To Appear in *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, Boston, MA, July 2006.

Value-Function-Based Transfer for Reinforcement Learning Using Structure Mapping

Yaxin Liu and **Peter Stone**
Department of Computer Sciences, The University of Texas at Austin
{yxliu,pstone}@cs.utexas.edu

Abstract

Transfer learning concerns applying knowledge learned in one task (the source) to improve learning another related task (the target). In this paper, we use structure mapping, a psychological and computational theory about analogy making, to find mappings between the source and target tasks and thus construct the transfer functional automatically. Our structure mapping algorithm is a specialized and optimized version of the structure mapping engine and uses heuristic search to find the best maximal mapping. The algorithm takes as input the source and target task specifications represented as qualitative dynamic Bayes networks, which do not need probability information. We apply this method to the Keepaway task from RoboCup simulated soccer and compare the result from automated transfer to that from handcoded transfer.

Introduction

Transfer learning concerns applying knowledge learned in one task (the source) to improve learning another related task (the target). Human learning greatly benefits from transfer. Feasible transfer often benefits from knowledge about the structures of the tasks. Such knowledge helps identifying similarities among tasks and suggests where to transfer from and what to transfer. In this paper, we discuss how such knowledge helps transfer in reinforcement learning (RL) by using structure mapping to find similarities. Structure mapping is a psychological theory about analogy and similarities (Gentner, 1983) and the structure mapping engine (SME) is the algorithmic implementation of the theory (Falkenhainer, Forbus, & Gentner, 1989). SME takes as input a source and a target representated symbollically and outputs a similarity score and a mapping between source entities and target entities. To apply structure mapping to transfer in RL, we need a symbolic representation of the RL tasks, namely, the state space, the action space, and the dynamics (how actions change states). To this end, we adopt a qualitative version of dynamic Bayes networks (DBNs). Dynamic Bayes networks are shown to be an effective representation for MDP-based probabilistic planning and reinforcement learning (Boutilier, Dean, & Hanks, 1999; Guestrin *et al.*, 2003b; Kearns & Koller, 1999; Sallans & Hinton, 2004). Although the probabilities in DBNs are probably too problem-specific to be relevent for transfer, the dependencies represented as links are useful information. The qualitative DBN (QDBN) representation thus ignores probabilities but uses different types of links for different types of dependencies. In this paper, we specialize and optimize SME to work with QDBNs efficiently using heuristic search to find the best maximal mapping, since QDBNs typically involve at least an order of magnitude more entities than previous SME applications.

We recently proposed a value-function-based approach to transfer in reinforcement learning and demonstrated its effectiveness (Taylor, Stone, & Liu, 2005). This approach uses a transfer functional to transform the learned state-action value function from the source task to a state-action value function for the target task. However the transfer functional is handcoded, based on a handcoded mapping of states and actions between the source and the target tasks. As an application of the optimized SME for QDBNs, we generate the mapping of states and actions and thus the transfer functional automatically, using domain knowledge represented as QDBNs.

Prior work considering transfer using DBN models (Guestrin *et al.*, 2003a; Mausam & Weld, 2003) assumes that the source task is a planning problem represented as DBNs (thus with probabilities). We only requre a weaker model and consider source tasks that are reinforcement learning problems.

The main contribution of this paper is to use structure mapping to find similarities between the source and target tasks based on domain knowledge about these tasks, in the form of QDBNs in particular, and to automatically construct mappings of state variables and actions for transfer.

Value-Function-Based Transfer

We recently developed the value-function-based transfer methodology for transfer in reinforcement learning (Taylor, Stone, & Liu, 2005). The methodology can transform the state-action value function of the source task to the target task with different state and action spaces, despite the fact that value functions are tightly coupled to the state and action spaces by definition. We analyze this approach and propose a refined framework for value-function-based transfer. We start by reviewing some concepts and assumptions of temporal-difference reforcement learning (Sutton & Barto, 1998). Reinforcement learning problems are often formulated as Markov decision processes (MDPs) with unknown parameters. The system of interest has a set of states of the environment, S, and a set of actions the agent can take, A. When the agent takes action $a \in A$ in state $s \in S$, the system transitions into state $s' \in S$ with probability $P(s'|s,a)$, and the agent receives finite real-valued reward $r(s,a,s')$ as a function of the transition. In reinforcement learning problems, the agent typically knows S and A, and can sense its current state s, but does not know P nor r.

