

Automatic State Construction using Decision Tree for Reinforcement Learning Agents

Manix Au and Frederic Maire

Smart Device Laboratory, Centre for Information Technology Innovation,
Faculty of Information Technology, Queensland University of Technology,
2 George Street, Brisbane, Queensland 4001 AUSTRALIA
Email: f.maire@qut.edu.au , mj.au@student.qut.edu.au

Abstract

Reinforcement Learning (RL) is a learning framework for modelling an agent and its interaction with its environment through actions, perceptions, and rewards. Intelligent agents should choose actions after every perception, such that their long-term reward is maximized. A well defined framework for this interaction is the partially observable Markov decision process model (POMDP). Unfortunately solving POMDPs is an intractable problem. To overcome the problem of partial observability, McCallum introduced the U-tree, a RL algorithm that uses selective attention and short-term memory to simultaneously address the intertwined problems of large perceptual state spaces and hidden states. A U-tree embodies the policy and the state representation of the environment of the agent. A U-tree combines the advantages of instance-based learning with robust statistical tests for separating noise from task structure. In this paper, we consider an alternative approach for the feature selection of past events for the construction of the state representation. We apply information theory and decision tree techniques to derive a variation of the U-tree. The relevance of the candidate features is assessed by ranking the information gain ratio with respect to the cumulative expected reward. Experiments carried on three different RL tasks demonstrate that our variant of the U-tree produces a more robust state representation and faster learning. This better performance can be explained by the fact that the information gain ratio exhibits a lower variance in return prediction than the Kolmogorov-Smirnov statistical test used in the original U-tree algorithm.

1 Introduction

Reinforcement Learning (RL) is a computational approach to automating goal-directed learning and decision making. It is a problem description rather than a specific method [1,8,11,13]. An RL agent learns how to map situations (states) to actions to achieve some given tasks. The agent is not told which actions to take, but instead must discover the actions that provide long-term benefits, by trial and error.

In simple RL systems, the state space of the environment is discrete. For example, the state space in a tic-tac-toe game is the set of all the possible combination of circles and crosses on the board. In practice, many RL systems have a continuous state space (like the pole balancing problem). An environment may contain many features, amongst which some are irrelevant (like the colour and the size of the circles and crosses in a tic-tac-toe game), or the environment might be only partially observable [2,4,6,9] (some features of the environment are not perceivable). Partial observability can be caused by various reasons such as the limitations of the sensors, noise and occlusions. Under these circumstances, an agent is unable to disambiguate amongst the different states of the environment. Under partial observability, the current sensory information an agent receives may not unambiguously determine the state of the environment.

One way to deal with partial observability is to create an internal state representation dynamically constructed from the observation history [2,4,6,7]. In Section 2, we review such a method called U-tree. Then in Section 3, we introduce an alternative to U-tree. In Section 4, we present experimental results that demonstrate that our variation of the U-tree presents some interesting advantages over the original U-tree.

2 Automatic state construction with decision tree

U-tree [7] is a RL method designed to overcome the limitations of purely reactive policies. A U-tree allows the agent to extract relevant information, from current and past observations to create its own internal state representation. These internal states are an abstraction of the environment.

A U-tree is a decision tree that classifies the history (timely indexed sequence of observations) of an agent into a number of states. Every node of the U-tree represents a particular state of the agent.

The root node represents a single state with no distinction. The U-tree algorithm aims to extract relevant features to grow the tree. During the growth process, the state space representation is progressively refined. The most refined states correspond to the leaves of the U-tree, where action-value vectors are stored to represent the agent policy.

The U-Tree algorithm was designed for partially observable environments with large state space dimensions. A large state space dimension results in abundance of observations. Most of the observations are not task relevant and are not required for the internal state construction. Hidden states occur when the current observation alone is insufficient to determine the state of the environment [2,7]. Memory from previous observations is needed to augment the current perceptual input to reveal the hidden states. For example, a driver agent can be overloaded with information about the surroundings on the road. With respect to the task of driving safely, the agent needs only to be aware of the approaching vehicles and the traffic signals. Information such as the colour of a vehicle is irrelevant. Since it is impossible for the driver to look forward and backward simultaneously, the hidden state problem arises when the agent considers changing lanes. Using only the forward view percept, it cannot distinguish between states, which correspond to the presence and the absence of an approaching vehicle on the lane it wishes to change to. These two undistinguished states are said to be hidden. In order to change lanes safely, the agent needs to consider information gathered from previous perceptions to determine the hidden state.

