and task-subtask hierarchies [4, 7, 8, 9, 2, 11, 6, 5]. Discovering task-subtask hierarchies automatically is attractive for at least two reasons. First, it avoids the significant human effort in engineering the task-subtask decomposition, state abstractions, and task-specific rewards. 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 the hierarchies itself can be amortized over several domains. There have been some successful approaches to learning task hierarchies automatically, for example, `VISA`[6] and `HEXQ` [5]. These approaches define tasks around changing the values of state variables. `HEXQ` uses a heuristic that relies on the differences in the frequencies of value changes in state variables to determine the task-subtask relationships. The most frequently changing variable is attached to the lowest-level subtask and is consequently learned first, followed by less frequently changing variables. The `VISA` algorithm uses dynamic Bayesian network (DBN) action models to analyze the effects of actions on state variables. The variables are grouped into clusters such that there is an acyclic influence relationship between the values of variables in different clusters (strongly connected components). This naturally defines a task-subtask hierarchy, where the state variables whose values influence the values of other state variables are assigned to lower-level subtasks. The `VISA` algorithm provides a more principled rationale for `HEXQ`'s heuristic.

In this paper, we describe a hierarchy learning algorithm that uses given DBN action and reward models as well as a single successful solution of a problem. Such a solution might be obtained from a demonstration by a teacher, or from having previously solved similar tasks. Our approach resembles the `VISA` algorithm in that we rely on the DBNs to extract the task-subtask relationships and the appropriate abstraction for each subtask. However, unlike `VISA`, our analysis of the causal relationships between variables uses the observed solution trajectory. This could allow us to learn more refined hierarchies in two possible ways. First, the hierarchies we learn decompose the global problem only into subproblems that were observed in the solution trajectory, rather than all possible subproblems. This leads to more compact subtask hierarchies. Second, by using the solution trajectory, we are

able to perform a more refined analysis of the causal relationships between state variables than just using the action DBNs. For example, while analyzing the action DBNs might lead us to conclude that one variable possibly influences another, the corresponding action(s) may never be part of the observed solution trajectory. In this case, the influence can be ignored, leading to stronger abstraction. Moreover, `VISA` induces *exit-option* hierarchies for which value-function decomposition is not applicable.

Our application domain is the real-time strategy game Wargus. We describe it here to help explicate our algorithm. The core theme of Wargus is multiple teams/tribes competing for dominion over a geographical area (map). Each team is controlled by a player, and the goal is to obliterate every other player's team from the map. In order to satisfy this objective, the players must build strong economies that can, in turn, support a powerful military capable of vanquishing their opponents.



(a) Wargus resource-collection map.

State variables
Peasant location: `a.l`
Peasant resource: `a.r`
Gold mine within sight radius: `reg.gold`
Trees within sight radius: `reg.wood`
Town hall within sight radius: `reg.townhall`
Required gold quota: `req.gold`
Required wood quota: `req.wood`

Primitive actions
Mine gold: `MG`
Chop wood: `CW`
Deposit: `Dep`
Navigate: `Goto(loc)`

(b) State and action spaces.

Figure 1: Wargus resource-gathering domain.

Maps have natural resources such as goldmines and forests from which usable resources such as gold and wood can be harvested by the peasant units. Building a unit such as a new peasant or footman, or even a structure such as a town hall requires a certain amount of the collected gold, wood, and supply (food provided by the farms). In our experiments, we focus on the resource-gathering aspect of the game. Figure 1 shows a typical resource-gathering Wargus map with a single peasant, and the corresponding state and action spaces. We circumvent the issue of dealing with numeric goals (e.g., the actual quotas of gold or wood required) by using binary variables instead (signifying the fulfillment of the quotas). The provided map has forests, goldmines, and a town hall (collection point). The structure of the task hierarchy learned in one such map could be transferred over to another in which the locations of the entities within the map are all shuffled around. Note that the value function itself does not transfer to a different map.

Our approach to learning MAXQ hierarchies focuses on 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. Our algorithm has three components. We first annotate the given trajectory using causal links between the actions corresponding to the setting and checking of variables. Second, we use these causal annotations to segment the trajectory into subtasks and associate them with appropriate terminate conditions.

Finally, we use the DBN models, reward definitions, and termination conditions to define the appropriate relevant variables for each subtask. We recursively process each segment until a termination condition based on state abstraction is met. We provide some details on each step below.