A policy $\pi : S \mapsto A$ is a mapping from states to actions. A temporal-difference reinforcement learning method gradually improves a policy until it is optimal, based on the agent's experience. Value-function-based reinforcement learning methods implicitly maintain the current policy in the form of a state-action value function, or a q-function. A

q-function $q : S \times A \mapsto \mathbb{R}$ maps from state-action pairs to real numbers. The value $q(s, a)$ indicates how good it is to take action a in state s. The q-function implicitly defines a policy π^q such that $\pi^q(s) = \arg\max_{a \in A} q(s, a)$. More precisely, the value $q(s, a)$ estimates the expected total (discounted or undiscounted) reward if the agent takes action a in state s then follows the implicit policy. The reinforcement learning agent improves the current policy by updating the q-function.

Consider the source task with states $\mathring{S}$ and actions $\mathring{A}$ and the target task with states S and actions A, where $\circ$ distinguishes the source from the target. The value-function-based transfer method (Taylor, Stone, & Liu, 2005) can deal with the general case $\mathring{S} \neq S$ and/or $\mathring{A} \neq A$ by using a transfer functional ρ that maps the q-function of the source problem, $\mathring{q}$, to a q-function of the target problem, $q = \rho(\mathring{q})$, provided that the rewards in both problems have related meanings. The transfer functional ρ is defined based on correspondences of states and actions in the source and target tasks, with the intuition that corresponding state-action pairs have values similar to each other. The transfer functional ρ is specific to the source-target problems pair and specific to the representation of q-functions. As demonstrated by (Taylor, Stone, & Liu, 2005), the functional ρ can be handcoded based on human understanding of the problems and the representation.

When the handcoded transfer functional is not readily available, especially for cross-domain transfer where straightfoward correspondences of states and actions do not exist, it is desirable to construct the functional automatically based on knowledge about the tasks. This paper introduces one approach to doing so by using (1) source and target task models represented as qualitative DBNs and (2) a structure mapping procedure specialized and optimized for QDBNs.

To do this, we assume that the state has a representation based on a vector of state variables (or variables for short), that is, $s = (x_1, \ldots, x_m)$. The q-functions are represented using variables, that is, $q(s, a) = q(x_1, \ldots, x_m, a)$. In this way, correspondences of states are reduced to correspondences of variables. For the purpose of transfer, we define correspondences of variables and actions between the source and target tasks to be a mapping ρ_X from target task variables to source task variables and a mapping ρ_A from target task actions to source task actions. They are mappings from the target task to the source task since the target task is typically more difficult and has more variables and actions, and thus one entity (variable or action) in the source task may correspond to several entities in the target task. The transfer functional ρ then is fully specified by mappings ρ_X and ρ_A, as well as a representation-related mapping ρ_R that transforms values of the q-functions at the representation level based on ρ_X and ρ_A. In fact, the work from (Taylor, Stone, & Liu, 2005) follows this two-step model of ρ: we first specify the mappings ρ_X and ρ_A which are the same for the transfer problem, and then specify different representation-specific mappings for different representations we used, such as CMACs, RBFs, and neural networks.

We therefore assume that the representations for q-functions are given for the source and target tasks and the representation-specific mapping ρ_R is known. The main technical result of the paper is that we can find the mappings ρ_X and ρ_A automatically using structure mapping given task models represented as QDBNs.

Qualitative Dynamic Bayes Networks

We consider problems whose state spaces have a representation with a finite number of variables and whose action spaces consist of a finite number of classes of actions (although each class can have an infinite number of actions or be a continuous action space). In such problems, a state is a tuple of values $s = (x_1, \ldots, x_m)$, where $x_j \in X_j$ for $j = 1, \ldots, m$, and X_j are sets of values of the variables. An action often affects only a small number of variables, and the values of other variables remain unchanged or change following a default dynamics not affected by the action. For example, for an office robot whose responsibilities include delivering mail and coffee, the variables are its position, whether it has the mail and/or coffee, to whom to deliver, and its power level. Moving around changes its position and power level but not what it has nor to whom to deliver, and picking up mail does not change its position. Dynamic Bayes networks (Dean & Kanazawa, 1989; Boutilier, Dean, & Hanks, 1999) capture such phenomena by using a two-tiered Bayes network to represent an action, where the first tier nodes indicate values of variables at the current time step (before the action is executed) and the second tier nodes indicate values of variables at the next time step (after the action is executed). Links in the network indicate dependencies among the values of variables. A link is diachronic if each node belongs to different tiers and synchronic if both nodes are in the same tier. The transition probabilities, $P(s'|s, a)$, if known, are represented as conditional probability tables. For reinforcement learning problems, the probabilities are unknown. However, the graphical structure of the network captures important qualitative properties of the problem, and can often be determined using domain knowledge (Kearns & Koller, 1999).

