Utile Distinctions for Relational Reinforcement Learning

William Dabney and Amy McGovern
University of Oklahoma
School of Computer Science
amarack@ou.edu and amcgovern@ou.edu

Abstract

We introduce an approach to autonomously creating state space abstractions for an online reinforcement learning agent using a relational representation. Our approach uses a tree-based function approximation derived from McCallum's [1995] UTree algorithm. We have extended this approach to use a relational representation where relational observations are represented by attributed graphs [McGovern *et al.*, 2003]. We address the challenges introduced by a relational representation by using stochastic sampling to manage the search space [Srinivasan, 1999] and temporal sampling to manage autocorrelation [Jensen and Neville, 2002]. Relational UTree incorporates Iterative Tree Induction [Utgoff *et al.*, 1997] to allow it to adapt to changing environments. We empirically demonstrate that Relational UTree performs better than similar relational learning methods [Finney *et al.*, 2002; Driessens *et al.*, 2001] in a blocks world domain. We also demonstrate that Relational UTree can learn to play a sub-task of the game of Go called Tsume-Go [Ramon *et al.*, 2001].

1 Introduction

This paper introduces a method that combines the expressive power of a relational representation with the learning power of reinforcement learning (RL). The goal is to autonomously learn a relational state space approximation while simultaneously learning to act optimally. A relational representation enables an agent to reason at a higher level, which can allow an agent to learn in more difficult problems [Kaelbling *et al.*, 2001]. RL is an ideal technique for real-world control because it can learn to achieve a goal without being told how to accomplish it. There has been considerable recent interest in combining these techniques [Tadepalli *et al.*, 2004; van Otterlo, 2005].

The primary contribution of this paper is the introduction of an online RL method, Relational UTree, that can autonomously construct its own tree-based function approximation using a relational representation. UTree [McCallum, 1995] develops its own tree-based state space, allowing it to focus on the most important aspects of observations and to ignore irrelevant ones. Traditionally, a state representation is either designed by the user or learned from the policy.

Learning in a relational representation introduces two key challenges. The first is the exponential growth in search space and the second is that relational environments tend to have a high degree of autocorrelation. This has been shown to cause a selection bias that makes distinctions to appear useful when they are not [Jensen and Neville, 2002]. Relational UTree compensates for the size of the search space by using stochastic sampling [Srinivasan, 1999]. Autocorrelation violates the independent and identically distributed assumption made by many statistical techniques. We compensate for the effects of temporal autocorrelation by temporally sampling.

We separately address the need to adapt to changes in the environment by incorporating efficient tree restructuring [Utgoff *et al.*, 1997]. This allows Relational UTree to create more compact trees with the same representational power and helps to prevent overfitting in stochastic environments.

The most similar works are the G [Finney *et al.*, 2002; Chapman and Kaelbling, 1991] and TG algorithms [Driessens, 2004; Driessens *et al.*, 2006]. There are several major differences between TG, which is based in part on G-Algorithm, and Relational UTree. The primary difference is that Relational UTree can dynamically restructure its trees while TG must build a new tree from scratch. This allows Relational UTree to adapt to changes in the environment and to correct early mistakes made in the tree. While Relational UTree uses graphs to represent observations, TG uses first order logic predicates. In addition, TG creates a separate policy tree (called a P-tree) while Relational UTree derives its policy from the Q-value tree. Finney *et al.*'s [2002] approach is also based on G algorithm but uses a deictic representation, while Relational UTree makes use of a full relational representation and incorporates tree restructuring. We empirically compare the performance of Relational UTree to these approaches on blocks world environments.

2 Algorithm Description

The Relational UTree algorithm follows the UTree algorithm closely. We use the standard RL and partially observable Markov decision process (POMDP) notation where, at each time step t, the agent executes an action $a_t \in \mathcal{A}$, and receives an observation $o_{t+1} \in O$, and a reward $r_{t+1} \in \Re$ [Sutton and Barto, 1998; Kaebling *et al.*, 1998].