A history table records the observations made at each time step. A U-tree classifies the history table (the input) into an internal state (the class label).

In a U-tree, each internal node corresponds to a test on a feature, which consists of a observation f and its history index lag . The history index allows a form of short term memory by specifying the lagging in history of the feature to be tested at a given node. Leaf extension is periodically carried out to discover relevant features that could be used to grow the tree. A pool of candidate features cf is available at each

leaf. When an appropriate cf_{win} is selected to extend a leaf, the leaf is replaced by a stump (tree of depth one). In other words, the internal state represented by the extended leaf is partitioned into a set of more refined internal states (the leaves of the stump). The associated timely indexed observations, including the returns (cumulative rewards), will be distributed into the new leaves according to the value of the feature cf_{win} . The distributional differences found between the return at a leaf and the returns at the new leaves after the introduction of a candidate feature cf measures the suitability of cf for refining an internal state. In the original U-tree, the Kolmogorov-Smirnov (K-S) Two Sample test is used to provide hypothesis testing on the distributional difference in return distributions.

3 The IGR U-tree

Our new variant of the U-Tree differs from the original U-Tree in the feature selection criterion; we have replaced the Kolmogorov-Smirnov (K-S) test with an Information Gain Ratio (IGR) test. The IGR test is the standard feature selection criterion for decision tree learning [10]. IGR provides a “class purity” measure that is an alternative to the K-S distributional difference test. If a feature increases significantly the information with respect to the returns, then this feature must be relevant to the environment state representation for the task assigned to the agent. The IGR is computed as follows; Given a set of discretized returns with possible values included in $\{u_1, u_2, \dots, u_m\}$ and a feature f with values $\{v_1, v_2, \dots, v_n\}$, we first estimate the probability $\Pr(R = u_j)$ for $j = 1, \dots, m$ from the history. Then we calculate the *Information* (also known as the entropy) $I(R)$, which measures the homogeneity of the returns

$I(R) = -\sum_{j=1}^m \Pr(R = u_j) \log_2 \Pr(R = u_j)$. Next, we calculate the *Information*

Gain $IG(R | f)$ (also known as the *conditional entropy*), which measures the expected reduction in information caused by partitioning the return R according to the feature

f . We have $IG(R | f) = I(R) - \sum_{k=1}^n \Pr(f = v_k) I(R | f = v_k)$. The Information of the

feature f is $I(f) = -\sum_{j=1}^n \Pr(f = v_j) \log_2 \Pr(f = v_j)$. The Information Gain Ratio is

$$IGR(R | f) = \frac{IG(R | f)}{I(f)}.$$

4 Experimental results

Three sets of experiments were conducted to compare the original K-S test U-tree and our IGR test U-tree. The test problems were a robot navigation problem, McCallum’s New York driving problem and an elevator control problem. The robot navigation problem involves an agent that learns to position itself to shoot a ball. The New York driving problem requires an agent to avoid vehicles by changing lanes safely [8]. The elevator control problem aim is to maintain passenger flow in a building for three elevators [15]. To allow automatic internal state construction, a set of candidate feature was pre-selected for feature extraction in each problem. This set of candidate feature is the product set of an observation set and a history index set. For example,

the candidate feature ‘observation X at lag 0’ denotes the current value of observation X; the candidate feature ‘observation Y at lag 2’ indicates the value of Y observed two time steps ago.