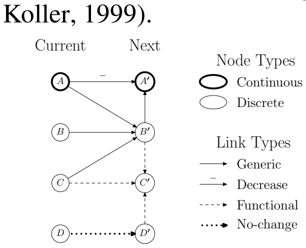


Figure 1: An example qualitative DBN

We define qualitative DBNs as an enhancement to the underlying graphical structure of DBNs by assigning types to nodes and links. These types can be defined in any convenient way to capture important qualitative differences of the nodes and links, for example, to indicate whether a node has continuous (the robot's power level) or discrete values (to whom to deliver), or to indicate types of dependencies such as no-change, increasing/descreasing, deterministic (functional) changes. The QDBN representation is similar to qualitative probabilisitic networks (Wellman, 1990) or causal networks (Pearl, 2000) in this aspect. An example QDBN is shown in Figure 1 along with its node and link types, where the next time step nodes are denoted with a prime. Generic links simply indicate dependency. We also use generic links when their types of dependency are unknown. A node can also be generic for the same reason. A descrease link exists only between nodes for the same variable at different time steps to indicate that the value decreases. A functional link indicates that the value of a node is a function of the values of its parents. For example, there exists a function f such that $C' = f(B', C, D')$. A no-change link indicate a special dependency between nodes for the same variable such that the value does not change. Note that with a slight abuse of notation, we use X_j to denote (1) the variable, (2) the set of values of the variable, and (3) nodes in QDBNs (X_j and X'_j) corresponding to the variable. Its meaning should be evident by context.

A qualitative model of a reinforcement learning problem consists of a set of QDBNs, one for each action (or each class of actions). Actions can also have types like nodes and links. Since we ignore (or are ignorant of) probabilities, this model is the same for all problems with the same set of variables and with actions of the same type. It is often not unreasonable to assume that we have the domain knowledge to specify QDBNs for the tasks at hand.

Structure Mapping for QDBNs

With knowledge about the source and the target tasks in the form of QDBNs, finding their similarities will help specify ρ_X and ρ_A, the mappings of variables and actions, for transfer. We do this using structure mapping. According to the structure mapping theory (Gentner, 1983), an analogy should be based on a system of relations that hold in both the source and the target tasks. Entities are mapped based on their roles in these relations, instead of based on their surface similarity such as names. The structure mapping engine (SME) is a general procedure for making analogies (Falkenhainer, Forbus, & Gentner, 1989) and leaves several components to be specified in particular applications. A (global) mapping between the source and target consists of a set of local matches. A local match $M = \langle \mathring{E}, E \rangle$ is a pair with a source entity $\mathring{E}$ and a target entity E, a global mapping is a set of consistent (to be explained shortly) local matches, and a maximal global mapping is a global mapping where no more local matches can be added. The SME procedure starts with local matches and gradually merges consistent local matches into maximal global mappings. Each global mapping is assigned a score and we seek a maximal global mapping with the highest score. The high-level steps of SME are shown in Algorithm 1.[1]

Algorithm 1 Structure Mapping Engine

1: generate local matches and calculate the conflict set for each local match;
2: generate initial global mappings based on immediate relations of local matches;
3: selectively merge global mappings with common structures;
4: search for a maximal global mapping with the highest score, using only global mappings resulting from Step 3;

The SME procedure specifies in detail how to check consistency and merge global mappings, but does not specify how to form local matches and initial global mappings, nor how to calculate similarity scores. The original SME also generates all possible maximal global mappings and then evalutes them in step 4. This approach does not work for QDBNs since QDBNs typically consist of many more entities than representations used in previous SME applications and thus generating all mappings results in combinatorial explosion. We next present a specialized and optimized version of SME for task models represented as QDBNs. It does not only specify the missing parts of the orginal SME but also optimizes existing steps for QDBNs. We therefore refer to it as the SME-QDBN method.

Step 1: Generate Local Matches Entities in QDBNs are variables, actions, and links. We allow only variables to match variables, actions to match actions, and links to match links. We also require that an entity can only match an entity of the same type or of the generic type. For example, a decrease link can match another decrease link or a generic link, but not a no-change link. In addition, a diachronic link matches only diachronic links and a synchronic link matches only synchronic links. Note that a node is not considered an entity for matching since it is only a replication of a variable. We generate all possible pairwise local matches.

Two local maps $M_1 = \langle \mathring{E}_1, E_1 \rangle$ and $M_2 = \langle \mathring{E}_2, E_2 \rangle$ are consistent iff $(\mathring{E}_1 = \mathring{E}_2) \Leftrightarrow (E_1 = E_2)$. In other words, SME enforces a 1-1 mapping of entities of the source and target tasks. The set of inconsistent local matches, or the conflict set, of a local match M, denoted as $M.\mathit{Conflicts}$, is calculated after all local matches are generated.