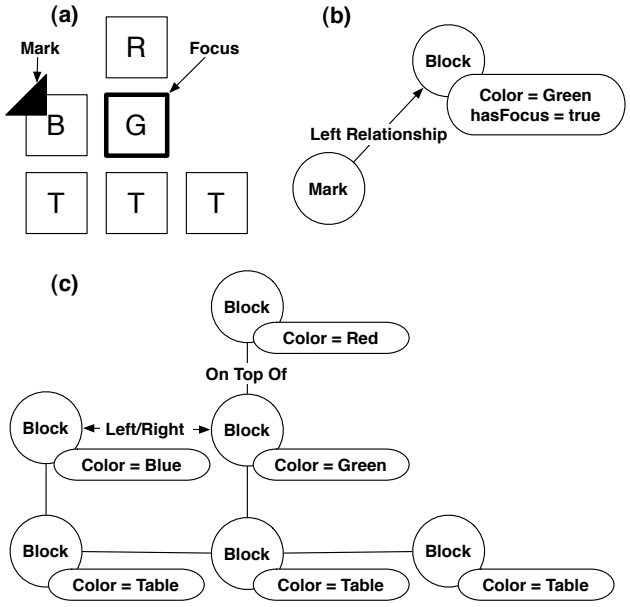

Figure 1: a) Example blocks world configuration. b) Partially observable *blocksworld1* [Finney *et al.*, 2002] observation. c) Fully observable *blocksworld2* [Driessens *et al.*, 2001] observation.

An observation takes the form of an attributed graph $G = \langle V, E, A(V), A(E), T(V), T(E) \rangle$. V is the set of all objects in the environment, represented by vertices in the graph and E is the set of all relationships, represented by edges [McGovern *et al.*, 2003]. $A(V)$ and $A(E)$ are the set of attributes for the vertices and edges respectively, the elements of which are discrete or real valued attributes. $A(V)$ and $A(E)$ may contain zero or more elements. $T(V)$ and $T(E)$ are required discrete valued *types* for vertices and edges. Example observations for blocks world are shown in Figure 1. Some attributes and relationships are omitted for readability. The graphical representation used does not place individual identifiers on objects, but instead objects are identified by their type, attributes, and relationships to other objects. This property of graphical representations is advantageous for generalizing concepts.

Our Relational UTree description uses McCallum's [1995] notations. We refer to the set of distinctions that lead to a tree node and the state represented by a leaf node as s. A transition instance, $T_t = \langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$, represents one step of the agent's experience. The set of instances contained in a leaf node s is denoted $\mathcal{T}(s)$, and the set of instances contained in a leaf node s where the next action is a is $\mathcal{T}(s,a)$. The time ordered set of all transition instances is $H = \{T_0, T_1, ..., T_{|H|}\}$. The leaf node to which a specific transition instance belongs is denoted by $L(T_t)$.

2.1 Relational Utile Distinctions

We refer to a distinction as being utile if and only if the distinction statistically separates the set of transition instances so that the agent is better able to predict reward. The sets of possible distinctions are: Object existence $\{(x,h) \mid x \in T(V), h \in \mathbb{N}\}$, relationship existence $\{(x,o_1,o_2,h) \mid x \in T(E), o_1 \neq o_2 \forall o_1, o_2 \in M(V), h \in \mathbb{N}\}$, and attribute value $\{(a,\ value_a, o_1, h) \mid (a, value_a) \in A(o_1), o_1 \in \{M(V) \cup M(E)\}, h \in \mathbb{N}\}$, where h is the history index, M is the memory, x is a parameter limiting the number of previous observations that can be considered, and $\mathbb{N}$ is the set of natural numbers.

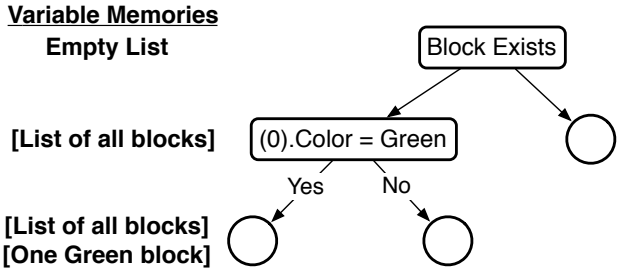

Figure 2: Example variable memory creation as the instance from Figure 1c is dropped down the tree.

