Transfer Learning via Relational Templates

Scott Proper and Prasad Tadepalli *

Oregon State University
Corvallis, OR 97331-3202, USA
{proper,tadepall}@eecs.oregonstate.edu

Abstract. Transfer Learning refers to learning of knowledge in one domain that can be applied to a different domain. In this paper, we view transfer learning as generalization of knowledge in a richer representation language that includes multiple subdomains as parts of the same superdomain. We employ relational templates of different specificity to learn pieces of additive value functions. We show significant transfer of learned knowledge across different subdomains of a real-time strategy game by generalizing the value function using relational templates.

Key words: model-based, afterstates, transfer learning, relational reinforcement learning, Markov decision processes

1 Introduction

Transfer learning is defined as transferring knowledge learned from one domain to accelerate learning in another domain. In this paper, we argue that transfer learning can be viewed as generalization in a rich representation language over a superdomain that includes multiple subdomains as special cases. In particular, we consider a language that includes relational templates that may each be instantiated to yield pieces of an additive value function. Each piece of the value function may be applicable to a single subdomain, multiple subdomains, or to the entire superdomain based on its structure. The advantage of this approach is that transfer happens naturally through generalization.

In this paper we learn value functions for a model-based multiagent reinforcement learning problem. The value function is a sum of terms, which are entries in multiple tables. Each term is generated by instantiating a relational template with appropriate parameters. By indexing the table entries with types of units, we learn values that correspond to units of only certain types. By not indexing them with any type, one could learn more general terms that apply across all subdomains. Thus controlling the specificity of the relational templates controls the generalization ability and hence the transfer ability of the system.

We learn value functions for real-time strategy games that includes multiple instances of multiple types of units. Each subdomain consists of a small number

* We gratefully acknowledge the support of the Defense Advanced Research Projects Agency under grant number FA8750-05-2-0249.

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

of units of certain type. We train the system on a set of multiple subdomains, and test the system on a test domain. We show that in most cases, there is positive transfer. In other words after training from similar domains, the performance in the test domain raises more quickly than it would without such prior training on similar domains. However, we see some instances of negative transfer, and some cases where adding more domains to the training set decreases the rate of learning in the test domain. While the results support the thesis that transfer learning can be successfully viewed as generalization in a suitably rich relational language, it also suggests that if performance in a single test domain is the goal, it is often best to train it on a similar training domain.

2 Afterstate Total Reward Learning

We assume that the learner's environment is modeled by a Markov Decision Process (MDP), defined by a 4-tuple $\langle S, A, p, r \rangle$, where S is a discrete set of states, and A is a discrete set of actions. Action u in a given state $s \in S$ results in state s' with some fixed probability $p(s'|s,u)$ and a finite immediate reward $r(s,u)$. A *policy* μ is a mapping from states to actions. Here, we seek to optimize the total expected reward received until the end of the episode starting from state s. This is denoted by $v(s)$ and satisfies the following Bellman equation:

$$v(s) = \max_{u \in A} \left\{ r(s,u) + \sum_{s'=1}^{N} p(s'|s,u) v(s') \right\} \tag{1}$$

The optimal policy chooses actions maximizing the right hand side of this equation. We can use the above Bellman equation to update $v(s)$. However this is often computationally expensive because of high stochasticity in the domain. Thus, we want to replace this with a sample update as in model-free reinforcement learning. To do this, we base our Bellman equation on the "afterstate," which incorporates the deterministic effects of the action on the state but not the stochastic effects [1, 2]. Since conceptually the afterstate can be treated as the state-action pair, it strictly generalizes model-free learning. We can view the progression of states/afterstates as $s \xrightarrow{a} s_a \rightarrow s' \xrightarrow{a'} s'_{a'} \rightarrow s''$. The "$a$" suffix used here indicates that s_a is the afterstate of state s and action a. The stochastic effects of the environment create state s' from afterstate s_a with probability $P(s'|s_a)$. The agent chooses action a' leading to afterstate $s'_{a'}$ and receiving reward $r(s', a')$. The environment again stochastically selects a state, and so on. We call this variation of afterstate total-reward learning "ATR-learning". ATR-learning is similar to ASH-learning from [2], however we use total reward here instead of average reward. The Bellman equation for ATR-learning is as follows:

$$v(s_a) = \sum_{s'=1}^{N} \left\{ p(s'|s_a) \left[\max_{u \in A} \{ r(s', u) + v(s'_u) \} \right] \right\} \tag{2}$$

We use sampling to avoid the expensive calculation of the expectation above. At every step, the ATR-learning algorithm updates the parameters of the value

Table 1. Various relational templates used in experiments. See Table 2 for descriptions of relational features.