Step 2: Generate Initial Global Mappings Initial global mappings are formed based on relations that entities participate in for their respective tasks. Since relations are essential for structure mapping, this step encodes constraints from these domain-specific relations, in addition to the 1-1 constraints from step 1. For QDBNs, the central relation is that a link belongs to an action and connects a pair of variables (which may be the same but at different time steps). Each initial global mapping thus consists of a link match, an action match, and one or two variable matches. For example, if $\mathring{A}$, $\mathring{B}$ are variables in the source task such that the diachronic link $\mathring{A} \to \mathring{B}'$ belongs to action $\mathring{a}$ and A, B are variables in the target task such that the diachronic link $A \to B'$ belongs to action a, then an initial global mapping is

$$G = \left\{ \langle \mathring{A} \xrightarrow{\mathring{a}} \mathring{B}', A \xrightarrow{a} B' \rangle, \langle \mathring{A}, A \rangle, \langle \mathring{B}, B \rangle, \langle \mathring{a}, a \rangle \right\},$$

where we annotate the links with actions. Such relational constraints will rule out global mappings such as $\left\{ \langle \mathring{A} \xrightarrow{\mathring{a}} \mathring{B}', A \xrightarrow{a} B' \rangle, \langle \mathring{a}, a_1 \rangle \right\}$ where $a_1 \neq a$.

For a global mapping G, its set of inconsistent local matches, or local conflict set, is

$$G.\mathit{lConflicts} = \bigcup_{M \in G} M.\mathit{Conflicts}.$$

The set of all initial global mappings is denoted as $\mathcal{G}_0$. For later convenience, we define $G.V$, $G.A$, and $G.L$ to be the subset of G that contains only variable matches, action matches, and link matches, respectively. Since they are still global mappings, the notation such as $G.V.\mathit{lConflicts}$ is well-defined. Since G is a 1-1 mapping, we can use the notations $G(\mathring{E})$ to denote the matched target entity of $\mathring{E}$ and $G^{-1}(E)$ to denote the matched source entity of E.

Step 3: Selectively Merge Some Global Mappings Consistent global mappings can be merged into larger global mappings, which indicate more similarities than smaller ones. Two global mappings G_1 and G_2 are consistent iff

$$\left(G_1 \cap G_2.\mathit{lConflicts} = \varnothing\right) \wedge \left(G_1.\mathit{lConflicts} \cap G_2 = \varnothing\right).$$

The merged global mapping is simply $G = G_1 \cup G_2$ and its local conflict set is $G.\mathit{lConflicts} = G_1.\mathit{lConflicts} \cup G_2.\mathit{lConflicts}$.

We can form maximal global mappings directly using $\mathcal{G}_0$.

[1] We slightly altered the description of SME to better suit our presentation, but did not change how it works.

A direct approach requires trying a factorial number of combinations, a prohibitive procedure. A more tractable way is to use a guided search process in the space of all global mappings. However, the initial global mappings are too small and do not have much information about their potential for forming large global maps. Therefore the search essentially starts at random. To give the search a better starting position, we perform preliminary merges of the initial global mappings as follows.

First, for all global mappings $G \in \mathcal{G}_0$, we calculate

$$G.\mathit{Commons} = \left\{G' \in \mathcal{G}_0 \,\middle|\, G' \neq G, G \cap G' \neq \varnothing, \mathsf{consistent}(G, G')\right\}.$$

The set $G.\mathit{Commons}$ contains all consistent global mappings that share structures with G. We also calculate the global conflict set for G as

$$G.\mathit{gConflicts} = \left\{G' \in \mathcal{G}_0 \,\middle|\, \neg\mathsf{consistent}(G, G')\right\}.$$

Based on this information, we use the recursive merge algorithm shown in Algorithm 2.

Algorithm 2 Selectively merge initial global mappings

```
𝒢₁ ← ∅;
for all G ∈ 𝒢₀ do
    𝒢₁ ← BasicMergeRecursive(G, G.Commons, 𝒢₁);
```

$\mathcal{G}'$ = $\mathsf{BasicMergeRecursive}(G, \mathcal{C}, \mathcal{G})$

```
if 𝒞 = ∅ then
    return CheckedInsert(𝒢, G);
for all G′ ∈ 𝒞 do
    𝒢 ← BasicMergeRecursive(G ∪ G′, (𝒞 \ {G}) \ G′.gConflicts, 𝒢);
return 𝒢;
```

The merge starts with initial global mappings and uses consistent global mappings in their *Commons* sets as candidates. $\mathsf{BasicMergeRecursive}$ does the main work. The parameter $\mathcal{C}$ is the set of candidates to be merged with G, and $\mathcal{G}$ is used to collect all merged global mappings. We record the result when no more global mapping can be merged, and recurse otherwise using a reduced candidate set. $\mathsf{CheckedInsert}$ is a modified insertion procedure that avoids having two global mappings such that one contains or equals to the other. The merge generates another set of global mappings $\mathcal{G}_1$.