The variables o_1 and o_2 are pointers to *variable memories*. Variable memories reference previous distinctions in s, allowing simple distinctions to be used to construct complex queries. For each type of distinction, the set of variable memories created when an instance is dropped down a node with that distinction is given by: $\{v \in V_i \mid T(v) = X\}$ for object distinctions, $\{(e, v_1, v_2) \in E_i \mid T(e) = X, v_1 \neq v_2, v_1, v_2 \in V_i\}$ for relationship distinctions, and $\{p \in V_i \cup E_i \mid (a, value_a) \in A(v), a = X, value_a = Y\}$ for attribute value distinctions.

Figure 2 shows an example of how Relational UTree uses variable memories. As the observation shown in Figure 1(c) falls down the root node, blocks are added to the variable memory. The second node selects blocks from this list with "Color = Green" and saves the matches.

Relational UTree allows static objects to be referenced so that their non-static attributes may be accessed. The set of static objects is defined as $S = \bigcap_H V_i$. A focal object, $f \in V$, can be incorporated to allow an agent to reason deictically. Let $M(V,N)$ be the set of object variable memories for the node N. At the root node, N_r, $M(V,N_r) = S \cup \{f\}$. The set of object variable memories created by the distinction at a given node, N, is denoted $m(V,N)$. If we let N_p be the parent node to N, then the set of object variable memories at any node, N, is defined as $M(V,N) = M(V,N_p) \cup m(V,N_p)$. We similarly define $M(E,N)$, given $M(E,N_r) = \emptyset$.

2.2 The Relational UTree Algorithm

The Relational UTree algorithm works as follows:

1. Create a tree with no distinctions. Initialize $\mathcal{T}(s) = \emptyset$ and $H = \emptyset$.
2. Take one step of experience in the environment. Choose the action a_{t-1} to be the policy action of $L(T_{t-1})$, with ε probability of choosing a random action. Record the experience as $T_t = \langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$ and $H = H \cup \{T_t\}$. Using standard decision tree methods, drop T_t down the tree. Save T_t to the leaf node, $L(T_t) = s$, setting $\mathcal{T}(s) = \mathcal{T}(s) \cup \{T_t\}$. Mark all tree node $N \in s$ as stale, where a

stale node indicates that the distinction at that node may need to be changed during restructuring.

In some environments, observations o_t and o_{t+1} are autocorrelated. For example, in the *blocksworld2* environment shown in Figure 1 (c), the objects and relationships will remain largely the same regardless of the next action performed. This is an example of temporal autocorrelation, and can cause a feature selection bias [Jensen and Neville, 2002]. In these situations, we remove autocorrelation through temporal sampling. Every c steps, T_t is forgotten, where c is a user defined value.

3. With the leaves of the tree representing the states, perform one step of value iteration using Equations 1 - 4. The equations for estimated immediate reward and estimated probability of arriving in state s' after executing action a in state s are given in Equation 3 and 4 and directly follow McCallum's equations.

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} Pr(s'|s,a)U(s') \tag{1}$$

$$U(s) = \max_{a \in \mathcal{A}} Q(s,a) \tag{2}$$

$$R(s,a) = \frac{\sum_{T_i \in \mathcal{T}(s,a)} r_i}{|\mathcal{T}(s,a)|} \tag{3}$$

$$Pr(s'|s,a) = \frac{|\forall T_i \in \mathcal{T}(s,a)\ s.t.\ L(T_{i+1}) = s'|}{|\mathcal{T}(s,a)|} \tag{4}$$

4. Update the tree every k steps by first ensuring the quality of current distinctions, followed by expanding the tree by adding new distinctions at the leaves. All stale tree nodes N are updated as follows.

