The Thing We Tried That Worked: Utile Distinctions for Relational Reinforcement Learning

William Dabney AMARACK@OU.EDU
Amy McGovern AMCGOVERN@OU.EDU
School of Computer Science, University of Oklahoma, Norman, Ok 73019

Abstract

This paper introduces a relational function approximation technique based on McCallum's UTree algorithm (McCallum, 1995). We have extended the original approach to handle relational observations using an attribute graph of observable objects and relationships (McGovern et al., 2003). Furthermore, we address the inherent challenges that arise with a relational representation. We use stochastic sampling to manage the search space (Srinivasan, 1999), and sampling to address issues of autocorrelation (Jensen and Neville, 2002). We prevent the algorithm from growing an overly large and complex tree by incorporating Iterative Tree Induction's approach (Utgoff, 1995). We compare Relational UTree's performance with similar relational learning methods (Finney et al., 2002) (Driessens et al., 2001).

1. Introduction

It is well known that a relational representation can allow an agent to learn and plan at a higher level of abstraction than a propositional representation (Kaelbling et al., 2001). Many real-world learning tasks are relational in nature, and call for learning methods that can work in relational environments. Reinforcement learning is an ideal machine learning technique for real-world control because it is capable of learning the specified goal without being told how to accomplish it. And while using reinforcement learning to actively learn in propositional environments is well understood, how to combine it with relational learning remains an open problem (Finney et al., 2002). Relational reinforcement learning promises to extend the applicable domains for reinforcement learning, and improve active learning performance in relational domains.

This paper introduces a novel method for combining relational learning with an established reinforcement learning algorithm. Relational UTree is an online relational reinforcement learning algorithm based on the UTree algorithm. We compare this new algorithm to other methods for learning in relational environments. Specifically Relational UTree shares similarities with TG-Algorithm (Driessens et al., 2001), and other relational learning techniques(Finney et al., 2002). The deictic G-Algorithm most resembles Relational UTree because both learn an internal representation for states using a tree of distinctions made on observations (Finney et al., 2002). The TG-Algorithm is an incremental learner which predicts Q-values of state action pairs using a regression tree (Dzeroski et al., 2001). Additionally, while both approaches are applied to blocks world, one is a partially observable learning task, and the other is a fully observable version of the domain. We apply Relational UTree to both domains and further discuss the similarities and differences of the algorithms in greater detail in subsequent sections.

UTree allows an agent to handle tasks when given both too much and too little perceptual information (McCallum, 1995). UTree develops its own tree-based state space, allowing it to focus on the most important aspects of observations and to ignore irrelevant ones. Relational UTree extends this algorithm and allows an agent to learn to identify salient aspects of complex relational tasks and to learn effectively online in complicated relational environments. In classical reinforcement learning, a state representation is either designed by the user, or learned separately from the policy. While providing an agent with prior knowledge has many benefits, it is important to identify how much prior knowledge is actually required to learn cer-

Presented at the ICML Workshop on Open Problems in Statistical Relational Learning, Pittsburgh, PA, 2006.

tain tasks. Relational UTree allows an agent to learn a state space approximation, and policies for these states simultaneously. This allows the agent to begin improving performance before the underlying state space approximation has converged, and to continue to utilize this knowledge once convergence is reached.

Learning in a relational representation introduces two key challenges that we address. The first is the exponential growth in search space. The second challenge is that relational environments tend to have a higher degree of autocorrelation, which has been shown to cause a selection bias that can cause some distinctions to appear utile when they are not (Jensen and Neville, 2002). Relational UTree compensates for the exponential growth in the state space by using stochastic sampling (Srinivasan, 1999). Autocorrelation violates the independent and identically distributed (i.i.d.) 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 the efficient tree restructuring approach from Iterative Tree Induction (Utgoff, 1995). This allows Relational UTree to create more compact trees with the same representational power. Restructuring prevents the problem of creating overly complex trees encountered in other tree based learning algorithms (Driessens et al., 2001).