Step 4: Search for Best Maximal Global Mapping We start with the set of global mappings $\mathcal{G}_1$ to find the maximal global mapping with the highest similarity score, where a higher score indicates greater similarity. The original SME strategy of generating all maximal global mappings is intractable since the time complexity is factorial in the number of global mappings in $\mathcal{G}_1$. We instead use a depth-first heuristic search method to cope with the complexity. We first describe how to calculate the similarity scores. Suppose that the target task has m variables $X_1, X_2, \ldots, X_m$ and n actions $a_1, a_2, \ldots, a_n$. Let G_{a_i} be the restriction of G onto action a_i such that all members of G_{a_i} are relevent to a_i:

$$G_{a_i} = G.V \cup \left\{\langle \bullet, \bullet \xrightarrow{a} \bullet\rangle \in G.L \,\middle|\, a = a_i\right\},$$

where $\bullet$ indicates "don't care" values. We refer to G_{a_i} as an action mapping since it concerns only one action in the target domain (also one action in the source domain). Let G_a be an action mapping and let G_{a,X_j} be the restriction of G_a to node X_j such that all members of G_{a,X_j} are outgoing links from node X_j in the QDBN of a (thus a set of diachronic links):

$$G_{a,X_j} = \left\{\langle \bullet, X \xrightarrow{a} \bullet\rangle \in G.L \,\middle|\, X = X_j\right\}.$$

Similarly, let G_{a,X'_j} be the restriction of G_a to node X'_j (thus a set of synchronic links):

$$G_{a,X'_j} = \left\{\langle \bullet, X' \xrightarrow{a} \bullet\rangle \in G.L \,\middle|\, X = X_j\right\}.$$

We refer to G_{a,X_j} and G_{a,X'_j} as node mappings. We define the score of a global mapping as the sum of the scores of all valid node mappings, and the score of a node mapping as the ratio of the number of matched links to the number of total links in the source and target QDBN while counting matched links only once. Formally, let G_a be an action mapping of G and $G_{a,X}$ be a node mapping of G_a, and let $O(X, a)$ denote the set of outgoing links from node X in the QDBN of a. We have

$$\mathsf{score}(G) = \textstyle\sum_{i=1}^{n} \mathsf{score}(G[a_i])$$

$$\mathsf{score}(G_a) = \textstyle\sum_{j=1}^{m} \left(\mathsf{score}(G_a[X_j]) + \mathsf{score}(G_a[X_j{}'])\right)$$

$$\mathsf{score}(G_{a,X}) = \begin{cases} 1 & \text{if } O(G_a^{-1}(X), G^{-1}(a)) = O(X, a) = \varnothing \\ \dfrac{|G_{a,X}|}{\left|O(G_a^{-1}(X), G^{-1}(a))\right| + |O(X, a)| - |G_{a,X}|} & \end{cases}.$$

The score of a node mapping is between zero and one. The score is one if the source and target nodes match completely, and the score is zero if no links are matched at all. The score should satisfy an intuitive property of monotonicity, that is, if $G \subset G'$, then $\mathsf{score}(G) \leq \mathsf{score}(G')$. Note that if $G \subset G'$ then $G_{a_i} \subseteq G'_{a_i}$ for all $i = 1, \ldots, n$, and if $G_a \subseteq G'_a$ then $G_{a,X_j} \subseteq G'_{a,X_j}$ for all $j = 1, \ldots, m$. Therefore, to show that the property holds, it is sufficient to show that it holds for node mappings. The number of matched links cannot be greater than the number of outgoing links of the corresponding nodes in the source or the target tasks, that is,

$$|G_{a,X}| \leq \left|O(G_a^{-1}(X), G^{-1}(a))\right| \text{ and } |G_{a,X}| \leq |O(X, a)|.$$

Since for $x, y \geq 0$ with $\max(x, y) > 0$, and for z, z' satisfying $0 \leq z \leq z' \leq \min(x, y)$, it holds that

$$\frac{z}{x + y - z} \leq \frac{z}{x + y - z'} \leq \frac{z'}{x + y - z'},$$

the monotonicity property holds for node mappings, and thus for global mappings.