   (a) First, stochastically generate a set of distinction trees Φ. Include the existing distinction tree, ϕ_N, in the set Φ, with $\Phi = \Phi \cup \{\phi_N\}$. A distinction tree is made up of one or more distinctions on a set of instances, organized into a tree structure. We consider distinction trees with depth up to some constant k, and consider larger depths only if no utile distinction trees are found. Each distinction tree $\phi \in \Phi$ defines a set of fringe leaf nodes $\mathcal{L}$.

   For each fringe leaf node $s \in \mathcal{L}$, the set of expected future discounted rewards for s makes up a distribution $\delta(s)$ given by Equation 5 [McCallum, 1995].

$$\delta(s) = \{r_{t_i} + \gamma U(L(t_{i+1})) \mid t_i \in \mathcal{T}(s)\} \tag{5}$$

   Calculate the Kolmogorov-Smirnov distance between each pair of distributions, denoted $KS(\delta_0, \delta_1)$. Let P_ϕ be the set of these p-values, given by Equation 6.

$$P_\phi = \{KS(\delta(s_i), \delta(s_j)) \mid \forall s_i, s_j \in \mathcal{L}\} \tag{6}$$

   Choose the best distinction tree from among Φ and the current distinction tree, ϕ_N, using Equation 7.

$$\phi' = \min_{\phi \in \Phi} \frac{\sum_{p \in P_\phi} log(p)}{|P_\phi|} \tag{7}$$

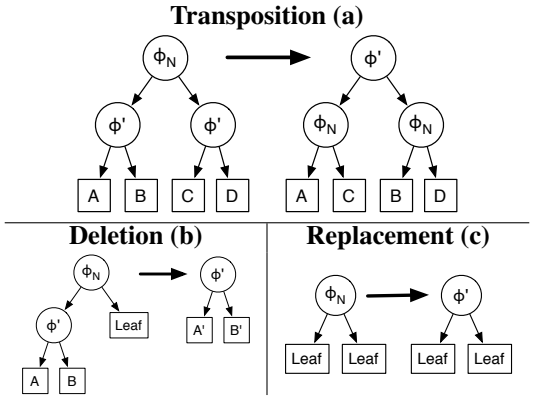

Figure 3: Let N be the current node, and $C(N) = \{N_1, N_2\}$ denote the children of N. **a)** $\phi_{N_1} = \phi_{N_2} = \phi'$. Perform tree transposition by setting $\phi'_{N_1} = \phi'_{N_2} = \phi_N$ and $\phi'_N = \phi'$. **b)** $\phi_{N_1} = \phi'$ and $C(N_2) = \emptyset$. Set $\phi_N = \phi'$ and $C(N) = C(N_1)$. Reclassify the instances $\mathcal{T}(N_2)$ using the N_0 subtree. **c)** $C(N_1) = C(N_2) = \emptyset$. Set $\phi_N = \phi'$, and use ϕ' to reclassify the instances $\mathcal{T}(N_1) \cup \mathcal{T}(N_2)$

   If there is no $p \in P_{\phi'}$ below the user specified value, then N is pruned from the tree. We used $p = 0.001$ for this paper. Otherwise, ϕ_N is replaced by ϕ' through a series of tree transpositions. For distinction trees of depth greater than one, the distinctions are 'pulled-up' one at a time, beginning with the root distinction.

   (b) Once the best distinction tree at a given node is determined, the tree is restructured following Utgoff *et al.* [1997]. If the current distinction at N is already the best, $\phi_N = \phi'$, then we are done. Otherwise, a recursive depth first traversal of the tree is applied until one of the base cases, shown in Figure 3, is reached. At this point, the utile distinction, ϕ', is "pulled-up" by recursively performing the tree transposition steps shown in the figure. After a single "pull-up" has been performed, the next base case in the tree is addressed until the best distinction tree, ϕ', is at the target node, N.

   When the tree restructuring operation has completed for N, mark it as not stale. If the node has changed through tree restructuring, then mark all nodes in its subtree as stale. Continue to apply step 4 to each stale child node of N. This process continues until no branches are stale, and the quality of the distinctions in the tree are ensured. Finally, perform value iteration until convergence.