2. Algorithm Description

The Relational UTree algorithm follows the UTree algorithm closely. We use the standard reinforcement learning (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).

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. $T(V)$ and $T(E)$ are the types for vertices and edges. $A(V)$ and $A(E)$ can be empty but type is required. Example observations for blocks world are shown in Figure 1 (Finney et al., 2002; Driessens et al., 2001).

Our Relational UTree description uses McCallum's notations (1995). We refer to the set of distinctions, that lead to a tree node, as s. When the node is a leaf node,

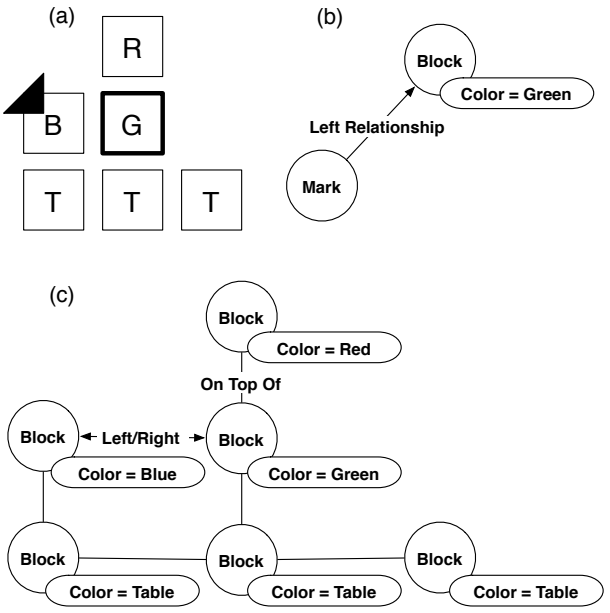

Figure 1. a) Example blocks world configuration. Bold outline and arrow indicate the agent's focus and additional marker location respectively. b) Partially observable *blocksworld1* observation. c) Fully observable *blocksworld2* observation.

the state represented by that leaf node is also referred to 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 $\mathcal{T}(s)$. 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 $\{(x, h) \mid x \in T(V), h \in \mathbb{Z}_{max}\}$, relationship existence $\{(x, o_1, o_2, h) \mid x \in T(E), o_1 \neq o_2 \forall o_1, o_2 \in M(V), h \in \mathbb{Z}_{max}\}$, and attribute value $\{(a, value_a, o_1, h) \mid (a, value_a) \in A(o_1), o_1 \in \{M(V) \cup M(E)\}, h \in \mathbb{Z}_{max}\}$, where h is the history index and M is the memory. Distinctions can be made on previous observations, up to a user defined maximum distance from the current one, $h \in \mathbb{Z}_{max}$.

The variables o_1 and o_2 are pointers to *variable memories*. Variable memories reference previous distinctions in s. For each type of distinction, the set of variable memories created when an instance is dropped

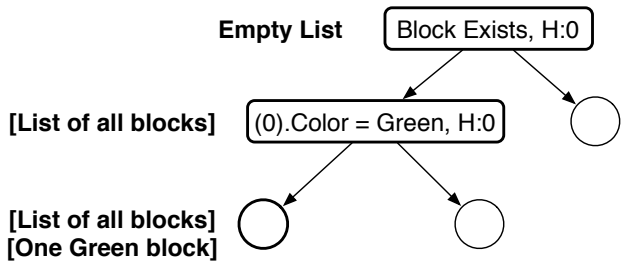

Figure 2. Example variable memory creation as an instance is dropped down the tree. Both distinctions have a history index of zero, H:0, and therefore are on the current observation.

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 which, among these blocks has the attribute value pair "Color = Green," and adds all matches to the list.