To find the maximal global mapping with the best score, we perform heuristic search in the space of all possible global mappings. Since the search objective is maximization and there are more choices earlier than later (since merging reduces the possible number of global mappings to merge), we choose to use a depth-first search strategy. Assume that we can also calculate an upper bound of the score of a global mapping. The search records the best score encountered and uses the upper bounds to prune: if the upper bound is lower than the current best score, we do not need to consider further merging. The search procedure is shown in Algorithm 3, where score^+ is the upper bound of the score.

Algorithm 3 Depth-first search

```
return DfsRecursive(∅, 𝒢₁, 0);
```

best' = $\mathsf{DfsRecursive}(G, \mathcal{C}, \mathit{best})$

```
if 𝒞 = ∅ then
    if score(G) > best then
        best ← score(G);
        record G as the best maximal global mapping;
    return bestScore;
for all G′ ∈ 𝒞 do
    if score⁺(G ∪ G′) > best then
        best ← DfsRecursive(G ∪ G′, (𝒞 \ {G′}) \ G′.Conflicts, best);
```

Now we define the upper bound of the score of a global mapping. First note that once the mapping of an action is fixed, merging another global mapping cannot change this mapping. Similarly merging does not change the mapping of a variable either. Therefore for an existing node mapping $G_{a,X}$, the inverse mappings $G^{-1}(a)$ and $G_a^{-1}(X)$ do not change. The upper bound of a global mapping can again be calculated by summing up the upper bounds of all node mappings. Since the numbers of outgoing links are fixed for the source and target nodes, the upper bound of an existing node mapping $G_{a,X}$ is reached by considering all unmatched outgoing links matched, except for two cases: (1) if an outgoing link does not have a match to any outgoing link in the other domain such that the match is consistent with the global mapping and (2) if the numbers of remaining outgoing links (not counting matched and inconsistent ones) are different and such we can only use the smaller number. The upperbounds for unmatched variables and actions are calculated similarly.

Constructing ρ Automatically

We now can find the mappings ρ_X and ρ_A automatically given task models represented as QDBNs. In fact, the SME-QDBN method does almost that, except that some target actions and variables are not mapped due to the 1-1 constraint of structure mapping. We fill this gap also using structure mapping. Since actions play a central role in reinforcement learning tasks, we first consider ρ_A. Suppose that the SME-QDBN method produces a global mapping G, but G does not contain a mapping for a target action a. We find $\rho_A(a)$, the corresponding source action, using the score for an action mapping. For source action $\mathring{a}$ and target action a, the mapping for variables of G, $G.V$, induces a mapping of links in $\mathring{a}$ and a as follows: two links are mapped together if they can form a local match (with consistent types) and their head and tail nodes (variables) are mapped together in $G.V$. This mapping of links together with $G.V$ forms an induced action mapping, denoted as $G_{a[\mathring{a}]}$. Notice that if $\langle \mathring{a}, a\rangle \in G$, then $G_{a[\mathring{a}]} = G_a$. We thus define

$$\rho_A(a) = \begin{cases} G^{-1}(a), & a \text{ appears in } G, \\ \arg\max_{\mathring{a}\in\mathring{A}} \mathit{score}(G_{a[\mathring{a}]}), & \text{otherwise.} \end{cases}$$

Let $\bar{G}$ be the global mapping "enhanced" by ρ_A and induced mapping of links. $\bar{G}$ is in fact not consistent under the 1-1 requirement of SME, but we can imagine the source task has a shadow action (and QDBN) for each additional appearance of a source action in $\bar{G}$, and then the 1-1 constraint is restored. After ρ_A is defined, the case for ρ_X is defined similarly. Since additional variables are more likely associated with additional actions, we take into account all target actions when defining ρ_X. Unlike actions, a variable affects all QDBNs of a task. We consider one QDBN at a time. For source variable $\mathring{X}$ and target variable X, the induced mapping of links is the same as that for actions with the additional variable match $\langle \mathring{X}, X\rangle$ added to $\bar{G}.V$. Let $\bar{G}_{a,X[\mathring{X}]}$ be the induced node mapping. We now can define the score of mapping target variable X to source variable $\mathring{X}$ based on G and ρ_A to be

$$\sum_{a\in A} \mathit{score}(\bar{G}_{a,X[\mathring{X}]}),$$

where the definition of the score for a node mapping remains the same. We then define $\rho_X(X)$ to be the maximizing source variable $\mathring{X}$, if X does not appear in G. Therefore, we can put the automatically constructed ρ_X and ρ_A together with the representation mapping ρ_R to form the transfer functional ρ. This approach is semi-automatic since ρ_R is still specified by hand.