5. Every k steps, expand the tree at the leaves. For each leaf node of the tree, s, determine the best distinction tree for the instances at that node using the process outlined in step 4 (a). If the new distinction tree is utile, then expand the tree by adding a subtree at the leaf s and dropping the instances $\mathcal{T}(s)$ down the new distinction tree. This removes s from the list of leaves and adds the set of leaves $\mathcal{L}$ to that list. Continue expanding the tree until it is no longer utile to do so. Perform value iteration until convergence after expansion is done.
6. Repeat at step 2 until stopped.

2.3 Stochastic Sampling

We use stochastic sampling [Srinivasan, 1999] to address the large space of possible distinctions introduced with a relational representation. Srinivasan shows that if we can sample tests stochastically that we can look at only a small fraction of the total and still be highly confident in finding one that is among the best possible. The number of tests that must be sampled is given by $n \geq \frac{\ln(1-\alpha)}{\ln(1-k)}$, where α is the probability of finding a test in the top $(100 \times k)\%$ [Srinivasan, 1999]. The key to this equation is that the sample size does not depend on the number of possible distinctions.

For this paper, we have used $k = 0.01$ and $\alpha = 0.99$ to be 99% confident in finding a distinction in the top 1%. In this situation we only need to sample 458 distinctions. By gradually reducing k, we can move the search for distinctions toward an exhaustive search. With our values, it took under a second to expand a leaf node containing 5000 instances. When we reduce k to 0.001, it takes two minutes to do the same expansion. Similarly, with $k = 0.0001$ the expansion time for the leaf node is almost 13 minutes. This demonstrates that an exhaustive search of the distinction space is not feasible. To compensate, we use stochastic sampling.

2.4 Tree Restructuring

In the early stages of learning, an agent will have only seen a fraction of the environment and may create a state representation that is not well suited to the true nature of the environment. To prevent the tree from over-fitting and to allow it to adapt to changing environments, Relational UTree implements online tree restructuring based on the Iterative Tree Induction algorithm [Utgoff *et al.*, 1997]. Iterative Tree Induction ensures that the best split for the instances is used at each node of the tree, beginning with the root and working its way to the leaves.

The original ITI algorithm kept track of a list of statistics for each possible test at every tree node. Because Relational UTree is instance based, this would be redundant. While ITI looked at the list of statistics for a node to decide the current best test at that node, Relational UTree regenerates tests for the node and decides which is best directly. Keeping track of all possible tests is not a practical solution in this situation because of the large search space for distinctions in relational environments. This large search space is a well known problem for inductive logic programming [Dzeroski *et al.*, 2001]. Although it is more computationally expensive to recompute the tests, restructuring leads to significantly smaller trees which reduces overall computation time.

3 Experimental Results

3.1 Blocks World Results

We apply Relational UTree to two versions of the blocks world domain. The first, *blocksworld1*, is a partially observable blocksworld task with low-level actions and two blocks, as in Finney *et al.* [2002]. The second, *blocksworld2*, is a fully observable domain with high-level relational actions and three blocks, as in Driessens *et al.* [2001]. Both domains contain moveable colored blocks and unmovable table blocks. Example observations for both domains are shown in Figure 1.

In the *blocksworld1* domain, the agent has a focus marker and a deictic marker. The agent perceives all attribute information for the focused object but none for the marked object. The marker is only observable if it is adjacent to the focus marker. For example, if the marker is below the focus block, then the agent will observe a block object, a marker object, and a relationship indicating the block object is on the marker object. The small difference from Finney's domain arise from the translation from a deictic representation to a truly relational one. The goal is to pick up the green block, which requires first removing any blocks covering it. The actions available to the agent are identical to that of Finney: *move-focus(direction)*, *focus-on(color)*, *pick-up()*, *put-down()*, *marker-to-focus(marker)*, and *focus-to-marker(marker)*. The agent is given a reward of +10 for reaching the goal, a penalty of -0.2 for invalid moves and -0.1 per time step.

