Knowledge Organization and Structural Credit Assignment

Joshua Jones and Ashok Goel
College of Computing
Georgia Institute of Technology
Atlanta, USA 30332
{jkj, goel}@cc.gatech.edu

Abstract

Decomposition of learning problems is important in order to make learning in large state spaces tractable. One approach to learning problem decomposition is to represent the knowledge that will be learned as a collection of smaller, more individually manageable pieces. However, such an approach requires the design of more complex knowledge structures over which structural credit assignment must be performed during learning. The specific knowledge organization scheme chosen has a major impact on the characteristics of the structural credit assignment problem that arises. In this paper, we present an organizational scheme called Externally Verifiable Decomposition designed to facilitate credit assignment over composite knowledge representations. We also describe an experiment in an interactive strategy game that shows that a learner making use of EVD is able to improve performance on the studied task more rapidly than by using pure reinforcement learning.

1 Introduction

The need for decomposition in learning problems has been widely recognized. One approach to making learning in large state spaces tractable is to design a knowledge representation composed of small pieces, each of which concerns a more compact state space than the overall problem. Techniques that would be intractable for the problem as a whole can then be applied successfully to each of the learning subproblems induced by the set of components.

Such composite knowledge representations, however, require the design of top-level structures that combine the knowledge that will be stored at individual components into a usable whole that encodes knowledge about the complete problem. These structures raise new issues for credit assignment. Specifically, there is a need to perform structural credit assignment over the top-level structure during learning.

The temporal credit assignment problem takes as input the outcome of a sequence of actions by an agent and gives as output a distribution over the actions in the sequence, where the output distribution specifies the relative responsibility of the actions for the outcome. In contrast, the structural credit assignment problem takes as input the outcome of a single action by an agent and gives as output a distribution over the components of the agent, where the output distribution specifies the relative responsibility of the components for the outcome. In this work, we are interested (only) in the structural credit assignment problem as it pertains to a learning agent's knowledge. In particular, we are interested in specifying and organizing knowledge components so as to enable accurate and efficient structural credit assignment over the resulting structure.

The question then becomes what might be the design principles for organizing knowledge and what additional knowledge might be encoded with each component to facilitate structural credit assignment? In this paper, we present an organizational and encoding scheme that we call Externally Verifiable Decomposition (or EVD). We also describe experimental results in an interactive strategy game, comparing reinforcement learning with EVD.

2 Externally Verifiable Decomposition

We begin by informally describing the EVD scheme with an illustrative example from an interactive strategy game called Freeciv (http://www.freeciv.org). We provide a formal description of EVD later in the paper.

2.1 Freeciv

FreeCiv is an open-source variant of a class of Civilization games with similar properties. The aim in these games is to build an empire in a competitive environment. The major tasks in this endeavor are exploration of the randomly initialized game environment, resource allocation and development, and warfare that may at times be either offensive or defensive in nature. Winning the game is achieved most directly by destroying the civilizations of all opponents. We have chosen FreeCiv as a domain for research because the game provides challenging complexity in several ways. One source of complexity is the game's goal structure, where clean decomposition into isolated subgoals is difficult or impossible. The game is also partially observable. The map is initially largely hidden, and even after exploration does not reflect state changes except in portions directly observed by a player's units. A consequence is that the actions of opponents may not be fully accessible. Also, the game provides a very large state space, making it intractable to learn to play well without decomposition.

It is not necessary to understand the game completely for the purpose of understanding this study, but some specifics are in order. The game is played on a virtual map that is divided into a grid. Each square in this grid can be characterized by the type of terrain, presence of any special resources, and proximity to a source of water such as a river. In addition, each square in the grid may contain some improvement constructed by a player. One of the fundamental actions taken while playing FreeCiv is the construction of cities on this game map, an action that requires resources. In return for this expenditure, each city produces resources on subsequent turns that can then be used by the player for other purposes, including but not limited to the production of more cities. The quantity of resources produced by a city on each turn is based on several factors, including the terrain and special resources surrounding the city's location on the map, the construction of various improvements in the squares surrounding the city, and the skill with which the city's operations are managed. As city placement decisions are pivotal to success in the game, an intelligent player must make reasoned choices about where to construct cities.