Relational UTree allows static objects to be referenced so that their non-static attributes may be accessed. The set of static objects, S, is defined as $S = \bigcap_H V_i$. A focal object, $f \in V$, is another useful construct which allows an agent to reason deicticly. 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 $\mathcal{T}(s) = \mathcal{T}(s) \cup \{T_t\}$. For every tree node $N \in s$ set $N.isStale = true$.

   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. Perform one step of value iteration (Bellman, 1957), with the leaves of the tree representing the states, using Equations 1 - 3. 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 2 and 3 and directly follow McCallum's equations. We denote the set of instances contained in a leaf node s, where the next action is a, as $(T)(s, a)$.

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

$$R(s, a) = \frac{\sum_{T_i \in \mathcal{T}(s,a)} r_i}{|\mathcal{T}(s, a)|} \quad (2)$$

$$Pr(s'|s, a) = \frac{|\forall T_i \in \mathcal{T}(s, a) \ s.t. \ L(T_{i+1}) = s'|}{|\mathcal{T}(s, a)|} \quad (3)$$

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. We denote a single distinction tree associated with the tree node N by ϕ_N. 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 4 (McCallum, 1995).

$$\delta(s) = \{r_{t_i} + \gamma U(L(t_{i+1})) \mid t_i \in \mathcal{T}(s)\} \quad (4)$$

To accurately calculate $\delta(s)$, the states' utility values must be updated. We currently perform value iteration to update these values for each distinction considered. However, due to the high computation cost involved, Prioritized Sweeping (Moore and Atkeson, 1993) is a potential replacement.

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 5.

$$P_\phi = \{KS(\delta(s_i), \delta(s_j)) \mid \forall s_i, s_j \in \mathcal{L}\} \quad (5)$$

Choose the best distinction tree from among Φ and the current distinction tree, ϕ_N, using Equation 6.

$$\phi' = \min_{\phi \in \Phi} \frac{\sum_{p \in P_\phi} log(p)}{|P_\phi|} \quad (6)$$

If any $p \in P_{\phi'}$ is above the user specified cut-off value, then the node N is pruned from the tree because the existing distinction, and any generated distinctions, are not utile. We used a p-value of 0.001 for the experiments in 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 (1995). 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, and if the node has changed through tree restructuring, then mark all nodes in its subtree as stale. Continue to apply step 4 to each child node of N, provided it is marked as stale. 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.

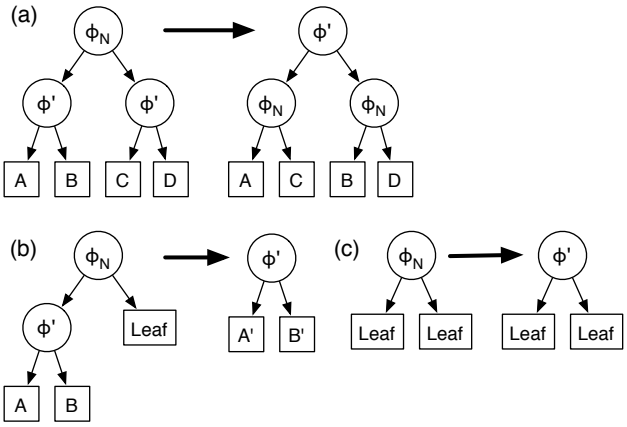

Figure 3. Let N be the current node, and $C(N) = \{N_1, N_2\}$ denote the children of N. **a)** $\phi_{N_1} = \phi_{N_2} = \phi'$. Perform tree transposition by setting $\phi'_{N_1} = \phi'_{N_2} = \phi_N$ and $\phi'_N = \phi'$. **b)** $\phi_{N_1} = \phi'$ and $C(N_2) = \emptyset$. Set $\phi_N = \phi'$ 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 $\mathcal{T}(N_1) \cup \mathcal{T}(N_2)$

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, through the same process outlined in step 4 (a). If the 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, 1995). 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 each node of the tree. 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. This is much more computationally expensive. However, keeping track of all possible tests is not a practical solution in this situation because of the very 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). We believe the advantages of restructuring compensate for the added computation time required to recompute distinctions.