The second domain, *blocksworld2*, is fully observable and the task is to stack the green block on the red block. The actions are *move(x,y)* for unique blocks x and y. As with Driessens, the agent received a reward of 1 for reaching the goal in the minimal number of steps and a 0 otherwise.

The performance of Relational UTree with and without tree restructuring for both domains is shown in Figure 4 (left panels). Empirical performance is ε-optimal, with $\varepsilon = 0.10$, for both domains. These results are averages across 30 runs. Because this is online performance, large changes to the tree's structure are reflected as temporary drops in performance. This happens less frequently as the tree converges.

In the more difficult *blocksworld1* domain, Relational UTree converges faster than Finney's approach with more accurate results. After 100,000 training instances Finney's performance was approximately 68% with final convergence to 80%. Our method converges to 90% of optimal, due to the use of ε-greedy exploration methods, and does so within the first 100,000 steps.

Comparatively, the actions in the *blocksworld2* domain are higher level which allows the agent to discover the goal much faster than in *blocksworld1*. The performance of Driessens' TG algorithm on the same domain was comparable. TG does not explore the environment, and does not use an ε-greedy exploration method like we do. This allows their algorithm to converge to an average reward of 1, while we converge to an average reward per trial of 0.9.

Figure 4 (right panels) compares the tree sizes with and without tree restructuring for the two domains. In both domains, performance is comparable but the average tree size is considerably smaller when tree restructuring is used. The agent is able to construct a smaller tree to capture the same amount of information because it can remove irrelevant information gained early in learning. The use of restructuring introduces an increased variance due to the temporary loss in performance directly following a large tree restructuring. However, smaller trees result in a significant improvement in running time.

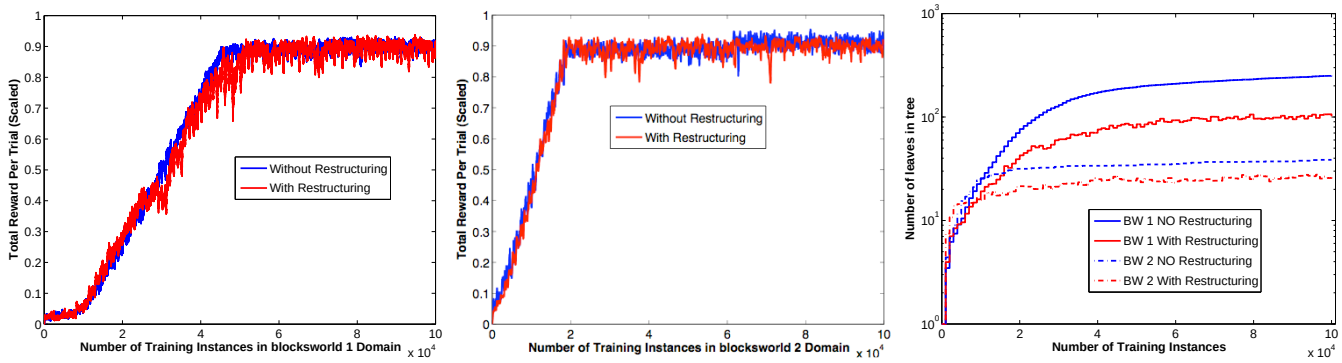

Figure 4: Learning curves for Relational UTree in the *blocksworld1* and *blocksworld2* domains. Tree sizes, with and without tree restructuring, are shown for both domains on the right.

3.2 Autocorrelation

To detect potential temporal autocorrelation, we used randomization on sets of observations. We perform 10000 randomizations on this data, each time performing a Kolmogorov-Smirnov test. The test statistics form a distribution which can be analyzed to find the effective sample size, similar to what was done with χ^2 by Jensen and Neville [2002]. If there is no autocorrelation, the effective sample size should match the actual sample size used in the tests.