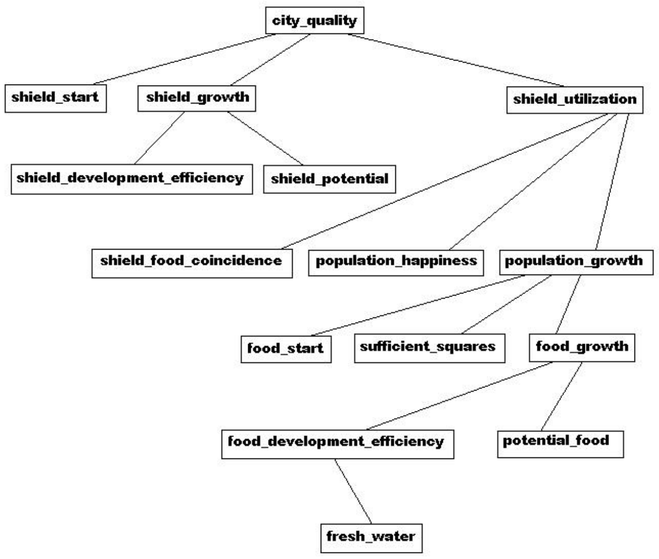

Figure 1: City Estimate Decomposition

2.2 Learner Design

We have designed an agent that plays FreeCiv. In this study, we are focused on evaluating EVD for a relatively small but still challenging part of the game. To that end, we have applied both EVD and Q-learning [Watkins, 1989] to a part of the module responsible for making decisions about city placement, specifically a part responsible for estimating the expected resource production over time if a city were built at a particular map location. This estimate is a state abstraction that can be used by a higher level reasoner to make decisions about where to place cities. In the experiment described in this paper, we are not concerned with the success of a higher level reasoner, but only in acquiring the knowledge needed to produce state abstractions that accurately project resource production.

EVD Learner Design

Informally, the EVD scheme for organizing an agent's decision-making process has the following characteristics (we provide a formal description of EVD later). Firstly, from a top-level perspective, knowledge is organized in a state-abstraction hierarchy, progressively aggregating and abstracting from inputs. Figure 1 illustrates the hierarchy used to apply EVD in our chosen problem setting. This decomposition is a simplification of actual game dynamics, but is sufficient for the purposes of this study. The output of this structure is one of nine values representing the expected resource production of the terrain surrounding the map tile represented by the inputs. Specifically, the value represents the (short and long term) estimated resource production on each turn relative to a baseline function. The baseline is computed as a function of the number of turns for which the city in question has been active. This baseline accounts for the fact that absolute resource production is expected to increase over time. Note that no single estimate value may be correct for a given set of inputs, so the goal of learning is to minimize rather than eliminate error in these estimates.

The leaf nodes in the hierarchy discretize raw features of the world. For example, *food_start* takes as input a raw perception available from the environment (a set of numbers) that represents the food production value of game tiles in the inspected area. This leaf node produces as output an integral value from 1 to 5, representing the food resources initially available to the prospective city in a form usable by the parent node. These discretization functions in leaf nodes are hard-coded in this study.

Nodes at any level higher than leaf nodes in the hierarchy view the outputs of child nodes as features and aggregate and abstract them into states at their level. These abstractions made at intermediate nodes are learned; that is, each node contains a learner (the specific type of which is not specified by EVD) capable of mapping the vector of inputs to an output value. In this work, we use a simple table-lookup procedure within each node to perform the mapping, where learning is performed via a simple scheme that gradually changes table entries as examples are presented. For example, the *population_growth* component contains a three dimensional table indexed by the output values of *food_start*, *sufficient_squares* and *food_growth*, and produces an integral output value from 1 to 5, abstractly representing the expected population growth for the prospective city.

The most important feature of EVD is the association of predictive knowledge with each non-leaf component. Specifically, this knowledge establishes failure (success) conditions for each output value available to the component with which it is associated. In general these conditions could indicate "hard" success or failure, or values along some gradient between success and failure, interpretable as severity of failure and/or confidence in local failure. Here, we deal only with digital success or failure. The predictive knowledge associated with a component can be thought of as encoding the semantics of the state abstraction knowledge acquired by the learner within the component.

This predictive knowledge is encoded in structural credit assignment functions, one of which is associated with each agent component. These functions are used to reflect upon the correctness of each component's action as information becomes available through perception subsequent to value production. The set of values produced by each component's function form an (unnormalized) distribution over the agent structure, as required for structural credit assignment. As an example, the *population_growth* node has an associated function that takes as input the actual population of a city, the number of turns that have passed since the city was built, and the value that was produced by the *population_growth* node's table during previous inference, and returns a boolean value indicating whether the value produced is correct in light of the first two parameters. This function was hand coded during the design phase, representing the intended semantics of the state abstraction to be produced by the node.