3. Experimental Results

3.1. Task Descriptions

We apply Relational UTree to two versions of the blocks world domain. The first, which we will refer to as *blocksworld1*, is the focused deictic domain and task outlined by Finney et al. (2002). The second, referred to as *blocksworld2*, is the same domain used for the RRL-TG algorithm (Driessens et al., 2001).

In both domains there are multiple movable blocks as well as unmovable table blocks. The blocks have a color attribute, with possible values of "red," "blue," "green," and "table." Otherwise, the tasks, and what is observable, varies between the two domains.

In the *blocksworld1* domain, the agent has a focus marker, and one additional marker. The agent perceives all attribute information for the focused object but none for the marked object. If the marker is adjacent to the focus marker, then that marker is also visible to the agent, along with a relationship between the focus block and the marker. For instance, if the marker is below the focus block then the agent will perceive a block object, a marker object, and a relationship indicating the block object is on the marker object. This is slightly different from Finney's domain, but the difference is only due to a translation from a deictic representation into a truly relational one. The goal of the task is to pick up the green block, which requires first removing any blocks covering it.

The actions available to the agent for the *blocksworld1* domain are identical to the reference domain. They are *move-focus(direction)*, *focus-on(color)*, *pick-up()*, *put-down()*, *marker-to-focus(marker)*, and *focus-to-marker(marker)* (Finney et al., 2002). This domain is partially observable, and is run with two movable blocks. The agent is given a large positive reward (+10) for reaching the goal state, a small negative reward (-0.2) for invalid moves such as picking up blocks that are not clear, and a smaller negative reward (-0.1) per time step.

The second domain, *blocksworld2*, is fully observable and is run with three movable blocks. We examine the task of stacking two specific blocks on top of each other in this domain. The actions are *move(x,y)* for unique blocks x and y. For this domain, we must give blocks unique identifiers to remain consistent, we used the color attribute for this purpose. Then, we can describe the task as stacking the green block on top of the red block. A reward of 1 was given when the goal state was reached in the minimal number of steps, otherwise a 0 reward was given.

These two domains, while both versions of blocks world, are remarkably different. The *blocksworld1* is partially observable, with low level deictic actions. The *blocksworld2* domain is fully observable with very high level relational actions. An example observation for *blocksworld1* and *blocksworld2* was shown in Figure 1 (b) and (c) respectively.

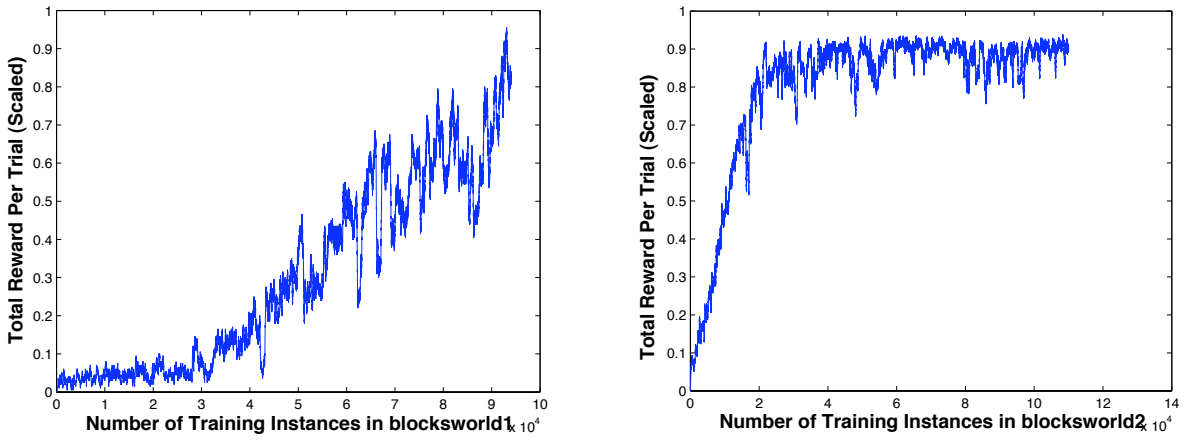

Figure 4. Learning curve for Relational UTree in both the *blocksworld1* and *blocksworld2* domains.