Template #	Description
#1	$\langle Distance(A,B), UnitHP(B), EnemyHP(A), UnitsInrange(B)\rangle$
#2	$\langle UnitType(B), EnemyType(A), Distance(A,B), UnitHP(B), EnemyHP(A), UnitsInrange(A)\rangle$
#3	$\langle UnitType(B), Distance(A,B), UnitHP(B), EnemyHP(A), UnitsInrange(A)\rangle$
#4	$\langle EnemyType(A), Distance(A,B), UnitHP(B), EnemyHP(A), UnitsInrange(A)\rangle$
#5	$\langle UnitX(A), UnitY(A), UnitX(B), UnitY(B)\rangle$

function in the direction of reducing the temporal difference error (TDE), i.e., the difference between the r.h.s. and the l.h.s. of the Bellman equation:

$$TDE(s_a) = \max_{u \in U(s')} \{r(s',u) + v(s'_u)\} - v(s_a) \tag{3}$$

3 Function Approximation via Relational Templates

In this paper, we are interested in object-oriented domains where the state consists of multiple objects or units O of different classes, each with multiple attributes. Relational templates generalize the tabular linear value functions (TLFs) of [2] to object-oriented domains [3]. As with TLFs, relational templates also generalize tables, linear value functions, and tile coding. A relational template is defined by a set of relational features over shared variables (see Table 1). Each template is instantiated in a state by binding its variables to units of the correct type. An instantiated template i defines a table θ_i indexed by the values of its features in the current state. In general, each template may give rise to multiple instantiations in the same state. The value $v(s)$ of a state s is the sum of the values represented by all instantiations of all templates.

$$v(s) = \sum_{i=1}^{n} \sum_{\sigma \in \mathcal{I}(i,s)} \theta_i(f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)) \tag{4}$$

where i is a particular template, $\mathcal{I}(i,s)$ is the set of possible instantiations of i in state s, and σ is a particular instantiation of i that binds the variables of the template to units in the state. The relational features $f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)$ map state s and instantiation σ to discrete values which index into the table θ_i. All instantiations of each template i share the same table θ_i, which is updated for each σ using the following equation:

$$\theta_i(f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)) \leftarrow \theta_i(f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)) + \alpha(TDE(s,\sigma)) \tag{5}$$

Table 2. Meaning of various relational features.

Feature	Meaning
$Distance(A,B)$	Manhattan distance between enemy A and unit B
$UnitHP(B)$	Hit points of an friendly unit B
$EnemyHP(A)$	Hit points of an enemy A
$UnitsInrange(A)$	Count of the number of units in range of (able to attack) enemy A
$UnitX(A)$	X-coordinate of units A
$UnitY(A)$	Y-coordinate of units A
$UnitType(B)$	Type (archery or infantry) of unit B
$EnemyType(A)$	Type (tower, ballista, or knight) of enemy A

where α is the learning rate. This update suggests that the value of $v(s)$ would be adjusted to reduce the temporal difference error in state s. In some domains, the number of objects can grow or shrink over time: this merely changes the number of instantiations of a template.

We say a template is more refined than another if it has a superset of features. The refinement relationship defines a hierarchy over the templates with the base template forming the root and the most refined templates at the leaves. The values in the tables of any intermediate template in this hierarchy can be computed from its child template by summing up the entries in its table that refine a given entry in the parent template. Hence, we can avoid maintaining the intermediate template tables explicitly. This adds to the complexity of action selection and updates, so our implementation explicitly maintains all templates.

4 Experimental Results

We performed all experiments on several variations of a real-time strategy game (RTS) simulation. We tested 3 units vs. a single, more powerful enemy unit. As the enemy unit is more powerful than the friendly units, it requires coordination to defeat. Units also vary in type, requiring even more complex policies and coordination. We implemented our RTS simulation on a 10x10 grid-world. The grid is presumed to be a coarse discretization of a real battlefield, and so units are permitted to share spaces. Units, either enemy or friendly, were defined by several features: position (in x and y coordinates, hit points (0-6), and type (see Table 3). We also defined relational features such as distance between agents and the assigned enemy unit, and aggregation features such as a count over the number of opposing units within range. In addition each unit type was defined by how many starting hit points it had, how much damage it did, the range of its attack (in manhattan distance), and whether it was mobile or not. Friendly units were always created as one of the weaker unit types (archer or infantry), and enemies units were created as one of the stronger types (tower, ballista, or knight).

Table 3. Different unit types.

Unit	HP	Damage	Range	Mobile
Archer	3	1	3	yes
Infantry	6	1	1	yes
Tower	6	1	3	no
Ballista	2	1	5	yes
Knight	6	2	1	yes

Friendly units had five actions available each time step: move in one of four directions, wait, or attack the enemy (if in range). Enemy units had the same options (although a choice of whom to attack) and followed predefined policies, either attacking if in range or approaching (if mobile). An attack at a unit within range always hits, damaging it and killing it (removing it from the game) if it is reduced to 0 hit points. Thus, the number of units is reduced over time. Eventually, one side "wins" by destroying the other, or a time limit of 20 steps is reached. We gave a reward of $+1$ for a successful kill of an enemy unit, a reward of -1 if an friendly unit is killed, and a reward of $-.1$ each time step to encourage swift completion. Thus, to receive positive reward, it is necessary

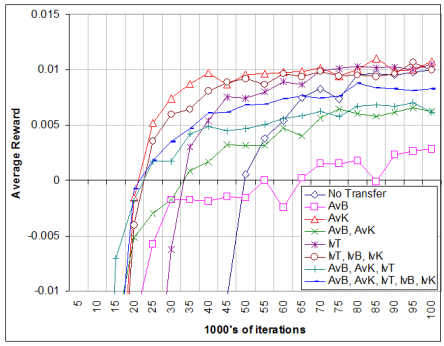

Fig. 1. Comparison of training on various source domains transferred to the Archers vs. Tower domain.