Training conditions; An episode terminates if the agent achieves the tasks or if the maximum iterations allowed is reached. The various common parameters used in the experiments are listed in the table below

Parameter	Description	Value
Exploration rate	Rate that indicates the probability of choosing a random action in a ϵ - greedy policy	$\epsilon = \begin{cases} 1 & \text{for ep 1} \sim 10 \\ 0.15 & \text{thereafter} \end{cases}$
Discount rate	Rate that discounts the future rewards in return computation	$\gamma = 0.7$
Learning rate for action value	Rate that determines the change ratio of the action values with respect to new experience	$\beta_q = 0.05$
Learning rate for action preference	Rate that determines the change ratio of the action selection preferences with respect to new experience	$\beta_p = 0.1$
Frequency for action value update	The regularity of updating the action values, expressed in terms of the number of episodes	$freq_q = 2$
Frequency for action preference update	The regularity of updating the action selection preferences, expressed in terms of the number of episodes	$freq_p = 4$
Frequency for internal state refinement	The regularity of refining the internal states, expressed in terms of the number of episodes	$freq_s = 5$
K-S test critical region	Indicates the threshold probability that test statistic must exceed to announce difference depicted	$p = 0.1$

The first 10 episodes are used for experience gathering. The U-tree begins its development at the end of the 10th episode and is checked for improvement every 5 episode. At the beginning of each episode, the agent observes its environment as its first action by default. When the U-tree exhibits no further development for a period of 40 episodes, stage 2 learning is initiated with the respective reinforcement function.

The following subsections describe each RL test problem in terms of the environment, the task, the action set, the reward, the candidate feature set and the training conditions.

4.1 The robot navigation problem

The environment; The RL system in the robot navigation problem is implemented using the Kiks simulator. The environment consists of a field, which is 1200 mm by 700 mm, a single goal, which is 300 mm wide and a ball, which is 90 mm in diameter. The position of the ball and that of the agent are randomly initialized.

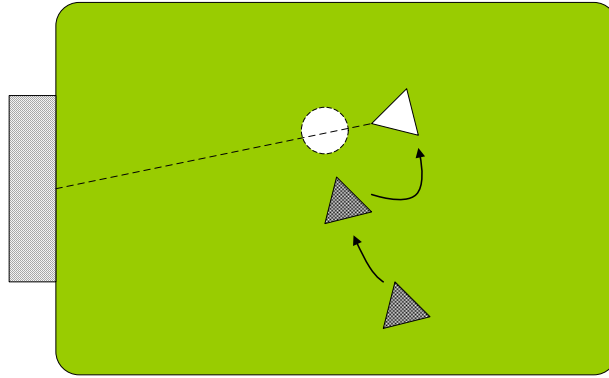


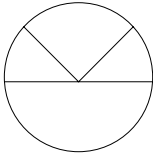
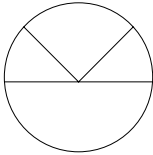
Figure 1: A snap-shot of the robot navigation problem

Task; The agent learns to get close to the ball during stage 1 of learning and align itself with the ball and the goal in a shooting position in stage 2. **Action Set;** The available actions are *move forward* (the agent moves forward by 10 mm), *turn left* (the agent turns left by 9 degrees), *turn right* (the agent turns right by 9 degrees) and *observe* (the agent perform a panoramic view to observe the position of the ball and that of the goal). **Reward;** Two distinct reinforcement functions are used for the two stages of learning.

Stage 1: $r = c_1 \cdot |width_{ball} - width_{desired}| + c_2$ where c_1, c_2 are constants

Stage 2: $r = c_3 \cdot width_{ball} + c_4 \cdot angle_{ball/goal} + c_5$ where c_3, c_4, c_5 are constants

Candidate feature set; The set of candidate feature is a product set of a feature set and a history index set. The feature set consists of the observations listed in the following table. They describe what the agent perceives on the soccer field. The history index set consists of $lag = \{0,1,2,3,4\}$.

Feature	Description	Value
Ball centre	The relative ball position 	0 if Unknown 1 if Left 2 if Front 3 if Right 4 if Rear
Ball width	The relative ball width	Piecewise constant function
Goal centre	The relative goal position 	0 if Unknown 1 if Left 2 if Front 3 if Right 4 if Rear
Goal width	The relative goal width	Piecewise constant function
Angle from goal to ball	The relative angle between the centre of the goal to that of the ball	Piecewise constant function
Previous action	The previous action taken	Move forward, turn left, turn right, observe

Last observe	An interval indication of the number of time step t pasted since the previous observe action	Piecewise constant function
Random number	A number randomly generated	$\{1, \dots, 10\}$

4.2 The New York driving problem

The environment; The environment is a one-way road, which consists of four lanes. The road traffic includes the agent's vehicle and other trucks. The agent travels at a speed of 16 meter per seconds. It has a visual horizon of 66 meters in front and behind. There are two types of trucks, the slow trucks and the fast trucks. The slow trucks travel at a speed of 12 meter per seconds and they appear in front of the agent's vehicle. The fast trucks travel at a speed of 20 meter per second and they appear from behind the agent's vehicle. All the trucks stay in their lanes.