To annotate the trajectory with causal information, we determine the set of relevant and irrelevant variables for each action. 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, and it is *irrelevant* otherwise. Given a trajectory, the set of *trajectory-relevant* (t-relevant) variables to an action A is the subset of the relevant variables that were actually checked or changed when A was executed in the trajectory. A *causal edge* labeled with a variable v connects two actions A and B (B later in the sequence than A) iff v is t-relevant to both A and B and irrelevant to all actions in between A and B. A *causally annotated trajectory* (CAT) is a pair (T, C) where T is a trajectory and C is the set of all causal edges in T. While performing this annotation, we ignore variables that cannot affect the goal conjunction (such variables can be found using a static analysis of the action DBNs). A sample CAT for the Wargus domain is shown in figure 2.

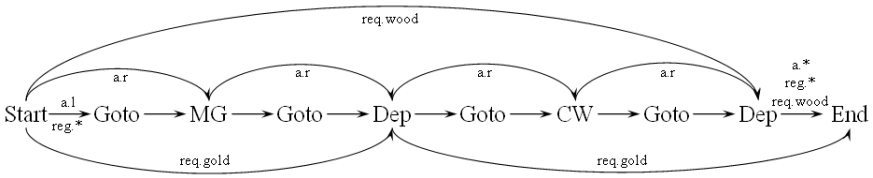


Figure 2: Causally annotated trajectory (CAT) for the Wargus resource-gathering domain.

After the CAT is constructed, we recursively decompose it into subtasks as follows. We maintain an unsolved goal list which is initially populated with the literals in the goal conjunction. At each step, we remove an arbitrary literal from this list and find the last action that satisfies the goal literal under consideration. We then find the longest sequence of actions immediately prior to and including the last action such that no causal edge leaves the sequence. The idea is to find all actions that are "responsible" for the specific literal being considered. The partition of the trajectory thus found is our initial subtask definition. Once the current trajectory has been partitioned into subtasks, we recursively call the procedure on each of the found subtasks at this level. A subtask is not partitioned further if the abstraction does not increase, i.e., if the child task does not have fewer relevant variables than the parent.

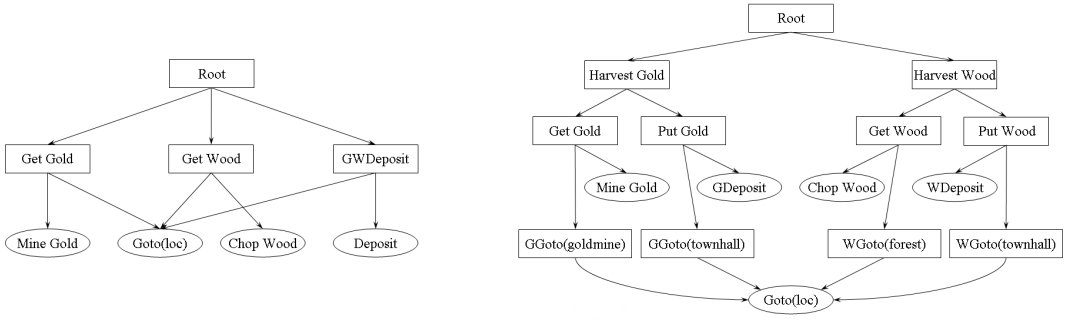
After finding the partition that constitutes a subtask, we assign this subtask a set of actions and a termination predicate. In both cases, we use the DBNs to generalize from the observed trajectory, since we want the subtask to be applicable to state-action pairs not in the observed trajectory. To assign the termination condition for this subtask, we use the relational test(s) in the action and reward DBNs satisfied by the variables on the causal edges leaving the subtask. To determine if the set of primitive actions available to the subtask should be expanded, we create a merged DBN structure using the observed primitive actions. The merged DBN represents possible variable effects after any sequence of the observed primitive actions. Next, for each primitive action DBN that we did not see in this trajectory, we create a merged DBN that represents a sequence of such actions. If this is a subgraph of the subtask DBN, we add this action to the set of actions available to this subtask. The rationale here is that the added action does not increase the set of

relevant variables. Finally, the set of nonterminal states for this subtask is the set of states from which the termination states are reachable using the generalized set of actions.

We can show that our induction procedure parses the entire input trajectory, and generates a unique decomposition at each (recursive) level. We can further establish some simple properties of the induced hierarchies. We can show that there exists some hierarchical policy which, when executed on the induced hierarchy, can (with nonzero probability) generate the trajectory that was used to induce the hierarchy. Next, we can show that our procedure only makes safe state abstractions. Finally, we can show that, if we count the total number of variables that are relevant to representing the value functions across all subtasks, this count does not decrease with any local change to our solution (a local change adjusts the boundaries between the subtask partitions found by our algorithm). In this sense, since no local change improves the state abstraction, the induced hierarchies are locally optimal.