Application to Keepaway

Now we apply this approach to the Keepaway domain and compare the automatically constructed transfer functional ρ_{auto} to the handcoded transfer functional ρ_{hand}. From (Taylor, Stone, & Liu, 2005), Keepaway is a subtask of RoboCup soccer and a publicly available benchmark for reinforcement learning (Stone *et al.*, 2006), in which one team of n_K keepers must try to keep possession of the ball within a small rectangular region, while another team of n_T takers try to take the ball away or force the ball out of the region. The keepers learn how to keep the ball and the takers follow a fixed strategy. The transfer problem of focus is to transfer from 3v2 Keepaway[2] to 4v3 Keepaway.

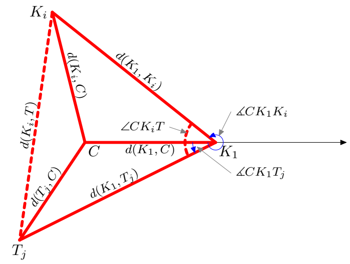


Figure 2: Variables for Keepaway

For our purpose of exposition, we focus on the actions and variables of Keepaway. Further details on Keepaway can be found in (Stone *et al.*, 2006) and the references therein. The keeper with the ball, referred to as K_1, can choose from **Hold** or **Pass**k to teammate K_k. There are 3 actions for 3v2 Keepaway and 4 for 4v3. The variables are distances and angles based on the positions of the players and the center of the playing region C (see Figure 2):

- $d(K_i, C)$ for $i = 1, \ldots, n_K$ and $d(T_j, C)$ for $j = 1, \ldots, n_T$;
- $d(K_1, K_i)$ for $i = 2, \ldots, n_K$ and $d(K_1, T_j)$ for $j = 1, \ldots, n_T$;
- $d(K_i, T) = \min_{j=1,\ldots,n_T} d(K_i, T_j)$ for $i = 2, \ldots, n_K$; and
- $\angle K_iK_1T = \min_{j=1,\ldots,n_T} \angle K_iK_1T_j$ for $i = 2, \ldots, n_K$,

where $\angle$ indicates angles ranging $[0°, 180°]$. There are 13 variables for 3v2 Keepaway and 19 for 4v3.

We specify QDBNs based on knowledge about soccer, the takers' strategy, and the behavior of other keepers' not with the ball. Two of the closest takers T_1 and T_2 always move toward the ball, and the remaining takers, if any, try to block open pass lanes. Therefore, if K_1 does **Hold**, K_1 does not move, T_1 and T_2 will move towards K_1 directly, other keepers try to stay open to a pass from K_1, and other takers move to block pass lanes; and if K_1 does **Pass**k, K_k will move towards the ball to receive it, K_1 will not move until someone blocks the pass lane, T_1 and T_2 now move towards K_k directly, and the remaining players move to get open (keepers) or to block pass (takers).

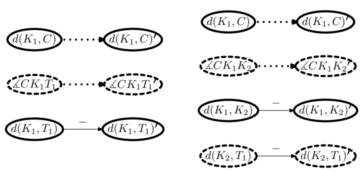
Unfortunately, we found that the current set of variables is not convenient for specifying the QDBNs, since (1) the set of variables is not complete as they cannot completely determine player positions and (2) changes in positions caused by actions cannot be directly described in these variables. We choose to add in some additional variables for the purpose of specifying QDBNs only. These variables are (also see Figure 2)

- $\measuredangle CK_1K_i$ for $i = 2, \ldots, n_K$ and $\measuredangle CK_1T_j$ for $j = 1, \ldots, n_T$;

[2] 3 keepers and 2 takers. Similarly for 4v3.

- $\measuredangle K_i K_1 T_j$ for $i = 2, \ldots, n_K$ and $j = 1, \ldots, n_T$; and
- $d(K_i, T_j)$ for $i = 2, \ldots, n_K$ and $j = 1, \ldots, n_T$,

where $\measuredangle$ indicates directed angles ranging $[0°, 360°)$. The players' positions are completely specified modulo rotation around C and the set of variables $\mathbf{X} = \{d(K_1, C), d(K_1, K_i), \measuredangle CK_1K_i, d(K_1, T_j), \measuredangle CK_1T_j\}$ with $i = 2, \ldots, n_K$ and $j = 1, \ldots, n_T$, is complete. We can use $\mathbf{X}$ instead of the original set of variables in learning algorithms. We however decide not to do that in favor of comparing directly with the handcoded transfer functional, which is defined for the original variables. Therefore, the additional variables are hidden to the learning algorithm and only the original ones are observable to the learning agent. Since $\mathbf{X}$ is complete, the original variables can be determined using $\mathbf{X}$ based on elementary geometry.