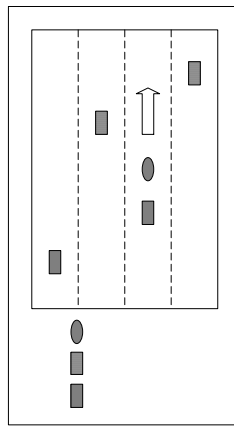


Figure 2: A snap shot of the New York driving problem

Task; The objective of the agent is to avoid colliding with the trucks as it moves forward. The agent tries not to run into slow trucks and not to be reached by fast trucks. When the agent runs into the rear of a slow truck, it performs a squeeze by scraping the side of the truck to move forward. When the agent is reached by a fast truck from behind, the fast truck begins to beep its horn until the agent moves out of its way. **Action set;** The available actions are *observe forward left* (look forward at closest truck in the left lane to the agent), *observe forward* (look forward at closest truck in the current lane of the agent), *observe forward right* (look forward at closest truck in the right lane to the agent), *observe backward left* (look backward at closest truck in the left lane to the agent), *observe backward* (look backward at closest truck in the current lane of the agent), *observe backward right* (look backward at closest truck in the right lane to the agent), *move to observed lane* (move to the lane previously observed).

Reward; The reward is

$$r = \begin{cases} 0.1 & \text{- clear progress} \\ -1 & \text{- honked} \\ -10 & \text{- squeeze} \end{cases}$$

Candidate feature set; The candidate features are listed in the table below.

Feature	Description	Value
Agent lane	The lane of the agent	{1,2,3,4}
Closest front	Distance of the closest truck in front of the agent	Piecewise constant function
Closest front left	Distance of the closest truck on the front left of the agent	Piecewise constant function
Closest front right	Distance of the closest truck on the front right of the agent	Piecewise constant function
Closest rear	Distance of the closest truck behind the agent	Piecewise constant function
Closest rear left	Distance of the closest truck on the rear left of the agent	Piecewise constant function
Closest rear right	Distance of the closest truck on the rear right of the agent	Piecewise constant function
Hear horn	True if the agent is honked	Yes, No
Previous action	The previous action taken	Any action
Random number 3	A number randomly generated	{1, 2, 3}
Random number 5	A number randomly generated	{1, 2, 3, 4, 5}

During the driving simulation, new trucks appear in randomly selected lanes and both types of trucks are equally probable to come into view.

4.3 The elevator control problem

The environment; The environment is a simulation of A 10 floor building with three elevators. Each elevator has a maximum capacity of 10 passengers and must stop on a floor to unload and upload any passenger.

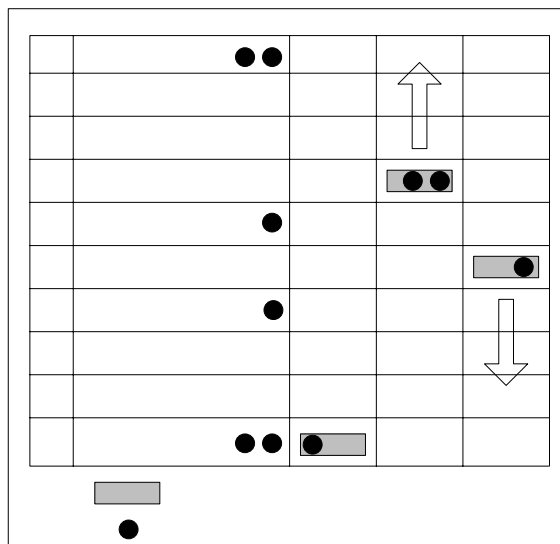


Figure 3: A snap shot of the elevator control problem

Task; The agent controls three elevators, and aims at bringing the passengers to their destinations quickly. **Action set;** The set of actions for the elevator central control is a triple product of the action set of each elevator. The three elevators are identical. The actions available for each elevator are described in the following table.