For the experimental setup, we are interested in empirically verifying if the transfer of structural knowledge from one map to another helps speed up learning in the latter. To this end, we randomly generate multiple pairs of maps. We use a simple metric to assess the learning ability of the agent in any map, namely, the negative duration (in simulation time) of accumulating the requisite amounts of gold and wood [1]. Given a pair of maps, source and target, the transfer procedure is as follows:

1. The optimal policy for source is learned using Q-learning.
2. An optimal trajectory is generated for source from which a task hierarchy is induced (with the help of the DBN action models).
3. The induced task hierarchy is plugged into the MAXQ value-function learning algorithm for target.



(a) Manually engineered hierarchy.

(b) Induced hierarchy.

Figure 3: Wargus resource-gathering task hierarchies.

Comparing the manually engineered task hierarchy for this domain in figure 3(a) to the induced task hierarchy in figure 3(b), we observe that the induced hierarchy is a lot more detailed. The plot in figure 4 shows three learning curves for target: standard Q-learning (without transfer), MAXQ-learning given the manually engineered hierarchy, and MAXQ-learning given the induced hierarchy. Although there is less subtask sharing, the induced hierarchy allows the agent to learn much quicker (almost instantaneously!) because the induced hierarchy enforces much stricter policy constraints than does the manually

[1] A successful run in a map is defined as an episode, and learning the optimal policy normally takes several episodes.

engineered hierarchy. Nevertheless, these strict policy constraints are generic enough to be applicable across any such randomly generated resource-gathering map.

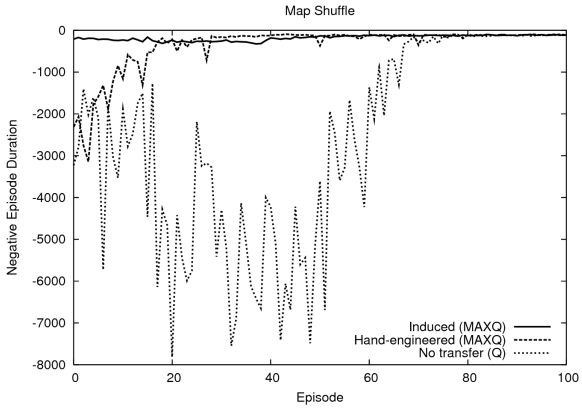


Figure 4: Empirical results in the Wargus resource-gathering domain.

In conclusion, this abstract presents an approach to automatically inducing MAXQ hierarchies from models and demonstrations. Transferring only structural knowledge across MDPs is shown to be a viable alternative to transferring the entire value function or learned policy itself. While we believe that this algorithm is promising, it has several limitations. In current work, we are extending it to handle disjunctive goals. Further, our current approach only handles goals of achievement. In future work, we plan to extend our approach to handling goals of maintenance as well.

References

[1] D. Andre and S. Russell. State Abstraction for Programmable Reinforcement Learning Agents. In *Proceedings of AAAI*, 2002.

[2] Ö. Şimşek and A. Barto. Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning. In *Proc. of the 21st International Conference on Machine Learning*, pages 751–758, 2004.

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

[4] B. Digney. Emergent Hierarchical Control Structures: Learning Reactive/Hierarchical Relationships in Reinforcement Environments. *From Animals to Animats*, 4:363–372, 1996.

[5] B. Hengst. Discovering Hierarchy in Reinforcement Learning with HEXQ. In *Proc. of the 19th International Conference on Machine Learning*, 2002.

[6] A. Jonsson and A. Barto. Causal Graph Based Decomposition of Factored MDPs. *Journal of Machine Learning Research*, 7:2259–2301, 2006.

[7] A. McGovern and A. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proc. of the 18th International Conference on Machine Learning*, pages 361–368, 2001.

[8] I. Menache, S. Mannor, and N. Shimkin. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. In *Proc. of the 14th European Conference on Machine Learning*, pages 295–306, 2001.

[9] M. Pickett and A. Barto. PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. In *Proc. of the 19th International Conference on Machine Learning*, pages 506–513, 2002.

[10] R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[11] S. Thrun and A. Schwartz. Finding Structure in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 385–392, 1995.