3.2. Blocks World Results

During online learning in the *blocksworld1* and *blocksworld2* domains Relational UTree built a state space approximation while performing value-iteration on that approximation. The resulting learning curves for both domains are shown in Figure 4. In both domains the performance achieved is ϵ-optimal, with $\epsilon = 0.10$. Performance is taken as an average across 20 training experiments for *blocksworld1* and 10 experiments for *blocksworld2*. Because this is actual online performance, large changes to the tree's structure are reflected as temporary drops in performance. Larger drops indicate that a substantial tree restructuring has occurred. This happens less and less frequently as the tree converges.

In the more difficult, partially observable, domain of *blocksworld1* the agent's performance improves gradually over the course of 100,000 training instances. This improves upon both the speed of convergence and the accuracy of Finney's performance in a comparable domain (Finney et al., 2002). After 100,000 training instances Finney's performance was approximately 68%, and eventually converges to approximately 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.

For the *blocksworld2* domain, our agent converges much faster than the *blocksworld1* domain, taking only 25,000 instances. The actions in this domain are higher level, and allow the agent to discover the goal much faster than in *blocksworld1*. The performance of Driessens' RRL-TG algorithm on the same domain was comparable (Driessens et al., 2001). RRL-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 that is 90% of optimal. This domain is a simpler version of the blocks world domain, but is able to demonstrate that Relational UTree can be applied to a fully observable domain and still retain its ability to learn an appropriate state space approximation.

When the tree restructuring portion of Relational UTree was disabled, we observed differences in performance. In the *blocksworld1* domain, performance mirrors that of Finney et al. (2002). In the *blocksworld2* domain, we observed that performance was almost identical to that of Figure 4. These findings suggest that the advantage given by tree restructuring is different depending on the domain, but the properties causing this remain to be explored.

3.3. Autocorrelation

To detect potential autocorrelation in our environment we used randomization on sets of observations. We perform 10000 randomizations on this set of data, each time performing a Kolmogorov-Smirnov test. The test statistics form a distribution which we then analyzed. The equation for calculating Kolmgorov-Smirnov is partially dependent on the sample sizes used. Thus, we used a reversal of this equation to calculate the effective sample size given the test statistic 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.

$$p = \frac{max(N_1, N_2)}{(N_1 + N_2)} \tag{7}$$

$$R = \frac{(2 \times (K_\alpha)^2)}{D_\alpha} \tag{8}$$

$$N_E = 2 \times \frac{R}{(4 \times (p - p^2))} \quad (9)$$

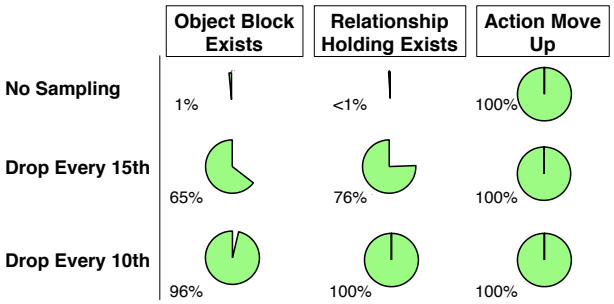

Figure 5. Effective sample size relative to actual sample size with variable amounts of sampling.