Action	Description
Stay	Stay on the current floor to upload and unload and passenger
Move up	Move up one floor
Move down	Move down one floor

Reward; The reinforcement function is negatively proportional to the total waiting time incurred by all passengers found in the building and the elevators. With a single passenger in the building, the reinforcement is given as follow.

$$r = \begin{cases} -t_{wait} & \text{if } t_{wait} \leq 40 \\ -3t_{wait} & \text{if } t_{wait} > 40 \end{cases}$$

where t_{wait} is the waiting time of a particular passenger. When more than one passenger is present, the reinforcement function of the system is expressed as the sum of the reinforcement contributed by each passenger. **Candidate feature set;** The list of candidate features is too long to be listed here. This list includes the motion and occupancy of each elevator, the presence of passengers on the current floor of each elevator, the presence of passengers waiting on floors above and below each elevator.

4.4 Performance comparison

The simulation experiments performed suggest no significant difference in performance between the K-S test U-tree and the IGR test U-tree with respect to the policies obtained at the end of training.

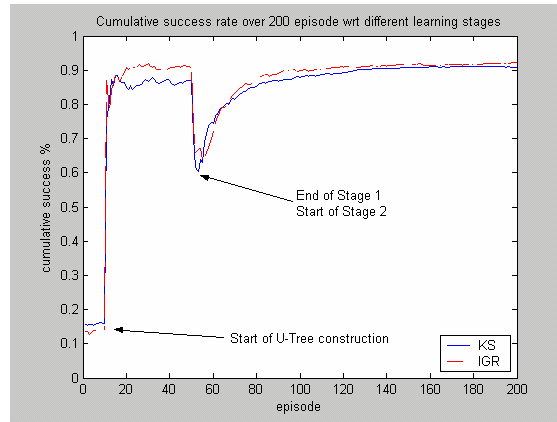


Figure 4: Performance comparison for the robot navigation problem

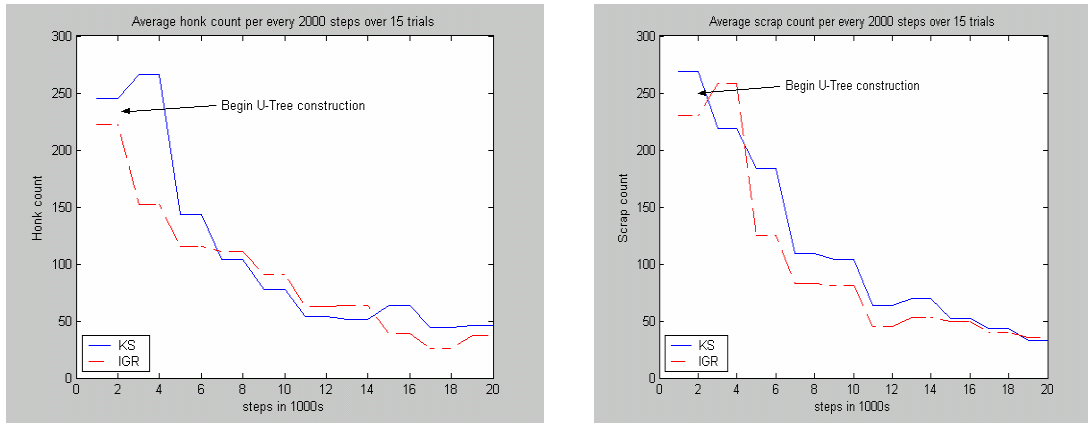


Figure 5: Performance comparison in average honk count and scrap count per every 2000 steps in the New York driving problem