The Kolmogorov-Smirnov test compares two distributions, of sizes $N1$ and $N2$. The total size of the data, $N1+N2$, remains constant throughout, while the proportion of data split into either distribution can change from one distinction to the next. In our experiment, the actual number of instances varied but the proportions that a specific distinction created was held constant. We use $p = \frac{max(N_1,N_2)}{(N_1+N_2)}$ to represent the relative sizes of the two distributions. By substituting p into the original equation for KS, given by [Sachs, 1982], we obtain $R = \frac{(2\times(K_\alpha)^2)}{D_\alpha}$ and $N_E = 2 \times \frac{R}{(4\times(p-p^2))}$ for the effective sample size N_E. K_α and D_α are the critical values for the KS test.

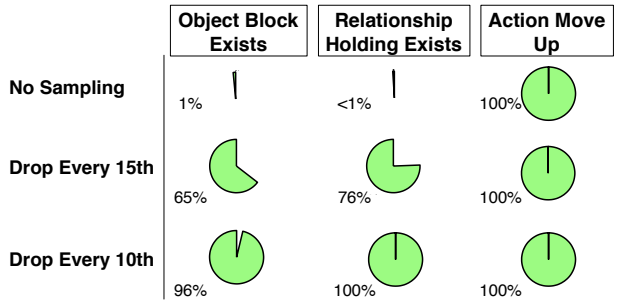

Figure 5: Effective sample size relative to actual sample size with variable amounts of sampling.

Figure 5 shows the results of our detection and removal of the temporal autocorrelation in the blocks world domain. The top pie chart for each group is the effective sample size without sampling. The lower two pie charts are for removing every 15th or every 10th instance. Effective sample sizes for an object existence, relationship, and action test are shown. The effective sample size is dramatically lower without sampling and small amounts of sampling dramatically increase the sample size. Tests on actions have no temporal autocorrelation because there is no direct correlation between what action will be performed and what the most recent action was. Since autocorrelation causes a feature selection bias [Jensen and Neville, 2002], Relational UTree used sampling to remove the autocorrelation.

3.3 Tsume Go Results

The Tsume Go domain is a sub-task of the game of Go where stones have already been placed in different configurations. The task is to identify the best move for protecting the player's stones or capturing the opponent's stones. Relational UTree learns to play the best first move of a Tsume Go game. Relational UTree is trained on a set of 30000 randomly sampled problems from the GoTools database [Wolf, 1996], and then tested on a set of 10000 randomly sampled independent problems from the same database. The agent receives a reward between 0 and 1 for correct answers depending on the quality of the move and is penalized -0.5 for incorrect attempts. It is also penalized with -1.0 for invalid moves such as placing a stone on top of another stone or trying to move off the board.

Similar to Ramon *et al.*'s [2001] approach, we test Relational UTree's approximation by ranking moves using the Q-values. Across 7 runs, Relational UTree averaged 65% accuracy on the test set after one move. Ramon *et al.*'s [2001] approach obtained 35% accuracy on a test set of 1000 problems generated by GoTools, after training on 2600 problems. We are encouraged by our results and are exploring this domain in greater detail as future work.

4 Discussion

Due to the significant performance difference between Finney and the large degree of similarities between our algorithm and Finney's, it is useful to discuss why Relational UTree was able to overcome some of the problems previously reported. Relying on observation history to disambiguate states and using state values to determine a policy before the states are disambiguated is a difficult problem that can be overcome through the use of tree restructuring. As the agent continues to learn, it carries older instances whose histories reflect outdated policies. As the agent's performance increases, the

vast majority of new observations will conform to an increasingly consistent policy. Given sufficient experience with the new policy, old observations will be in such a minority that splitting on them will no longer be statistically significant. Another method would be to remove the oldest memories of the agent when they are likely to no longer be relevant but this approach requires another parameter.

Finney's approach uses a combination of the G-Algorithm and UTree to learn with a deictic representation. In their approach, once the tree has been expanded at a node, all the instance information that caused that split is dropped. This is in contrast to the approach taken by Relational UTree, which saves instances and continues to use all observations to decide if a split is utile. We hypothesize that this difference is the cause of their convergence issues. Each split relies only on the most recent observations to see how the environment behaves. This could lead to incorrect assumptions if the current set of data is not representative of the entire environment.