Hold **Pass**2

Figure 3: QDBNs for 2v1 Keepaway

Now we specify QDBNs for **Hold** and **Pass**k. First consider **Hold**. K_1 does not move and thus $d(K_1, C)$ is unchanged. T_1 and T_2 go toward K_1 directly, therefore $\measuredangle CK_1T_1$ and $\measuredangle CK_1T_2$ do not change and $d(K_1, T_1)$ and $d(K_1, T_2)$ decrease. The remaining players' moves are based on their relative positions and we also encode changes in the related variables in the QDBN. We omit the details due to space limits. For **Pass**k, we consider the next time step to be the point of time shortly after the ball is kicked. Therefore, K_1 does not move after the pass so $d(K_1, C)$ is unchanged. K_k moves toward the ball to receive it, and thus $\measuredangle CK_1K_k$ does not change but $d(K_1, K_k)$ decrease. T_1 and T_2 move towards K_k and therefore $d(K_k, T_1)$ and $d(K_k, T_2)$ decreases. The remaining players move in the same way as in the case of **Hold** and we encode them in the QDBN the same way. For illustration purposes, Figure 3 shows interesting parts of the QDBNs in 2v1 Keepaway (dashed ovals indicate hidden variables), while the full QDBNs are too complex to be included.

We perform SME-QDBN on various sizes of Keepaway up to 4v3 starting from 2v1. We consider distances and angles as different and we also distinguish observable and hidden variables. Thus we have four types of variables and only variables of the same type match. We also have five types of links: no-change, decrease, functional, minimum, and generic. We show the similarity scores from SME-QDBN in Table 1. To compare results for different target tasks, we normalize the scores to be in $[0, 1]$ by dividing them by $2mn$, where m is the number of variables and n is the number of actions. Notice that the normalized score is not symmetric since we normalize using parameters for the target task. We also include self mapping to test the algorithm. SME-QDBN takes less than half a minute on a Dell 360n running Linux 2.6 to find the optimal mapping for 2v1 to 2v2, and is also quite fast for self mapping. However for larger problems such as 3v2 to 4v3, it takes more than a day to complete.[3]

We also determine ρ_A and ρ_X for 3v2 to 4v3 transfer. The result is however too large to be included. We observed that the result containing observable variables is exactly the same as the handcoded one from (Taylor, Stone, & Liu, 2005). For this reason, further experiments with Keepaway are not necessary since the successful transfer results from there apply directly.

[3]In fact, we performed additional optimizations to obtain the aforementioned running time.

Table 1: Similarity scores for Keepaway

Source	#var	#link	2v1	2v2	3v2	3v3	4v3
2v1	11(7)	26,26	1.00	0.63	0.26	0.17	0.09
2v2	16(9)	42,46	0.92	1.00	0.38	0.24	0.13
3v2	25(13)	77, 81, 81	0.87	0.75	1.00	0.69	0.36
3v3	32(15)	124,124,124	0.76	0.69	0.88	1.00	0.51
4v3	43(19)	178,178,178,178	0.74	0.67	0.79	0.88	1.00

Conclusion

In this paper, we propose to use structure mapping to study transfer in reinforcement learning. This is possible since important information about the domain can be captured using qualitative DBNs. Structure mapping can then find similarities based on QDBNs and then mappings of state variables and actions between the source and target tasks. Therefore we can automate the construction of the transfer functional for value-function-based transfer in reinforcement learning.

Acknowledgements

This work was supported by DARPA grant HR0011-04-1-0035 and NSF CAREER award IIS-0237699.

References

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5:142–150.

Falkenhainer, B.; Forbus, K. D.; and Gentner, D. 1989. The structure mapping engine: Algorithm and examples. *Artificial Intelligence* 41:1–63.

Gentner, D. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive Science* 7:155–170.

Guestrin, C.; Koller, D.; Gearhart, C.; and Kanodia, N. 2003a. Generalizing plans to new environments in relational MDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003b. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19:399–468.

Kearns, M. J., and Koller, D. 1999. Efficient reinforcement learning in factored MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, 740–747.

Mausam, and Weld, D. S. 2003. Solving relational MDPs with first-order machine learning. In *Proceedings of the ICAPS-03 Workshop on Planning under Uncertainty and Incomplete Information*.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Sallans, B., and Hinton, G. E. 2004. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research* 5:1063–1088.

Stone, P.; Kuhlmann, G.; Taylor, M. E.; and Liu, Y. 2006. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup-2005: Robot Soccer World Cup IX*.

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

Taylor, M. E.; Stone, P.; and Liu, Y. 2005. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 880–885.

Wellman, M. P. 1990. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence* 44:257–303.