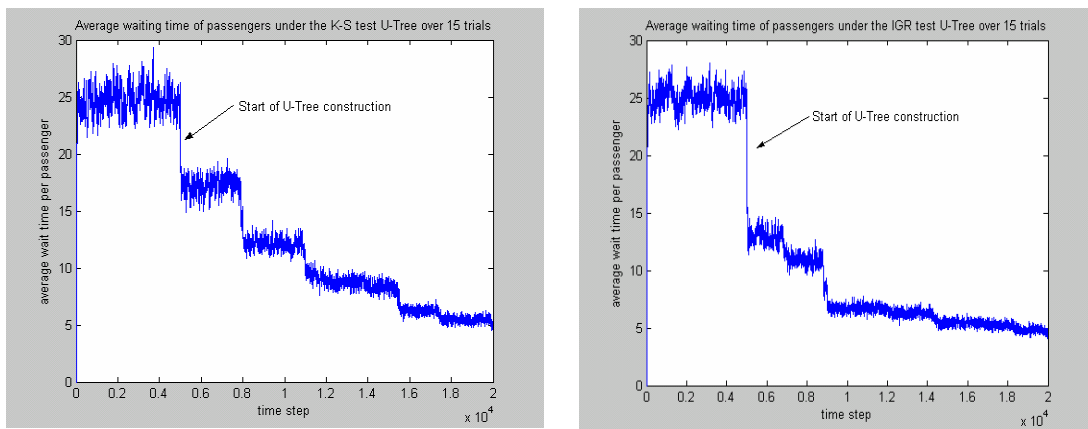


Figure 6: Average waiting times for passengers with the K-S test U-tree (left) and with the IGR test U-tree (right)

However, Figure 6 shows that our IGR test U-tree is capable to learn more rapidly than the K-S test U-tree when the experience collected is limited. To compare the suitable amount of experience needed for learning between the two tests, the internal state refinement frequency was reduced from 5 episodes (control) to 3 episodes in the robot navigation problem. Figure 7 shows that our IGR test U-tree outperforms the K-S test U-tree when the period for experience collection is shortened.

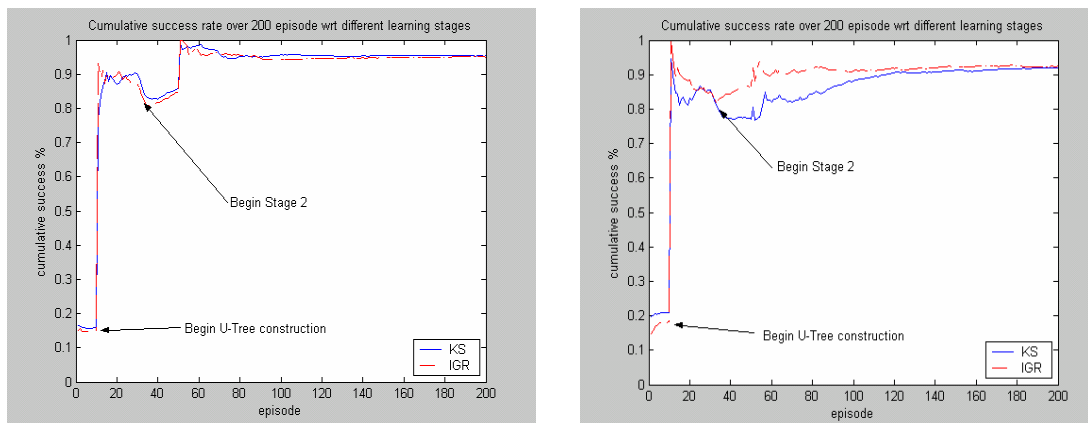
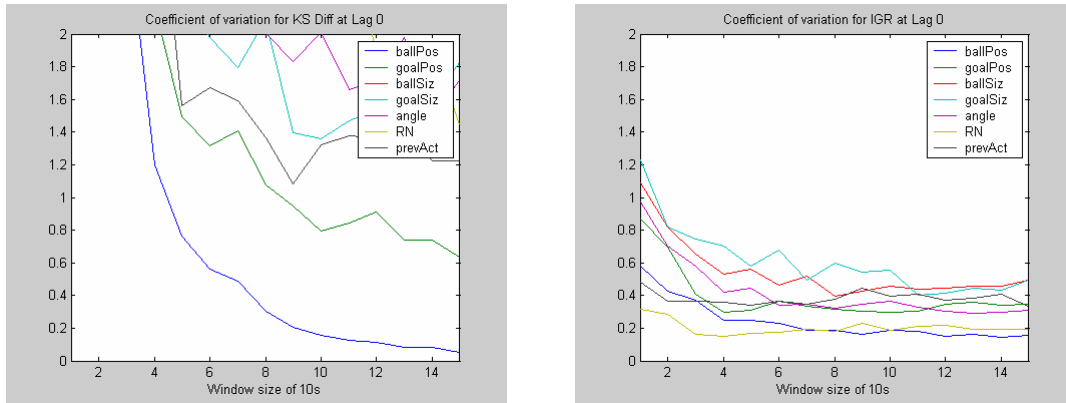