Another problem encountered comes from the nature of POMDPs. If the agent in *blocksworld1* performed the action *focus_on_color(red)*, its history of observations would not tell it what state it was in. Finney suggests that history based decision tree algorithms will never be able to fully disambiguate the state space. While it is true that the agent's history would not help it to know the current state of the world, this information is not required for optimal behavior. Instead, the optimal behavior for this agent is to explore. Balancing the need to explore to discover information about the environment, and seeking out potential rewards, is a primary property of POMDPs [Kaebling *et al.*, 1998]. As such, the optimal policy would explore the environment, thus providing the agent with the useful historical observations that it needs.

5 Conclusions and Future Work

We have introduced Relational UTree, a significant modification to the UTree algorithm that automatically creates a tree-based function approximation in a relational representation. Relational UTree allows us to apply the advantages of the UTree algorithm to inherently relational environments without the need to convert into propositional logic. We demonstrated that Relational UTree is able to learn in a blocks world domain and on the task of Tsume Go. Relational UTree addresses the exponential growth in search space of a relational representation using stochastic sampling. We also demonstrated that temporal sampling was necessary to address the issues of autocorrelation that arise in a relational representation. We separately show that incorporating tree restructuring into Relational UTree gives it the critical ability to learn compact representations and to adapt to changing environments.

Current and future work focuses on applying Relational UTree to more complex relational domains, including the full game of Go. We are also exploring ways to improve the efficiency of storing and handling large numbers of observations. We are studying knowledge transfer and background knowledge using Relational UTree. The ability to approximate a state space independently of a human designer allows for many interesting future applications.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments. This material is based upon work supported by the National Science Foundation under Grant No. NSF/CISE/REU 0453545.

References

D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of IJCAI-91*, 1991.

K. Driessens, J. Ramon, and H. Blockeel. Speeding up relational reinfocement learning through the use of an incremental first order decision tree learner. In *Proceedings of ECML - European Conference on Machine Learning*, pages 97–108, 2001.

K. Driessens, J. Ramon, and T. Croonenborghs. Transfer learning for reinforcement learning through goal and policy parameterization. Presented at the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning, 2006.

K. Driessens. *Relational Reinforcement Learning*. PhD thesis, Department of Computer Science, K.U. Leuven, 2004.

S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):5–52, April 2001.

S. Finney, N. Gardiol, L. Kaelbling, and T. Oates. The thing that we tried didn't work very well : Deictic representation in reinforcement learning. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence*, pages 154–161, 2002.

D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceedings of the 19th International Conference on Machine Learning*, pages 259–266, 2002.

L.P. Kaebling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

L. P. Kaelbling, T. Oates, N. Hernandez, and S. Finney. Learning in worlds with objects. In *The AAAI Spring Symposium*, 2001.

A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

A. McGovern, L. Friedland, M. Hay, B. Gallagher, A. Fast, J. Neville, and D. Jensen. Exploiting relational structure to understand publication patterns in high-energy physics. *SIGKDD Explorations*, 5(2):165–172, December 2003.

J. Ramon, T. Francis, and H. Blockeel. Learning a tsume-go heuristic with TILDE. In *Proceedings of CG2000, the Second International Conference on Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 151–169. Springer-Verlag, 2001.

L. Sachs. *Applied Statistics, A handbook of techniques*. Springer, 1982.

A Srinivasan. A study of two probabilistic methods for searching large spaces with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.

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

P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *Proceedings of the ICML workshop on Relational Reinforcement Learning*, 2004.

P. E. Utgoff, N.C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:5–44, 1997.

M. van Otterlo. A survey of reinforcement learning in relational domains. Technical Report TR-CTIT-05-31, CTIT Technical Report Series, ISSN 1381-3625, 2005.

T. Wolf. The program gotools and its computer-generated tsume go database. Technical report, School of Mathematical Science, 1996.