We term the property of a decomposition that is constructed to allow the encoding of this type of predictive knowledge "external verifiability" because each of the components is designed to have semantics that can be verified based on available percepts. Note that the technique is useful only when values produced by components can be verified in retrospect, when percepts become available after action selection – if they could be verified with certainty at the time values are produced, there would be no need for learning within components. Instead, the knowledge used for verification could simply be used to produce values directly.

This principle of external verifiability is used when designing a problem decomposition, guiding the choices that are made in terms of problem framing and the definition of agent components. The prescription of EVD is that these choices must be made so that the necessary predictive knowledge can be defined for each component. Beyond the facilitation of credit assignment during learning, this type of decomposition also has the advantage that agent components have meaningful, known semantics, allowing the results of learning to be easily interpreted.

Inference over an EVD structure is straightforward. In order to produce an output, each component recursively queries its children to obtain its inputs, and then uses its learner to map these inputs into an output value. This recursion is terminated at the leaves, at each of which a feature of raw input state is discretized and returned.

Q-Learning Agent Design

A separate Q-learning agent was also implemented for the purposes of comparison. The outputs of the EVD learner's input mappers are composed into a feature vector that is used as the input state description for the Q-learner. The set of values that can be produced as outputs by the EVD learner form the set of actions available to the Q-learner. This setup is intended to provide an I/O environment for the Q-learner that matches that of the EVD learner closely. Each time the action selected by the Q-learner corresponds to an incorrect resource production estimate for the current turn, a reward of -1 is given. If the action corresponds to a correct value for the current turn, a reward of 0 is given. Exploration is handled by initializing the value of all state-action pairs to 0, the highest value possible under this reward scheme.

3 EVD Credit Assignment Procedure

While predictive knowledge forms the basis for credit assignment over structures resulting from the application of EVD, it is not sufficient in and of itself. Predictive knowledge establishes failure conditions for each component. However, this knowledge does not capture the dependencies among agent components, and these dependencies are also significant in determining the failure status at each node. For this reason, the failure conditions used for credit assignment must incorporate both the local predictive knowledge at each component and some information about top-level knowledge structure.

This work focuses on decompositions with a hierarchical structure, that progressively aggregate and abstract from raw state features. For these state abstraction hierarchies, the observation necessary is that failure at a given component may be due either to erroneous knowledge (mapping from inputs to outputs) stored locally, or may be due to errors at the inputs arising from faults nearer the leaves. This structural characteristic, along with the functions enabling retrospective local verification when feedback becomes available, forms the basis for the credit assignment process associated with this decomposition technique.

Because the verification functions associated with each agent component may in principle be arbitrarily complex, it is advantageous to avoid evaluations whenever possible. In order to limit the number of evaluations required during each learning episode, we view incorrect behavior at a node as the production of a value that is both registered as a failure by the local verification function and that (recursively) results in incorrect behavior at the parent. Only the first condition is required for behavior at the root of the hierarchy to be viewed as incorrect. Viewing error in this way means that during some learning episodes, components that produce actions inconsistent with the local verification function may not be given weight by the credit assignment process, if the error is not propagated to the root of the hierarchy. The negative repercussion of this choice is that some opportunities for learning may not be fully exploited. However, if an action taken by some agent component is truly to be seen as incorrect, it is reasonable to require that there be some set of circumstances under which the erroneous action contributes to an overall failure. If such a situation exists, the credit assignment process will eventually recommend a modification at the component. If no such situation exists, it is safe to allow the component to continue with the "erroneous" behavior. The major benefit of this view is that when a component is found to have chosen an action that agrees with the local verification function, none of the components in the subtree rooted at the evaluated component need to be examined.

Given this view of error, the set of local verification functions associated with each component, and knowledge of the hierarchical structure of the decomposition, the structural

```
bool EVD_assign_credit(EVD d, percepts P)

  bool problem ← false

  if d→F(d→a, P) == 1
    return false
  end

  forall children c of d
    if EVD_assign_credit(c, P) == true
      problem ← true
    end
  end

  if !problem
    mark e
  end
  return true
end
```

Figure 2: Pseudo-code for EVD structural credit assignment. Each node has an associated structural credit assignment function as described above, denoted 'F' here. Each component is also expected to store its last action, 'a'.