Figure 7: Performance comparison between the K-S test U-tree and the IGR test U-tree at the internal state refinement frequency = 5 (left) and at the frequency = 3 (right)

This better performance can be explained in terms of the variability in return prediction for the K-S and the IGR test. The diagram below on the left shows the coefficient of variation for the statistical differences obtained from a set of candidate features under the K-S test, given different episode length. And that on the right shows the mean of the information gain ratio obtained under the IGR test. First stage learning in robot navigation domain was used for simplicity. The current ball position is the only feature relevant for this task.



Although both tests have picked the correct feature, the K-S test exhibits greater variability when the period for experience collection is short.

5 Conclusion

The experiments performed suggest that both the K-S test and the IGR test U-tree can provide sound state construction functionality in discrete domains. The advantage of our IGR test U-tree is that the IGR test U-tree produces a more robust state representation and enables faster learning. This better performance can be explained by the fact that the IGR exhibits a lower variance in return prediction than the K-S test used in the original U-tree.

References

- [1] Aberdeen, D., (2002). *A survey of approximate methods for solving POMDPs*. RSISE, Australian National University.
- [2] Cassandra, A.R., Laelbling, L.P., & Littman, M.L. (1994). Acting optimally in partially observable stochastic domains. *Proceedings of the Twelfth National Conference on Artificial Intelligence*. Seattle, WA.
- [3] Crites, R.H., and Barto, A.G. (1996). Improving elevator performance using reinforcement learning. In D.S. Touretzky, M.C. Mozer, M.E. Hasselmo (eds.), *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pp. 1017-1023. MIT Press, Cambridge, MA.
- [4] Dutch, A. (1998). *Solving POMDPs using selected past events*. *Proceedings of the 14th European Conference on Artificial Intelligence*.
- [5] Kaelbling, L.P., Littman, M., & Moore, A. (1996). *Reinforcement Learning: A Survey*. *Journal of Artificial Intelligence Research*, 237-285

- [6] Lin, L.J., Mitchell, T.M. (1992). *Memory approaches to reinforcement learning in non-Markovian domains* (Technical Report CS-92-138). Carbegie Nellon, Pittsburgh, PA.
- [7] Lovejoy, W.S. (1991). *A survey of algorithmic methods for partially observed Markov decision processes*. *Annals of Operations Research*, 28, 47-65.
- [8] McCallum, A.K. (1995). *Learning to Use Selective Attention and Short-Tern Memory in Sequential Tasks*. *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, The MIT Press, pp. 315—324.
- [9] McCallum, A.K. (1996). *Reinforcement learning with selective perception and hidden states*. Doctoral dissertation, University of Rochester.
- [10] Murphy, K.P. (2000). *A Survey of POMDP Solution Techniques* (Technical Report). Dept. of Computer Science, U.C.Berkeley.
- [11] Quinlan (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann.
- [12] Sutton, R.S., & Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. Cambridge MA: MIT Press.
- [13] Uther, W.T.B., & Veloso, M.M., (1996). *Tree Based Discretization for Continuous State Space Reinforcement Learning*. *Proceedings of AAAI-98, Madison, WI, 1998*.
- [14] Walkins, C.J.C.H., &Dayan, P. (1992). *Q-learning*, *Machine Learning*, (pp249-292).
- [15] Singh, S.P., Jaakkola, T., & Jordan, M.I. (1995). *Reinforcement learning with soft state aggregation*. *Advances in Neural Information Processing Systems* (pp. 361-368). The MIT Press.