Equation 9 gives the effective sample size. This depends upon the proportion of instances in the two distributions being compared. In our experiment, the actual number of instances were varied, but the proportions that a specific distinction created was constant. Equation 7 shows the calculation of the proportion difference between the two distributions being compared. Using this value of p to replace the sizes of the distributions, we were able to directly reverse the equation for Kolmogorov-Smirnov's critical values (Sachs, 1982). This gave Equation 8 and Equation 9, thus providing N_E as the effective sample size.

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 result here is that the effective sample size is dramatically lower without any sampling, and small amounts of sampling are able to improve the situation. Also worth notice is that tests on actions have no temporal autocorrelation. This is because there is no direct correlation between what action will be performed and what the most recent action was. We already know that in highly autocorrelated environments a feature selection bias can cause distinctions to be incorrectly interpreted (Jensen and Neville, 2002). Therefore we used sampling as an effective means by which to remove the autocorrelation.

4. Discussion

In the partially observable domain, *blocksworld1*, we were able to obtain an ϵ-optimal policy with a relatively small tree. Unbounded tree growth was the primary cause of poor results in the *blocksworld1* domain by Finney et al. (2002). Relational UTree's use of restructuring slows down tree growth, and prevents it from growing overly complex trees. We suggest that the problems with non-convergence in Finney's approach are likely due to a tree that was not sufficiently discriminating. Without tree restructuring, it was forced to grow significantly larger trees to cope. Capping the tree at a fixed size did not address the issue as it was unable to converge.

Finney's approach uses the G-Algorithm and UTree to learn with a deictic representation. In Finney's algorithm, once the tree has been expanded at a node, all the instance information that caused that split is forgotten by the agent. This strikes a stark contrast to the approach taken by Relational UTree, which is instance based and continues to use all observations to decide if a split is utile. The discarding of old observations after each split prevents tree restructuring, and could also be a cause for increased tree growth. Each split relies only on the most recent observations to see how the environment behaves. This could lead to making incorrect assumptions if the current set of data is not representative of the entire environment.

For the fully observable domain of *blocksworld2*, our agent quickly found an optimal policy, as did the TG-Algorithm. However, Relational UTree's state space approximation is able to focus on the most important aspects of the fully observable environment. Driessens' approach is a very effective learner, but relies on a predefined state space available to the agent. Relational UTree automatically learns a state space approximation.

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 encountered.

The problem of relying on observation history to disambiguate states, and using state utilities to decide policy before they are disambiguated, is a difficult problem that can be overcome through the use of tree restructuring. As the agent continues to learn, it must carry older instances whose histories reflect outdated policies. The concern is that as the agent attempts to build an effective state space approximation, it will be forced to construct inefficient trees to explain these spurious observations. However, 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 in the leaves 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.

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 provide no clues as to what state it is in. Finney raises this problem and 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 allows a reinforcement learning agent to automatically create a useful 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 the blocks world domains of Finney and Driessens. Relational UTree addresses the exponential growth in search space of a relational representation using stochastic sampling (Srinivasan, 1999). This paper demonstrated that stochastic sampling was critical to learning a good function approximation in a reasonable amount of time. 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 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. We are currently using the Tsume-Go domain. We are also exploring ways to improve the efficiency of storing and handling large numbers of observations. In addition, we plan to apply Relational UTree to robotic applications. The ability to approximate a state space independently of a human designer allows for many interesting future applications.

6. 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

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Driessens, K., Ramon, J., and Blockeel, H. (2001). 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.

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

Finney, S., Gardiol, N., Kaelbling, L., and Oates, T. (2002). 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 (UAI-02)*, pages 154–161.

Jensen, D. and Neville, J. (2002). Linkage and autocorrelation cause feature selection bias in relational learning. pages 259–266.

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

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

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

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

Moore, A. W. and Atkeson, C. G. (1993). Memory-based reinforcement learning: Efficient computation with prioritized sweeping. In *In Proceedings of Advances in Neural Information Processing 5*.

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

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

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

Utgoff, P. (1995). Decision tree induction based on efficient tree restructuring. Technical report, University of Massachusetts.