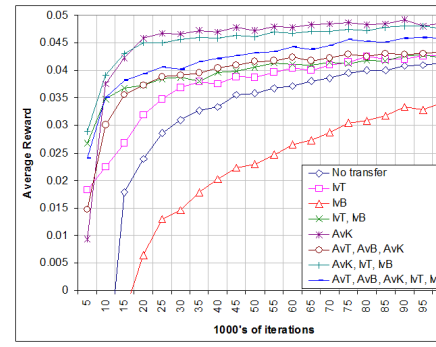

Fig. 2. Comparison of training on various source domains transferred to the Infantry vs. Knight domain.

for units to coordinate with each other to quickly kill enemy units without any losses of their own.

We used relational templates to define the value function (see Table 1). Template #1 is a "base" template which does not distinguish between types of objects and captures a generic function. We adapt this base template to create more refined templates which take into account differences between types of objects (see Tables 2 and 1). Templates #3-4 generalize across enemy unit types and friendly unit types respectively. Template #2 learns specialized domain-specific functions. Template #5 facilitates coordination between friendly units. Together, these templates create a "superdomain" capable of specializing to specific domains and generalizing across domains.

The results of our experiments are shown in Figures 1 and 2. Both figures show the influence that learning on various combinations of source domains has on the final performance of a different target domain (Archers vs. Tower or Infantry vs. Knight). We trained a value function for 10^6 steps on the various combinations of source domains indicated in each figure. Abbreviations such as "AvK" indicates a single kind of domain – Archers vs. Knights for example. Likewise I,T,B indicate Infantry, Towers, and Ballista respectively. When training multiple domains at once, each episode was randomly initialized to one of the allowable combinations of domains. We transferred the parameters of the relational templates learned in these source domains to the target "AvT" or "IvK" domains, and tested each target domain for 30 runs of 10^5 steps each, averaging the results of each run together. With all experiments conducted here, we used $\epsilon = .1$, $\alpha = .1$ and the ATR-learning algorithm.

Our results show that additional relevant knowledge (in the form of training on source domains that share a unit type with the target domain) is usually helpful, though not always. For example, in the IvK target domain, training on the IvB domain alone performs worse than not using transfer learning at all. However, training IvB and IvT together is better than training on IvT alone, and training on IvT is much better than no transfer at all. These results also

show that irrelevant information – training on the AvT and AvB domains, which do not share a unit type with the IvK domain – always harms transfer.

For the AvT target domain, transfer from any domain initially performs better than no transfer at all, but only a few source domains continue to perform better than no transfer by the end of each run. In both target domains the "AvK" source domain provides the best possible training for both target domains. The IvT, and IvT, IvB, and IvK source domains also perform well here.

5 Discussion

We have shown how relational templates may be refined from a "base" template that is applicable to all subdomains to more detailed templates that can specialize to particular subdomains based on the type features. By using several templates with different combinations of type features, we create a function approximator that generalizes between similar subdomains and also specializes to particular subdomains. This process allows for easy transfer of knowledge between subdomains.

Relational templates resemble the "class-based local value subfunctions" of [3], however in this paper we do not assume that relationships between objects, or even the exact numbers of objects in a domain, remain fixed throughout time. We also allow templates to reference multiple objects in the world. This flexibility makes relational templates a powerful and general function approximator.

Relational templates may also be used to transfer knowledge from small domains (such as that shown here) to larger domains with many units. We can do this using a technique called assignment-based decomposition [4], which decomposes the action-selection step of any reinforcement learning algorithm into an assignment level that assigns units to particular tasks, and a task execution level that chooses actions for units given their assigned task. This is similar to hierarchical reinforcement learning [5]. However we replace a joint value function at the root level with an approximate search technique over possible assignments.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. MIT Press (1998)
2. Proper, S., Tadepalli, P.: Scaling model-based average-reward reinforcement learning for product delivery. In Fürnkranz, J., Scheffer, T., Spiliopoulou, M., eds.: ECML '06. Volume 4212 of Lecture Notes in Computer Science., Springer (2006) 735–742
3. Guestrin, C., Koller, D., Gearhart, C., Kanodia, N.: Generalizing plans to new environments in relational mdps. In: IJCAI-03. (2003) 1003–1010
4. Proper, S., Tadepalli, P.: Solving multiagent assignment markov decision processes. In: AAMAS '09. (2009) 681–688
5. Makar, R., Mahadevan, S., Ghavamzadeh, M.: Hierarchical multi-agent reinforcement learning. In: AGENTS '01, Montreal, Canada, ACM Press (2001) 246–253