credit assignment process is as shown in Figure 2. The function is called when new percepts become available from the environment (here, on each new turn), and results in marking the nodes identified as responsible for failure, if any. When invoked, the procedure evaluates the verification function at the root of the hierarchy based on the relevant value previously selected at that component. If the verification is successful, no further action is taken. If an error is indicated, each child is visited, where the procedure is recursively repeated. If and only if no error can be found at any child, the current node is marked as being in error. The base case for this recursive procedure is achieved by defining leaves in the hierarchy as correct; that is, inputs representing raw state are never considered to be a source of error, but are provided with "dummy" verification functions that yield 1 for all inputs. This procedure treats error as a digital property of agent components, not making distinctions in degree of failure. Notice that this procedure also makes the commitment that error is due either to local knowledge *or* to erroneous inputs. Also note that this credit assignment procedure is purely structural. That is, no temporal credit assignment is handled by this algorithm. For this reason, the percepts 'P' can be directly attributed to the last action 'a' taken by each component. This is why cached last action values can be used in the algorithm above. In order to address both structural and temporal credit assignment, the method described here could be used to distribute credit structurally after another technique has distributed credit temporally.

Based on the result of structural credit assignment, learning is performed at each node that has been marked as erroneous, by whatever procedure is applicable to the type(s) of learners used within the components. Note that the decomposition method and accompanying method for structural credit assignment make no prescription whatsoever in terms of the knowledge format or learning procedure that is used within each node. In this work, a simple table-update routine was used within each component. However, in principle any other type of learning technique desired could exist within each component.

4 Experiments

In order to provide evidence that the decomposition technique and associated structural credit assignment method outlined above provide advantages over learning in a flat problem space, we have applied the technique to a problem within a strategy game playing agent, and compared the results with an RL implementation, specifically Q-learning, as discussed previously.

4.1 Procedure

Because we train and evaluate the learners in an on-line, incremental fashion, we cannot apply the standard training set/test set approach to evaluation. Rather, we evaluate the learners' performance improvement during training by segmenting the sequence of games played into multi-game blocks, and comparing overall error rate between blocks. In this way, we are able to compare error rate around the beginning of a training sequence with the error rate around the end of that sequence.

Errors are counted on each turn of each game by producing a value (equivalently, selecting an action), finishing the turn, perceiving the outcome of the turn, and then determining whether the value produced correctly reflects the resource production experienced on that turn. If the value is incorrect, an error is counted. Note that this error counting procedure contrasts with another possibility; producing a value only at the beginning of each game, and counting error on each turn of the game based on this value, while continuing to learn on each turn. While this alternative more closely matches the intended *use* of the learned knowledge, we chose to instead allow a value to be produced on each turn in order to reflect the evolving state of knowledge as closely as possible in the error count. A negative consequence of this choice is that some overfitting within games may be reflected in the error count. However, a decrease in error rate between the first and last block in a sequence can be seen as evidence of true learning (vs. overfitting), since any advantage due to overfitting should be as pronounced in the first group of games as in the last. Also note that error counting was consistent for both the EVD-based learner and the Q-learner.

In each trial, a sequence of games is run, and learning and evaluation occurs on-line as described above. The EVD-based learner is trained on sequences of 175 games, while the Q-learner is allowed to train on sequences of 525 games. We trained the Q-learner on sequences three times longer than those provided to the EVD learner to determine whether the Q-learner's performance would approach that of the EVD learner over a longer training sequence. As described above, we segment these sequences of games into multi-game blocks for the purpose of evaluation; the block sized used is 7 games. Each game played used a (potentially) different randomly generated map, with no opponents. The agent always builds a city on the first occupied square, after making an estimate of the square's quality. Building in the first randomly generated occupied square ensures that the learners will have opportunities to acquire knowledge in a variety of states. Though this

	EVD agent	Q-learning agent	
	7^{th} block	7^{th} block	21^{st} block
Without city improvements	24%	(4%)	1%
With city improvements	29%	7%	10%

Table 1: Average percent decrease (or increase, shown in parentheses) in error for decomposition-based learning implementation from block 1 to 7, and for the Q-learning agent from block 1 to blocks 7 and 21.

setup is simpler than a full-fledged game, it was sufficient to illustrate differences between the learners. In order to compensate for variation due to randomness in starting position and game evolution, results are averaged over multiple independent trial sequences. Each result for the EVD learner is an average of 60 independent trials. Each result for the Q-learner is an average over 25 independent trials; each trial is time consuming, as each trial for the Q-learner is three times as long as for the EVD-learner, and it did not seem likely that further trials with the Q-learner would offer significantly more information.

To compare the speed with which learning occurs in the two agents, we ran two separate sets of trials. The first set of trials was run in an environment where no city improvements were constructed in the area surrounding the city. The second set of trials did allow for the construction of city improvements, but had an identical environment in all other ways. For each set of environmental conditions, we measure the quality of learning by comparing the average number of errors counted in the first block of the sequences to the number of errors counted in the last block. In the case of the Q-learner, we make two comparisons. The first compares error in the first block to the block containing the 175th game, illustrating decrease in error over the same sequence length provided to the EVD learner. We also compare error in the first block to error in the last block of the Q-learner's sequences, to determine whether the Q-learner's improvement will approach that of the EVD learner over sequences three times as long. We perform this evaluation separately for each of the two environmental setups.

4.2 Results

The results of the experiment described above are summarized in Table 1. The EVD based learner is able to produce a greater improvement in error rate in each case, as compared to the Q-learner, both after the same number of games and after the Q-learner has played three times as many games. For the two scenarios, the average improvement in error rate is 26.5%, compared to only 1.5% after the same number of training examples for Q-learning. The decrease in error across a typical sequence was not strictly monotonic, but did exhibit progressive decrease rather than wild fluctuation. Even after three times as many games had been played by the Q-learning agent, the decrease in error rate is significantly less than that achieved using EVD after only seven blocks. In one case, it appears that learning has not yielded an advantage in error rate in the Q-learning agent even after 525 games. Examining the complete set of results for intervening blocks does mitigate this impression to some extent, as an overall downward trend is observed, with some fluctuations. However, given that the fluctuations can be of greater magnitude than the decrease in error due to Q-learning, the learning that has been achieved after this number of games does not appear significant. Based on the significant difference in observed learning rate, these trials provide evidence that the decomposed structure and accompanying structural credit assignment capabilities of EVD do offer an advantage in terms of allowing learning to occur more quickly in a large state space.

5 Formal Description of EVD Structure

As has been discussed, this paper is focused on decompositions that progressively abstract away from raw state features through a hierarchy of components. Abstraction hierarchies can be viewed as handling a class of tasks, termed select-1-out-of-n, in a way that has been identified and formally described by Bylander, Johnson and Goel [Bylander *et al.*, 1991]. We base our formal description of EVD structure on this class of tasks. The select-1-out-of-n task is defined as follows:

Definition 5.1 *Let C be a set of choices. Let P be a set of parameters. Let V be a set of values. Let an assignment of values in V to the parameters P be represented by a function $d : P \to V$. Then let D be the set containing all possible parameter assignments d. The* select-1-out-of-n *task is defined as a tuple, $< P, V, C, s >$, where s is a function $s : D \to C$.*

In practice, each of the parameters in P may have a distinct set of legal values. In this case, V is the union of the sets of legal input values to each parameter $p \in P$, and D is restricted to contain only functions d that provide legal assignments of values to parameters.

Before formally defining EVD structure, we need to define an *input mapper*.

Definition 5.2 *An* input mapper *is defined as a tuple $< p, V, C, \mathcal{T} >$, where p is a single input parameter, V is the set of values that can be taken by the parameter, and C is the set of possible output values. $\mathcal{T}$ is a function $\mathcal{T} : V \to C$ that implements the translation of input values to the choice alphabet.*

Now we can formally define EVD structure.

Definition 5.3 *An* EVD *is recursively defined as a tuple $< P, V, C, \mathcal{P}, \mathcal{S}, \mathcal{L}, F >$, where P is the set of input parameters, V is the set of values that can be taken by those parameters, and C is the set of choices that form the output of the judgement. $\mathcal{P}$ is a tuple $< P_1, ..., P_r >$ such that $\{P_1, ..., P_r\}$ is a partition of the parameters P of rank r. That is, $P_1, ..., P_r$ are non-empty disjoint sets whose union is P. $\mathcal{S}$ is a tuple $< s_1, ..., s_r >$, where s_i is an* EVD *with parameters P_i, values V and choices C if $|P_i| > 1$, or an* input mapper *with parameter $p, p \in P_i$, values V and choices C if $|P_i| = 1$. $\mathcal{L}$ is an arbitrary learner with domain C^r and range C. F is a function $F : e \times C \to \{1, 0\}$, where e is a representation of feedback perceived from the environment.*

Once again, V represents the union of the values taken by all EVD parameters; value assignments and functions involving value assignments are restricted to handle legal assignments only. This treatment of V is for notational convenience. Similarly, some subtrees may return only a subset of C, and $\mathcal{L}$ at the parent node need not handle outputs of a subtree that cannot legally be produced. The function F encodes predictive knowledge about the knowledge encoded in the component, as described above.

Evaluation of an EVD is handled in two steps. First, determine the input to $\mathcal{L}$ by evaluating each $s_i \in \mathcal{S}$, and then produce as output the result of applying $\mathcal{L}$ to the generated input vector. Input mappers are evaluated by applying $\mathcal{T}$ directly to the value of the sole parameter p. Learning over the EVD structure as a whole proceeds by first using the credit assignment technique described previously, and then applying feedback to the learner within each EVD component as dictated by the outcome of credit assignment.

6 Discussion

The intended contribution of this work is in making explicit the connection between structural credit assignment and decomposition of learning problems via composite knowledge representation, and in describing an approach that drives the design of knowledge representations based on the needs of structural credit assignment. The connection between decomposition and structural credit assignment has been recognized by Dietterich in his work on hierarchical reinforcement learning, where he refers to the problem as hierarchical credit assignment [Dietterich, 1998]. However, the MAXQ method takes a different approach to decomposition that is not directly driven by the need to perform credit assignment over the resulting structure, and focuses on temporal rather than state abstractions.

Layered learning [Whiteson *et al.*, 2005] makes use of decomposition hierarchies to address large learning problems. In layered learning, each component's learner is trained in a tailored environment specific to the component. The EVD technique is more akin to what is called "coevolution" of components in work on layered learning, where all of the learners in the decomposition hierarchy are trained as a complete system in the actual target domain. Some notion of external verifiability is implicit in hierarchies built to be trained with coevolution, as the evaluation functions used for each component must be evaluable based on available percepts. Here, we are explicit about the need to design decompositions specifically around this property, we allow the use of arbitrary (possibly heterogeneous) learners within each component, and provide a procedure for credit assignment over the decomposition that can help to limit evaluations. An additional distinction is that EVDs focus on state abstraction. These decompositions aim to limit the number of inputs to each component, ensuring a learning problem of manageable dimensionality at each component. In contrast, layered learning focuses on temporal abstraction, where components responsible for selection of abstract actions are not necessarily shielded from the need to consider many raw state features.

Work on Predictive State Representations (PSRs) [Littman *et al.*, 2001] outlines rationale for using state representations that directly encode predictions about future events. In a general way, the notion that knowledge should have a predictive interpretation is central to this work as well. However, the specific problems of decomposition and structural credit assignment that motivate this work are not the focus of PSRs, and there are clearly significant differences between PSRs and the work described here.

This work is an extension of our previous efforts [Jones and Goel, 2004]. In addition to comparing the effectiveness of EVD with reinforcement learning over a flat representation, this paper extracts and begins to formalize the key design principles from our previous work in the hopes that these principles may be useful to researchers designing knowledge representations for learning in other domains.

7 Conclusions

This paper presents an approach to designing a composite knowledge representation for a learning agent that is directly driven by the needs to perform structural credit assignment over the resulting top-level structure. The experiment described here provides evidence that the approach and accompanying credit assignment technique are sufficient for learning, and that the decomposition increases the tractability of learning in a large state space. This means that if a problem framing and set of components can be defined according to the principle of external verifiability for a given problem, EVD-based knowledge structure and credit assignment may be used to accelerate learning. In principle, this type of decomposition should be compatible with a variety of learning techniques within each component, and even with a heterogeneous set of techniques. Such combinations have the potential to create learners for complex problems with varying characteristics.

References

[Bylander *et al.*, 1991] T. Bylander, T.R. Johnson, and A. Goel. Structured matching: a task-specific technique for making decisions. *Knowledge Acquisition*, 3:1–20, 1991.

[Dietterich, 1998] T. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Proc. 15th International Conf. on Machine Learning*, pages 118–126. Morgan Kaufmann, San Francisco, CA, 1998.

[Jones and Goel, 2004] J. Jones and A. Goel. Hierarchical Judgement Composition: Revisiting the structural credit assignment problem. In *Proceedings of the AAAI Workshop on Challenges in Game AI, San Jose, CA, USA*, pages 67–71, 2004.

[Littman *et al.*, 2001] M. Littman, R. Sutton, and S. Singh. Predictive representations of state, 2001.

[Watkins, 1989] C. J. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge university, 1989.

[Whiteson *et al.*, 2005] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving keepaway soccer players through task decomposition. *Machine Learning*, 59(1):5–30, 2005.